Toward Vignette-Based Story Generation for Drama Management Systems

Mark O. Riedl
Institute for Creative Technologies
University of Southern California
Marina Del Rey, California, USA
riedl@ict.usc.edu

Carlos León
Depto. de Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
Madrid, Spain
cleon@fdi.ucm.es

ABSTRACT

Drama management is a technique by which a virtual world and computer-controlled characters within the virtual world are manipulated in order to bring about a dramatic experience for the human participant. Generative drama management systems use artificial intelligence to dynamically generate new story content in order to adapt to the participant. Story generation poses many challenges, one of which is ensuring that story content that is generated is good. In the absence of computational models of story aesthetics, we present an approach to story generation that makes use a library of vignettes – short narrative segments – that are presupposed to be "good." We introduce a story planning algorithm inspired by case-based reasoning that incorporates vignettes into the story being generated. The story planning algorithm requires that vignettes be in the appropriate domain. We further describe an approach for automatically translating vignettes from one domain to another, using analogical reasoning.

1. INTRODUCTION

An interactive story is an approach to interactive entertainment in which a system attempts to tell a story to an interactive participant, such that the user is afforded the opportunity to make decisions that directly affect the direction and/or outcome of the story. The goal of an interactive story system is to balance seemingly competing requirements: *plot coherence* and *perceived user self-agency* [17]. Plot coherence is the idea that the events that occur in a narrative have meaning and relevance to each other and to the outcome. Perceived user self-agency is the idea that the user, while immersed in a virtual world, perceives his or herself to be capable of making meaningful decisions.

One approach to interactive story is to use *drama management*. A drama manager (first introduced by [7]) is an agent that attempts to coerce a virtual world so that a player's interactive experience to conform to some pre-existing dramatic aesthetic. In the absence of a computational aesthetic, many drama management systems rely on a branching story [17] or plot graph [12, 18, 20, 24]. A branching story is a story-like artifact that incorporates decision-point at which an interactive participant can make choices and receive alternative story elements. A plot graph is a data structure that defines a set of partially ordered story elements that are available for the drama manager to choose to present to the interactive participant. One of the noted limitations of drama management is the exponential complexity of authoring branching stories [17]. That is, for every decision-point in a branching story, the amount of story content that must be authored is multiplied by b, where b is the branching factor.

An approach to drama management that may mitigate the authoring complexity is *generative drama management* (or *generative experience management* [15] for non-dramatic contexts). The generative drama management approach to interactive story suggests that if authoring branching stories is intractable for human authors then a computer system can generate story content in response to the actions and decisions of an interactive participant. The goal is to have the participant experience being part of an unfolding story. As the participant exerts his or her self-agency and deviates from the original story, the drama manager invokes an automated story generation system to adapt, modify, or re-generate story content. Through this technique the participant's actions and decisions are accommodated while the system simultaneously maintains a coherent narrative experience (according to some pre-existing aesthetic or set of attributes). Computers are very useful for performing recursive and repetitive tasks. Generative drama management systems include [1], [15], and [26].

In this paper we describe an approach to story generation that uses a combination of planning and case-based reasoning-like technologies to generate stories. This approach to story generation can be used for plan-based generative drama management systems such as those described in [15] and [26]. In Section 2 we discuss how knowledge-intensive planning techniques can be used to mitigate the lack of computational models of story aesthetics. We introduce a vignette-based planning algorithm inspired by case-based reasoning. The vignette-based planning approach, however, requires that vignettes have relevance to the characters and setting of the new story being generated. In Section 3 we describe a technique for transforming vignettes so as to be useful in the generation of new stories. In Section 4, we provide a brief discussion of related work. In Section 5, we present future work and conclusions.

2. COMPUTATIONAL AESTHETICS FOR STORY GENERATION

Generative drama managers rely on a story generation system to create, either *a priori* or in real time, the different possible story branches that can define a participant's experience in a virtual world. There are many approaches to story generation (e.g., [5, 9, 10, 13, 16, 22]). We favor a plan-based approach to story generation in the support of interactive story systems. Story generation is a hard problem that is not yet completely solved. Despite recent research in plan-based narrative generation (c.f., [14, 16]), narrative generation research remains hampered by the lack of a computational aesthetic that can be used by a generation system to evaluate the goodness/badness of a generated narrative (The problem of defining computational aesthetics may be more

challenging than the problem of story generation [14]). Heuristic functions, both handcrafted and learned, have been investigated as means to control a drama manager's real-time decision-making (c.f., [12, 18, 20, 24]). However, very different heuristics are required for story generation.

One way to circumvent the lack of aesthetics and heuristics is to rely on previous knowledge. A "knowledge intensive" approach [5] is to story generation is to use the fact that stories are not written in vacuums. That there are large amounts of knowledge about previously authored stories – some of which are known to be "good" – that may have relevance to new stories to be generated. The idea that one can learn from previous experience is the basis tenant of case-based reasoning

We assume the existence of a knowledgebase populated with vignettes – short segments of story – that represent "good" examples of common situations that occur in stories. For example, the knowledgebase would contain one or more specific instances of bank robberies, betrayals, cons, etc. It is important to note that the knowledgebase would include specific examples of these situations instead of general templates. The implication of the existence of this knowledgebase for story generation is that the story generator does not need to "reinvent the wheel," and thus does not need enough specialized knowledge to be able to create specialized narrative situations. Instead, a story generator can use the knowledgebase to splice in an example of an existing bank robbery that is known to be "good." There are many challenges that need to be addressed in order to achieve this ability in story generation of which we will discuss one: the transfer of vignettes from one domain to another to support vignette reuse in story generation.

2.1 Near and Far Transfer

Case-based reasoning typically involves some or all of the following stages:

- **Retrieve** – A process of retrieving one or more cases that solve problems similar to the current one.
- **Reuse** – A process of using (or adapting) a retrieved case in order to satisfy the new problem.
- **Revise** – A process of testing the solution to the new problem and repairing it, if necessary.
- **Retain** – A process of storing the new solution in order to be useful for future problem solving.

Further discussion of case-based reasoning is beyond the scope of this paper.

We define *near transfer* as the retrieval and reuse of a vignette that is already in the domain of the new story. For example, if the new story is about the characters Butch Cassidy and the Sundance Kid (American legendary Western outlaws) and there is a vignette that describes a bank robbery that was performed by Butch Cassidy and the Sundance Kid, then reuse is relatively trivial. Some revision may still be necessary to fit the vignette into the new story seamlessly due to particulars. If the vignette is about other characters but of the same roles, then near transfer still applies because reuse may simply involve adapting the vignette by substituting names of characters playing the roles.

We define *far transfer* as the retrieval and reuse of a vignette that is in a domain other than that in which the new story being generated is set. For example, if the new story is about 1920's American gangsters in Chicago and the applicable vignette is the same Western bank robbery from before, then the vignette must be adapted to fit the new domain.

The following definitions will be used throughout the remainder of the paper.

A domain $D = \langle S, \Lambda \rangle$ is a tuple such that S is a description of the state of a world and Λ is the set of all possible operations that can change the world.

Unlike the STRIPS and ADL representations of plan operators, we assume that Λ is the set of all ground operators, meaning the operators do not reference variables. The set of ground operators can be derived from STRIPS and ADL representations by substituting all valid combinations of ground symbols for all variables in all operator definitions. The use of ground operators is a necessity our vignette transformation algorithm (described in Section 3) and is merely a representational convenience that otherwise does not change how planners work in general.

A narrative plan $p = \langle I, A, O \rangle$ is a tuple such that I is a description of the initial state, A is a set of ground operators – called actions – and O is a set of temporal ordering constraints of the form $a_1 < a_2$ where $a_1, a_2 \in A$ and a_1 necessarily precedes a_2 in the story.

A narrative plan $p = \langle I, A, O \rangle$ is said to be of domain $D = \langle S, \Lambda \rangle$ if $a_i \in \Lambda$ for all $a_i \in A$ and $I \subseteq S$. Vignettes in the knowledgebase are assumed to be instances of story plans. Vignettes can be represented as incomplete plans, meaning that the plan is not sound. For example, a bank robbery vignette is not necessarily a complete story and cannot stand alone without actions that establish some of the conditions necessary for the vignette actions to occur.

A *minimal vignette* is a vignette (and thus represented as a story plan structure) such that the removal of any one action causes the vignette to no longer be considered a "good" example of the narrative situation it is intended to represent. We assume that the knowledgebase is populated to minimal vignettes.

2.2 Planning with Vignettes

Partial order planners and their kin have been used in plan-based story generation and plan-based interactive story systems (e.g., [1], [15], and [26]). The partial order planning algorithms we work with generate plans through a backward-chaining process of flaw revision. An *open condition flaw* exists when an action in the plan (or the goal state) has a precondition that is not established by a preceding action or the initial state. Partial order planners can repair this flaw by choosing one of the following repair strategies:

(i) Selecting an existing action in the plan that has an effect that unifies with the precondition in question.

(ii) Selecting and instantiating an operator from the domain operator library that has an effect that unifies with the precondition in question.

Our planning algorithm, the Vignette-Based Partial Order Causal Link (VB-POCL) planner, is a modification of standard partial order planners to take advantage of the existence of a knowledgebase of vignettes. The VB-POCL planning algorithm is similar to other case-based planners such as [2] and [4]. VB-POCL, like other case-based planners adds a third strategy for repairing open condition flaws:

(iii) Retrieve and reuse a case that has an action with an effect that unifies with the precondition in question.

The VB-POCL strategy for retrieving and reusing a case works as follows. Given an action in the plan that has an unsatisfied precondition VB-POCL non-deterministically chooses one of the three above strategies. Strategies (i) and (ii) are performed in the standard way (c.f., [23]). If strategy (iii) is selected, VB-POCL creates a new flaw, called a *fit flaw*. This flaw is satisfied only when all the actions in the retrieved vignette have been instantiated in the plan. Repairing a fit flaw is a process of selecting an action from the retrieved vignette and adding it to the new plan (or selecting an existing action in the plan that is identical to the selected action to avoid unnecessary action repetition). It may take several iterations of the planning algorithm to completely repair a fit flaw. This process may additionally lead to the creation of new open condition flaws that in turn are repaired through conventional planning (strategies i and ii) or by retrieving new vignettes (strategy iii). One of the interesting properties of case-based planning algorithms such as this is that they can operate when there are no applicable vignettes available; the algorithm can fall back on conventional planning. If applicable vignettes are available, plan-space search control algorithms are required to prevent a potential explosion of open conditions. If a vignette is retrieved, temporal and causal reasoning ensures that that vignette's actions are fit into the new plan at the appropriate place so as to preserve plan soundness.

VB-POCL relies on certain assumptions. First VB-POCL assumes that vignettes are *minimal*. VB-POCL doesn't stop refitting a vignette until all actions in the vignette are present in the new plan; the implication of this assumption is that discarding any one action, even if strictly extraneous from a causal perspective, will ruin the intended impact of the vignette. Second, VB-POCL assumes that vignettes in the library are in the domain of the story being generated. The implication of this assumption is that the planner does not need to deliberate about the tradeoff between the cost of retrieval and reuse (which is otherwise very high). VB-POCL *non-deterministically* chooses between flaw repair strategies (i and ii) and (iii), meaning that it applies *all* strategies to each and every flaw. This is not practical if vignettes require extensive far transfer adaptation.

The assumption that all vignettes are in the correct domain is strong and not necessarily always valid. As noted before examples of narrative situations can come from a wide assortment of domains. Further, it is desirable to invent new domains in order to tell new stories. Adding new characters or new operators to an existing domain necessarily results in a new domain. In the next section, we describe a technique for using analogy to transform story plans of one domain into story plans of another domain. This is part of a pre-processing stage in which all vignette plans in a vignette library (presumably of numerous and arbitrary domains) are transformed into vignette plans of a single, given domain.

3. ANALOGICAL TRANSFORMATION OF VIGNETTES

In our approach to story planning with vignettes, we require a library of vignettes in the domain of the new story to be generated. Since this is an unrealistic restriction, we have created a pre-processing phase that transforms vignettes in one domain into new vignettes in a target domain – the domain that the new story to be generated will be in. To engage in far transfer on vignettes, one must first find analogies between the source domain and the target domain. Analogy-finding algorithms such as the Structure-Mapping Engine (SME) [3] and Connectionist Analogy Builder (CAB) [8] have been demonstrated to be able to find analogies in stories (e.g., the "Karla the Hawk" story described in [3] and [8]) when they exist. Our problem is different: we have vignettes in a source domain, but no vignettes in the target domain; we are solving the problem of transforming a vignette in a source domain into a new vignette in a target domain.

CAB [8] is an implementation of a cognitive model of analogy that finds correspondences between concepts. Concepts are represented as nodes in a directed graph such that concepts that are related in some way are adjacent. Given two graphs, CAB produces a mapping with the analogies it has found between nodes. In the next section we show how we use CAB to transform a vignette in a source domain into a new vignette in a target domain.

3.1 The Vignette Transformation Algorithm

Using CAB for computing the analogies, we have developed an algorithm that receives, as input, a vignette in the source domain and information about the target domain, and creates, as output, a new vignette, analogous to the first one, in the target domain. That is, we perform far transfer on all vignettes in a library.

We define a plan operator as a tuple <*h*, *P*, *E*>, where:

- *h* is the head of the action, defining its name and its arguments. For example, for the action *take(prince, sword)*, we have the name *take,* and the ground arguments *prince* and *sword*, meaning that the prince took the sword.
- *P* is the set of ground preconditions, a set of propositions that define the previous state needed for the action in the story to be performed.
- *E* is the set of ground effects, that is, the set of propositions that are made true when the action is performed. For example, after the action *take(prince, sword)* is applied, the new state would contain the proposition *has(prince, sword)*.

CAB works with graphs, and thus we need to translate STRIPS-like operators to graphs. To translate an operator into a graph we create a root node with the head of the action, and children nodes for the arguments of that action. We add as children to the root node a *preconditions* node and an *effects* node, whose children are the propositions of the sets *P* and *E*, respectively. The graph is completed by adding propositions from the domain state information. State information provides context about the ground symbols that is essential for finding correct analogies between operators. Some operators have very similar structures (e.g., propositions that become negated) and CAB would be unable to find the difference between any two operators with surface-level structural similarity without additional information. An example of graph can be found in Figure 1. Gray nodes represent ground symbols. The white nodes at the bottom of the figure represent state information.

With this definition of the operators, the algorithm in Listing 1 finds a mapping for each operator in the source vignette. For efficiency we only transform the source domain operators that are actually used in the source vignette. The transformation algorithm iterates over the set of actions in the source vignette, and, for each one of them, finds the best possible analogous operator in the

Transform (p_s, D_t)
Given a vignette $p_s = <I, A, O>$ in the source domain, and target domain $D_t = <S, \Lambda>$ transform p_s into an analogous vignette.

Let $state_s = I(p_s)$
Let $state_t = S(D_t)$
Let $p_t \leftarrow <S(D_t), \varnothing, \varnothing>$
Let $map \leftarrow \varnothing$
Foreach $a_s \in A(p_s)$ **do**
    Let $a_t \leftarrow$ **find_best**(a_s, $\Lambda(D_t)$, $state_s$, $state_t$)
    $state_s \leftarrow$ **apply**($E(a_s)$, $state_s$)
    $state_t \leftarrow$ **apply**($E(a_t)$, $state_t$)
    $A(p_t) \leftarrow A(p_t) \cup \{a_t\}$
    $map \leftarrow map \cup \{a_s \Leftrightarrow a_t\}$
$O(p_t) \leftarrow$ **apply**(map, $O(p_s)$)
Return p_t

Listing 1. Vignette transformation algorithm.

target domain. Then, having the map between that source operator and the target operator, the algorithm updates the source and target domain state information, as if the respective operators had been performed, both in the source state and in the target state. This is carried out by applying the effects of the operators against their respective states. With the new source state and target state, we find the next mapping for the next action in the source vignette. This process is repeated until every action in the source vignette has been mapped to an operator in the target domain. It is important to update state information because, in each step of the algorithm, we compare operators that are partially defined by its actual state. Without updating the state, we would be comparing wrong information.

The *find_best* algorithm is responsible for finding the best operator in the target domain for an operator in the source domain. It is important to note that we have not developed a test for optimality, and thus there is not an exact way for finding the *best* mapping. However, we refer to the "best" or "optimal" operator when, intuitively, that operator would be chosen by a human.

We implement *find_best* as a single-elimination competition of target domain operators. The algorithm is shown in Listing 2. The routine compares a source operator with a randomly chosen pair of target operators. Paired target domain operators are merged into the same directed graph (see Figure 2 for an example), forcing CAB to choose which nodes make the best correspondences to nodes in the graph of the single source operator. This gives us information about correspondences between nodes in each graph and provides a metric for whether one target domain operator is *relatively* more analogical than another to the source domain operator. CAB maps the head of the source operator to the head of one of the target operators. The loser is discarded while the winner is matched against another randomly chosen target domain operator. This repeats until only one target domain operator remains. The single-elimination competition has been shown to be equally effective as algorithms that compare a source operator to all pairs of target domain operators while only requiring a linear number of comparisons.

Find_best is only as good as the structure-mapping algorithm, and CAB relies on the presence of contextual information in the graphs to make reliable analogies. The example in the next section shows how the system can find analogies even when

Find-Best (a_s, Λ_t, $state_s$, $state_t$)
Given source action a_s, target domain operators Λ_t, current source state $state_s$, and current target state $state_t$, find the best target action in Λ_t.

Let $g_s \leftarrow$ create graph from a_s and $state_s$
Let $winner \leftarrow$ choose and remove random element from Λ_t
While $\Lambda_t \neq \varnothing$ **do**
    Let $a_t \leftarrow$ choose and remove random element from Λ_t
    Let $g_t \leftarrow$ create graph from $winner$ and a_t and $state_t$
    Let $mapping \leftarrow$ **CAB**(g_s, g_t)
    $winner \leftarrow$ get the winner based on $mapping$
Return $winner$

Listing 2. Single-Elimination Competition algorithm.

operators have significantly different structures. See Section 5 for a brief discussion on ways to potentially improve analogy.

Once the algorithm is applied, we have the new vignette in the target domain. This new element can then be used by the story planner as a new operator in the working domain, thus using past information (the old vignette) as a source for new stories (the new vignette). Note that the resulting vignette is not guaranteed to be sound, but the planner can revise with conventional planning strategies after it splices the vignette into a new story plan being generated.

3.2 Example

We show how the vignette translation works on a very simple vignette with two operators being transferred to a domain in which we have three possible operators. In the vignette a prince takes a sword from the king, and then goes from the castle to the forest, as follows:

- *Take(Prince, Sword, King)*
- *Go(Prince, Castle, Forest)*

Part of the initial state information of the source vignette includes the fact that the prince and the king are in the castle, the king rules over the prince, and the king has the sword. An example of the *Take* action with some accompanying initial sate information is shown in Figure 1. In the target domain we have three possible operators: a soldier steals a gun from the general, the general leaves the camp, and the soldier leaves the camp:

- *Steal(Soldier, Gun, General)*
- *Leave(Soldier, Camp)*
- *Leave(General, Camp)*

The domain initial state information includes the following facts: the soldier and the general are at the camp, the general is the leader of the soldier, and the general has the gun. This set of operators has been chosen in order to show a simple example, but in a real translation, we would have a larger set of operators that includes more actions and all permutations of character arguments, including, for example, *Steal(General, Gun, Soldier)*.

The vignette translation algorithm executes as follows. The first action from the source vignette, *Take(Prince, Sword, King)*, is selected and combined with vignette initial state information. This operator is first compared with *Steal(Soldier, Gun, General)* and *Leave(Soldier, Camp)*, selected randomly from the target domain, and accompanying domain state information. Figure 2 shows the graph of the two target domain operators. CAB prefers the

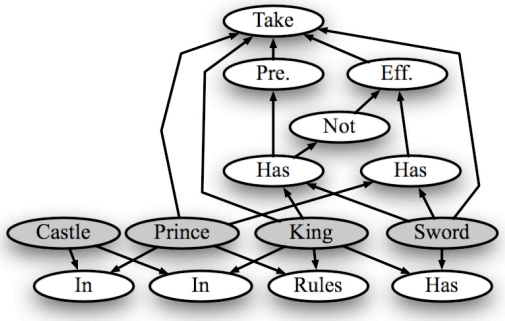

Figure 1. Graph representation of operator "Prince takes sword".

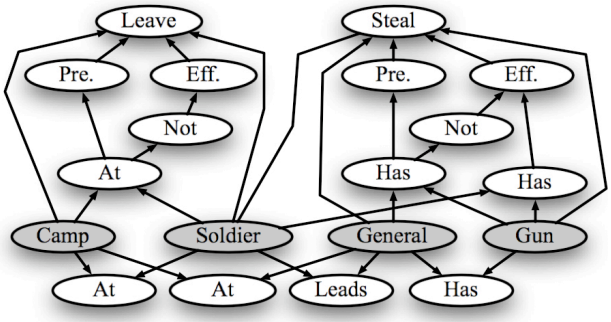

Figure 2. Graph of two competing operators, "Soldier leaves camp" and "Soldier steals gun".

correspondence between the head of *Take(Prince, Sword, King)* and the head of *Steal(Soldier, Gun, General)*.

The loser of that competition, *Leave(Soldier, Camp)*, is removed from the list of valid target operators. Next, *Take* is compared with the previous winner, *Steal*, and *Leave(General, Camp)*. Again, CAB prefers the correspondence of *Take* to *Steal*. Consequently, *Steal(Soldier, Gun, General)* is determined to be the best match for *Take(Prince, Sword, King)*. The effects of *Take* and *Steal* are applied to their respective domain states.

Now we repeat the same process for *Go(Prince, Castle, Forest)* with the new state information. First, *Go* is compared to *Steal(Soldier, Gun, General)* and *Leave(Soldier, Camp)*. Interestingly, the graph for the *Steal(Soldier, Gun, General)* operator is more similar to the graph for *Go* because *Go* and *Steal* both have three arguments, one precondition, and two effects of which one is the negation of one of the preconditions. However, the way in which the state information in the source graph correlates to the state information in the target graph causes CAB to prefer to correlate *Go(Prince, Castle, Forest)* with *Leave(Soldier, Camp)*. Next, comparing *Leave(Soldier, Camp)* and *Leave(General, Camp)*, CAB prefers to map the head of *Go* to the head of *Leave(Soldier, Camp)*. Again, the state information about the soldier and the general helps CAB distinguish between the Soldier and the General and thus find the right mapping. Consequently, *Leave(Soldier, Camp)* is determined to be the best match for *Go(Prince, Castle, Forest)*. The final transformed vignette is as follows:

- *Steal(Soldier, Gun, General)*
- *Leave(Soldier, Camp)*

4. RELATED WORK

The goal of this work is to augment the story creation ability of planner-based generative drama managers. Thus, this work has relevance to interactive narrative systems such as [1], [15], and [26]. Of direct relevance is related research on story generation (e.g., [5, 9, 10, 13, 16, 22]). Minstrel [22], ProtoPropp [5], and Méxica [13] use forms of case-based reasoning approaches to reuse of story elements. Minstrel uses specialized transformation procedures called ATRANS. ProtoPropp uses an ontology-based approach to CBR with some causal-link reasoning. Méxica is perhaps the most similar in that it reuses story elements and fills in missing elements with planning.

Case-based reasoning can also play a role in interactive story systems other than plot generation. CBR, as a model of individual agent reasoning, can also be used to select character behavior in non-drama-management approaches to interactive story [21], although discussion of this is outside the scope of this paper. CBR has also been explored in conjunction with player preference modeling to maximize interestingness when a drama manager can select among alternative plot points [20].

The planning algorithm presented in this paper is an adaptation of multiple-reuse case-based planning technologies such as [2] and [4]. One distinction between VB-POCL and other multiple-reuse case-based planners is that VB-POCL does not discard unnecessary actions from cases because of the minimal vignette assumption. VB-POCL shares many functional attributes with CBPOP [2]. However, CBPOP *deterministically* decides whether to attempt retrieval/reuse for any given flaw under the assumption that retrieval and reuse is costly and should be avoided whenever it seems unlikely to be effective. VB-POCL assumes retrieval and reuse is trivial due to the transformation of the entire vignette library to the target domain.

It may be possible to use hierarchical task network (HTN) planners [19] or a decompositional planner such as DPOCL [25] to achieve similar effects as VB-POCL. However, using HTNs or other decompositional techniques to generate story requires reasoning at higher levels of abstraction than the action (or event), and this introduces potentially rigid top-down structuring of plot that can limit opportunistic discovery such as in [14, 16].

The vignette transformation algorithm uses analogy to find mappings between actions in different domains. Analogical transformation has been used for story generation before. Hervás et al. [6] uses analogy to map the state of one domain to the state of another domain to construct informative expository statements such as "Luke Skywalker was the King Arthur of Jedi Knights." Our approach maps operators at which time, as a side effect, state is also mapped.

5. FUTURE WORK AND CONCLUSIONS

In order to make vignette-based story planning practical, we enforce the rule that vignettes must be close enough to the new story's domain that near transfer is possible and quick. However, this requires that vignettes undergo far transfer prior to invoking VB-POCL. This pre-processing stage, described in Section 3, is slow but only has to be done when a new domain is created. One interesting conclusion of our work to date is that the vignette transformation process can work with relatively little extraneous information. However, more knowledge is better and linking to WordNet [11] or other sources of ontological information will

increase the processing speed, provide more accurate results, and ease authoring of domain knowledge. The technique described in [6] is complimentary to our own and could contribute to our vignette transformation process by competently mapping characters between domains before an attempt at mapping domain actions is made.

This knowledge-intensive approach to story generation enables us to use pre-screened vignettes as the building blocks of new stories. There is no guarantee that a new story made up of assembled vignettes will be good. However, it may be possible to annotate vignettes with metadata that can further inform the story generator about when and how to incorporate vignettes into the story structure. Future work involves additional algorithms to reason about when to use vignettes versus other story generation techniques. Future work is also needed to determine the extent to which vignette-based story generation can scale.

This work represents a step toward more sophisticated plan-based story generation, in the absence of computational models of story aesthetics, by incorporating partial order planning approaches to story generation with case-based reasoning-like approaches to story generation. The eventual goal is to develop generative drama managers and experience managers that are more effective in adapting to the actions and intentions of interactive participants by generating better stories.

6. ACKNOWLEDGMENTS

The project or effort described here has been sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

7. REFERENCES

[1] Barber, H.M. and Kudenko, D. Dynamic Generation of Dilemma-based Interactive Narratives. In *Proc. of the 3rd AI and Interactive Digital Entertainment Conf.* (2007).

[2] Britanik, J. and Marefat, M. CBPOP: A Domain-Independent Multi-Case Reuse Planner. *Computational Intelligence, 20, 2,* (2004), 405-443.

[3] Falkenhainer, B., Forbus, K., and Gentner, D. The Structure-Mapping Engine: Algorithms and Examples. *Artificial Intelligence*, 41, (1989), 1-63.

[4] Francis, A.G., and Ram, A. A Domain-Independent Algorithm for Multi-Plan Adaptation and Merging in Least Commitment Planners. In *Proc. of the AAAI Fall Symposium on Adaptation of Knowledge Reuse* (1995).

[5] Gervás, P., Díaz-Agudo, B., Peinado, F., and Hervás, R. Story Plot Generation Based on CBR. *Journal of Knowledge-Based Systems*, 18, 4-5, (2005), 235-242.

[6] Hervás, R., Pereira, F.C., Gervás, P. and Cardoso, A. Cross-Domain Analogy in Automated Text Generation. In *Proc. of the 3rd Joint Workshop on Computational Creativity* (2006).

[7] Kelso, M., Weyhrauch, P., and Bates, J. Dramatic Presence. *Presence: The Journal of Teleoperators and Virtual Environments, 2,* (1993).

[8] Larkey, L.B. and Love, B.C. CAB: Connectionist Analogy Builder. *Cognitive Science, 27,* (2003), 781-794.

[9] Lebowitz, M. Story-Telling as Planning and Learning. *Poetics, 14,* (1985), 483-502.

[10] Meehan, J.R. *The Metanovel: Writing Stories by Computer*. Ph.D. Thesis, Yale University, New Haven, CT, 1976.

[11] Miller, G.A. Wordnet: A Lexical Database for English. *Communications of the ACM*, 38, 11, (1995), 39-41.

[12] Nelson, M., Mateas, M., Roberts, D.L., and Isbell, C. Declarative Optimization-Based Drama Management in Interactive Fiction. *IEEE Computer Graphics and Applications, 26, 3,* (2006).

[13] Pérez y Pérez, R. and Sharples, M. MEXICA: A Computer Model of a Cognitive Account of Creative Writing. *Journal of Experimental and Theoretical Artificial Intelligence*, 13, (2001), 119-139.

[14] Riedl, M.O. *Narrative Generation: Balancing Plot and Character*. Ph.D. Thesis, North Carolina State University, Raleigh, NC, 2004.

[15] Riedl, M.O., Stern, A., Dini, D., and Alderman, J. Dynamic Experience Management in virtual Worlds for Entertainment, Education, and Training. *International Transactions on Systems Science and Applications* (to appear).

[16] Riedl, M.O. and Young, R.M. An Intent-Driven Planner for Multi-Agent Story Generation. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi Agent Systems* (2004).

[17] Riedl, M.O. and Young, R.M. From Linear Story Generation to Branching Story Graphs. *IEEE Computer Graphics and Applications, 26, 3,* (2006).

[18] Roberts, D.L., Strong, C., and Isbell, C. Using Feature Value Distributions to Estimate Player Satisfaction Through an Author's Eyes. In *Proc. of the AAAI Fall Symposium on Intelligent Narrative Technologies* (2007).

[19] Sacerdoti, E.D. *A Structure for Plans and Behavior*. Elsevier, 1977.

[20] Sharma, M., Ontañón, S., Strong, C., Mehta, M., and Ram, A. Towards Player Preference Modeling for Drama Management in Interactive Stories. In *Proc. of the 20th Int. Conf. of the Florida Artificial Intelligence Research Society* (2007).

[21] Swartjes, I. Using Narrative Cases to Author Interactive Story Content. In *Proc. of the 6th Int. Conf. on Entertainment Computing* (2007).

[22] Turner, S. *The Creative Process: A Computer Model of Storytelling*. Lawrence Erlbaum Assoc., 1994.

[23] Weld, D. An Introduction to Least Commitment Planning. *AI Magazine, 15, 4,* (1994), 27-61.

[24] Weyhrauch, P. *Guiding Interactive Fiction*. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1997.

[25] Young, R.M., Pollack, M., and Moore, J. Decomposition and causality in partial-order planning. In *Proc. of the 2nd Int. Conf. on AI and Planning Systems* (1994).

[26] Young, R.M., Riedl, M.O., Branly, M., Jhala, A., Martin, R.J., and Saretto, C.J. An Architecture for Integrating Plan-Based Behavior Generation with Interactive Game Environments. *Journal of Game Development, 1*, (2004).