Delayed Roles with Authorable Continuity in Plan-based Interactive Storytelling

David Thue, Stephan Schiffel, Ragnar Adolf Árnason,
Ingibergur Sindri Stefnisson, and Birgir Steinarsson

School of Computer Science, Reykjavik University
Menntavegur 1, Reykjavik, 101, Iceland
{davidthue,stephans,ragnara13,ingibergur13,birgir15}@ru.is

Abstract. We present a plan-based story generator that allows authors to ensure continuity over the entities in a story without committing to which entities will fulfill the story's roles. By combining the ideas of authorable continuity and delayed role assignment in a plan-based storytelling context, our solution obtains benefits from both and mitigates some disadvantages. We introduce two notions of soundness for solutions that combine these ideas and then prove the soundness of our approach.

1 Introduction

Continuity is important in storytelling. In addition to being associated with enhanced feelings of meaning, it can aid in improving an audience's understanding of a story [6]. In interactive storytelling, the ability to delay authoring decisions until run-time is also important; it affords more opportunities to gather information about the audience before each decision is made, and it helps to avoid the pitfalls of having pre-made decisions thwarted by unexpected interactions [10, 11]. In the context of plan-based interactive storytelling, the most common way to ensure continuity for the entities in a story (including characters, objects, locations, etc.) has been to name specific entities as part of an Artificial Intelligence (AI) planning problem [1, 7–9] (e.g., Red should start the story at home, and have been distracted and eaten by the Wolf before the story ends). In the context of delayed authoring, however, naming specific entities in advance is both presumptive (e.g., the audience might prefer a different villain) and prone to failures (e.g., the Wolf might be killed before it can distract Red). Writing a less specific planning problem (e.g., stating that Red must be both distracted and eaten, but not by whom) can help avoid these problems [9], but doing so compromises the author's ability to ensure continuity; Red might be distracted by one character and eaten by another. Is there a way to author continuity for story entities without requiring specific ones to be named? The concept of *roles* – abstract versions of story entities that can be assigned to concrete entities just-in-time [5] – hints at a solution: allow authors to define and constrain a set of roles, and have them specify the story's progression constraints using those roles. For example: "Given a Hero and a Villain such that the Villain can eat the

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-48279-8_23

Hero, the Hero must start at home and have been distracted and eaten by the Villain before the story ends)." While this specification permits both authorable continuity and the delayed assignment of roles, it complicates the task of plan-based story generation. Which entities should be available for the planner to use, and how can we ensure that the planner's output will always permit a valid assignment of roles to concrete entities?

In this paper, we explore the challenge of enabling both delayed role assignment and authorable continuity in a plan-based storytelling context. We begin by analyzing the problem, noting its differences from related tasks and demonstrating key properties that its solution must hold to ensure that valid role assignments can be made. We follow with a discussion of related work. We then present an algorithmic solution with the desired properties, prove two aspects of its soundness, and discuss our initial implementation of it using an off-the-shelf planner. We conclude by reviewing our solution's benefits and limitations and offering some suggestions for future work.

2 Problem Formulation

We wish to extend plan-based story generation in a way that supports both authorable continuity and delayed role assignment. We clarify each of these terms in this section and follow with our analysis of the problem.

In plan-based story generation, a story is represented as a plan. A *plan* is a sequence or partial order of *operators*, each of which transforms the state of the story's world to bring it from some initial state to some desired state. Each *state* is represented as a conjunction of logical predicates, each of which describes some property of one or more entities in the world (e.g., Knows(Wolf, Red)). A *planning domain* specifies the set of possible operators and the set of possible predicates, and a *planning problem* specifies the initial and desired states of the world. A *planner* is software that accepts a domain and a planning problem and automatically outputs a plan (if one exists) which, when executed, transforms the problem's initial state to its goal state. *Executing a plan* means applying each of its operators to the world state in the order that the plan specifies.

In general, we say that an Interactive Storytelling (IS) system supports *authorable continuity* (AC) if it allows authors to: (i) describe constraints over a set of story entities, (ii) assert that different constraints should hold for those entities at different times in the story, and (iii) rely on the IS system to ensure that the given constraints are respected at the appropriate times during the story's execution. Plan-based systems usually support authorable continuity inherently, as their initial and goal states can require different predicates to hold for any specific entity at different times in the story. For example, one could author a plan-based story in which Red is hungry at the beginning and not hungry at the end. However, as we explain later on, in plan-based systems that also support delayed role assignment, support for authorable continuity is *not* guaranteed.

We say that an IS system supports *delayed role assignment* (DRA) if it allows authors to: (i) refer to the entities of a story using abstract terms (which are

often called *roles* [5, 12]), (ii) describe constraints governing how each abstract term should be assigned to a concrete story entity in the story's world, and (iii) rely on the IS system to assign each abstract term to a concrete entity at run-time, in a just-in-time way that respects the authored constraints.

Taken together, our desire to support both authorable continuity and delayed role assignment leads to the following requirements. In every solution to our problem, authors must be able to:

1. use abstract roles to refer generally to story entities (DRA i, AC i);
2. assert that different constraints should hold for the abstract roles at different times in the story (DRA ii, AC ii); and
3. rely on an IS system to ensure:
   (a) that every abstract role is assigned just-in-time to a concrete entity that respects the role's constraints (DRA iii), and
   (b) that every given constraint is respected at the appropriate time during the story's execution (AC iii).

2.1 Problem Analysis

To present our analysis of this problem, we consider it from two common perspectives in the field of Artificial Intelligence: planning and constraint satisfaction.

From the perspective of AI planning, we wish to construct and solve a planning problem that refers to a given set of abstract roles as entities in its initial state and goals (to meet requirements 1 & 2). To meet requirement 3a, the operators in the output plan must also use these abstract roles. Furthermore, it is important to guarantee that the planner will never return a plan which, by some properties of the operators therein, makes it impossible to assign one or more of its abstract roles. We will refer to solutions that uphold this guarantee as being *plan-sound*. Without this guarantee, it would be possible to obtain a plan containing some operators that could never be executed.

In addition to abstract roles, it is desirable for the plan to also involve concrete entities from the story's world. There are two reasons why. First, consider the case in which the planner could only use the given abstract roles as entities in its plans. If an author decided to specify a story with only a few abstract roles and a few constraints over them, the planner's operation would be severely restricted; it could only begin the plan with operators whose preconditions were satisfied by the given (few) constraints, and it could only use operators whose parameters could be filled by the given (few) abstract roles. For example, an operator requiring three entities could not be used if only two abstract roles had been given to the planner. If concrete entities could be used by the planner in addition to the abstract roles, these restrictions would be eased. Second, having more entities and relations available to the planner allows a wider variety of plans to be created. Henceforth, we refer to the kind of plan that we desire as *semi-abstract*, since it involves both abstract roles and concrete entities.

Creating semi-abstract plans is non-trivial. While it is common for *least commitment planners* to leave certain variables unbound during the planning process

(see [14] for a review), what we aim for is different. First, a semi-abstract plan's "variables" (i.e., abstract roles) arise as inputs to the planner, rather than being created by the planner during least-commitment planning. Second, we require roles to remain unbound in the final plan, rather than being bound before the plan is returned. Furthermore, while one could modify such a planner to leave some variables unbound in the final plan, more (and potentially extensive) modifications would be required to ensure that plan-soundness was maintained. We present an alternative in Section 4, wherein we obtain semi-abstract plans by automatically augmenting the inputs of an unmodified, off-the-shelf planner.

From the perspective of constraint satisfaction, we wish to solve a constraint satisfaction problem incrementally (e.g., for just one entity at a time) at runtime (to meet req. 3a). Specifically, given a set of roles and a set of constraints between them, we wish to assign each role to some concrete entity in the story's world in a way that respects those constraints (to meet req. 3b). Furthermore, we wish to do so in a just-in-time fashion; by delaying each step of the assignment until the moment that it becomes necessary, the benefits of delayed authoring can be brought within reach[1]. For the solution to be sound, we must guarantee that no (prior) step of the assignment can make it impossible to perform a later step. We will refer to solutions that uphold this guarantee as being *assignment-sound*. As we noted earlier, the interdependence of assignment steps precludes simply tweaking a least-commitment planner to leave some variables unbound.

At what point does each step of an assignment become necessary? Given a semi-abstract plan, its operators can be executed (perhaps in parallel) according to their partial order in the plan. Operators that involve only concrete entities can be executed immediately, but operators that involve abstract roles must have each of their roles assigned to a concrete entity before all of their preconditions can be checked. Therefore, for a given abstract role r and semi-abstract plan p, we say that r's assignment *becomes necessary* the first time that the execution of p needs to check the preconditions of an operator that involves r.

While different types of *narrative mediation* [8] could conceivably be used to work around impossible assignments (e.g., by finding a different plan), we instead seek to discover whether a solution exists that is both plan-sound and assignment-sound, and if so, how it might be made.

3 Related Work

Fairclough's OPIATE [2] used *case-based planning* to automatically adapt a set of hand-authored plans that involved abstract roles, and it assigned concrete entities to roles just-in-time as the plan was executed. Authors could create plans for the case base that constrained the same abstract characters at different times in the story. OPIATE technically meets the solution requirements that we stated in Section 2, but it assumed that every planning operator would be tightly coupled to a particular role in a story-universal set (e.g., the Victory operator

[1] A discussion of how these might be seized is beyond the scope of this paper.

could only ever be performed by the Hero). It could thus only generate stories that had roles from this set; generating stories with different roles would require both a new set of operators and a new case base. Our solution can generate stories with different roles using a single set of operators (and no case base).

As we noted in Section 2, when delayed role assignment is *not* part of a given IS system, plan-based story generation inherently supports authorable continuity; the initial and goal states allow constraints to be specified over any of the entities that could be involved in the plan, and they each describe different time points in the story. While previous attempts to add delayed role assignment to such systems have supported authorable continuity for concrete entities, there has been no similar support for abstract roles. For example, Riedl & Stern's Automated Story Director [9] could change which characters would achieve certain important world states (e.g., different agents could cause unrest in a marketplace), but there was no way for an author to create constraints for an abstract role (e.g., UnrestCauser) that spanned different points of time in the story's progression. Similarly, a system by Porteous and Cavazza [7] used *state constraints* to allow authors to constrain story entities at arbitrary points in the story's progression (not just the initial and goal states), but these entities could only be concrete. Barber's GADIN [1] allowed authors to describe and constrain abstract roles in the context of *dilemmas* (states requiring a character to choose between two costly actions), and it assigned concrete entities to each dilemma's roles at runtime. However, GADIN only used its planner *after* all assignments were complete; it thus did not support authorable continuity for its abstract roles.

Thue et al.'s PaSSAGE [12] allowed authors to constrain abstract roles that it then assigned just-in-time during the story, and constraints could be specified for any role across different times of the story (e.g., a Rival met early in the story could appear again later on). However, while PaSSAGE thus supported both delayed role assignment and authorable continuity, it did not use a planner to generate its stories; it relied instead on a pre-authored tree of possible events.

4 Proposed Approach

We now present a novel extension to plan-based story generation that supports both authorable continuity and delayed role assignment. To structure our presentation, we consider each of our stated solution requirements in turn (Section 2).

To allow authors to both refer to story entities generally using abstract roles (req. 1) and assert that different constraints should hold for those roles at different times in the story (req. 2), we adopt Thue & Halldórsson's notion of an *outline* [13]. An outline is a construct that allows authors to specify both a set of abstract story entities (i.e., roles) and constraints over them that should hold at different points of time during a story (e.g., Roles: Hero, Villain; Initial State: CanEat(Villain, Hero); Goal State: Distracted(Villain, Hero)$\wedge$ Ate(Villain, Hero)). Each outline can be used to generate multiple varied stories.

Meeting requirements 3a (enabling just-in-time role assignment) and 3b (ensuring correct plan execution) is more complicated. At a high level, we: (i) con-

struct a semi-abstract planning problem from the outline and the story world's current state, (ii) find a relaxed solution for assigning the outline's roles and add this to the planning problem, (iii) solve the plan, letting a planner explore potential role assignments alongside choosing operators for the plan, (iv) post-process the plan to recover any new role constraints and "forget" the planner's assignments, and (v) execute the plan, assigning roles to concrete entities when doing so becomes necessary. We describe these steps in the following paragraphs.

4.1 Detailed Approach

We first need a way to generate semi-abstract plans (Section 2.1). While any outline's constraints can be used to define a completely abstract planning problem, concrete entities and relations are needed to make a problem that can yield semi-abstract plans. For a given outline, we meet this need as follows. We begin by defining the initial state of a planning problem as the union of three components: (i) the outline's initial state, (ii) the current state of the story world, and (iii) a set of extra CanBe relations. CanBe(r, c) is a predicate that relates an abstract role r to a concrete entity c; it is true only when assigning r to c is part of a valid complete assignment for the outline's roles, given their constraints (i.e., when r "can (safely) be" c). We use a constraint solver to compute the CanBe relations just prior to building the planning problem. In general, if r is assigned to c in *any* solution, CanBe(r, c) will be true (even if r cannot be c in *some* solutions). Doing so is a relaxation of the problem that temporarily ignores any potential interdependence between roles, which allows the planner to start exploring potential assignments as part of its search. The interdependence of roles is accounted for during the planning process, as we describe later on.

To allow a planner to solve our semi-abstract problem while maintaining *plan-soundness* (Section 2), it is necessary to have the planner reason about potential role assignments while it searches for a sequence of operators that meets the plan's goals. We accomplish this task by automatically augmenting the domain of our planning problem. To begin, an author creates a set of predicates for describing the state of the world (e.g., Knows(c_1, c_2)) and a set of operators that define how the world can change. The author also uses this set of predicates to specify every constraint in the outline. The operators can be defined using PDDL syntax [3], as sketched in Listing 1 (our entities are PDDL "objects").

Defining an operator consists of specifying its parameters as entity variables and its preconditions and effects as logical statements. Taken together, the authored predicates and operators make up the initial planning domain that we then augment. Two augmentation steps are necessary.

Listing 1: A simple operator: c_1 distracts c_2 using a_1.

```
1 operator: distract(c1, c2, a1)
2     preconditions: c1 ≠ c2 ∧ Knows(c1, c2) ∧ Knows(c1, a1)
3     effects: Distracted(c1, c2) ∧ Knows(c2, a1)
```

4.1.1 A Mapping Operator. First, we generate a new operator: map(r, c). This operator allows the planner to experiment with assigning roles to concrete entities (e.g., Hero to Red) as it searches for a valid plan that includes the authored operators. Each of these operators records that the assignment is (temporarily) in force using the predicate: Mapped(r, c). For example, if successful, the operator map(Hero, Red) would yield the effect Mapped(Hero, Red).

To ensure both plan-soundness and assignment-soundness, we must only allow the planner to make temporary assignments which, when considered in the context of its previous assignments, correctly satisfy the outline's constraints. For example, if the outline constrained two roles with CanEat(Villain, Hero) and the planner had already "mapped" Hero to Red, then the possible assignments for Villain would need to be restricted to entities that can eat Red. Enforcing this sort of restriction requires extra logic in the preconditions of the map operator, as shown in Listing 2. Line 3 ensures that the planner can only (temporarily) assign each role once, and the CanBe ensures that the planner's first assignment will be part of a valid solution to the outline's constraints.

Listing 2: The map operator with sample tests of outline constraints.

```
1 operator: map(r, c)
2     preconditions:
3     ¬Bound(r) ∧ Abstract(r) ∧ ¬Abstract(c) ∧ CanBe(r, c) ∧
4     ∀r′, c′ : Abstract(r′) ∧ (r ≠ r′) ∧ Mapped(r′, c′) ⇒
5         [ (CanEat(r, r′) ⇒ CanEat(c, c′)) ∧ ··· ∧ (Knows(r, r′) ⇒
          Knows(c, c′)) ]
6     effects: Mapped(r, c) ∧ Bound(r)
```

For every role/concrete entity pair that has already been temporarily mapped (r' and c' on line 4), we must verify that the new candidates for mapping (r and c) will respect every outline constraint that exists between r and r'. In other words, for every outline constraint that holds between r and r', we must ensure that it also holds between c and c'. Line 5 shows examples of how testing these constraints might look, but in general these tests are automatically generated to suit the given outline. The map operator is thus unique to each outline. We explain our need for the Bound(r) predicate in Section 4.1.3.

4.1.2 Effect Synchronizers. Our second augmentation to the planning domain involves modifying every authored operator. Once the planner has mapped a role to a concrete entity, it is important for plan-soundness that any effect that occurs to one of the two also occurs to the other. For example, if Hero was mapped to Red, Villain was mapped to Wolf, and the operator distract(Villain, Hero, Flowers) occurred, then the effects that should come true would be more than just Distracted(Villain, Hero) ∧ Knows(Hero, Flowers). Distracted(Wolf, Hero) and Distracted(Villain, Red) should also be true, along with all of the

other variations that can be generated by exchanging roles with concrete entities in each of the effects' parameters. To keep each operator's effects synchronized across roles and concrete entities once they have been mapped, we automatically add additional effects to every authored operator, as demonstrated in Listing 3.

Listing 3: An augmented version of the distract operator from Listing 1.

1 **operator:** distract(c_1, c_2, a_1)
2 **preconditions:**
Bound(c_1)∧Bound(c_2)∧Bound(a_1)∧$c_1 \neq c_2$∧Knows(c_1, c_2)∧Knows(c_1, a_1)
3 **effects:** Distracted(c_1, c_2) ∧ Knows(c_2, a_1)
4 [$\forall c_1', c_2'$: (Mapped(c_1', c_1) ∨ Mapped(c_1, c_1') ∨ $c_1 = c_1'$) ∧
5 (Mapped(c_2', c_2) ∨ Mapped(c_2, c_2') ∨ $c_2 = c_2'$)
6 ⇒ Distracted(c_1', c_2')]
7 [$\forall c_2', a_1'$: (Mapped(c_2', c_2) ∨ Mapped(c_2, c_2') ∨ $c_2 = c_2'$) ∧
8 (Mapped(a_1', a_1) ∨ Mapped(a_1, a_1') ∨ $a_1 = a_1'$)
9 ⇒ Knows(c_2', a_1')]

For every possible combination of the operator's parameters that appears in its effects ((c_1, c_2) and (c_2, a_1) on line 3), we retrieve all of the roles and concrete entities that have been mapped to or from the terms that the parameters represent (lines 4-5 and 7-8). We then apply each of the effects that use that combination of parameters (one for each combination, in Listing 3) to all possible variations of the parameters and their mapped counterparts (lines 6 and 9).

4.1.3 Tracking Temporary Assignments. We use the predicate Bound throughout our solution to track whether or not any role r still needs to be mapped (if so, Bound(r) will be false). This information is useful in several ways. First, we ensure that the map operator will be used by the planner by adding preconditions in every operator to assert that each of its parameters must be Bound (line 2 in Listing 3). Next, we also ensure that the planner will make use of every role r in the outline by (i) adding Bound(r) to the goal state of the semi-abstract planning problem for every such r (required for assignment-soundness) and (ii) adding Bound(r) as an effect of map(r, c) (line 6 in Listing 2). Finally, to allow the planner to instantiate operators using concrete entities, we add Bound(c) to the initial state of the problem for every concrete entity c.

To continue our example, suppose that the world has the concrete entities {Red, Wolf, Troll, Ogre, Flowers} and an initial state such that the Wolf, Troll, and Ogre all know and can eat Red, and that the Wolf and the Troll know about the Flowers but the Ogre does not. The following is a partial, semi-abstract plan that our approach might generate, with some parts of the story replaced with "..." for brevity: "map(Hero, Red), ..., map(Villain,Wolf), distract(Wolf, Red, Flowers), ..., eat(Wolf, Red), ...". The plan is semi-abstract because it contains both roles (e.g., Hero) and concrete entities (e.g., Flowers).

4.1.4 Plan Post-processing. Once a semi-abstract plan has been generated, three post-processing steps are needed. First, every instance of the map operator is removed. Second, since the planner could have instantiated operators that involve mapped concrete entities in their parameters, we must visit each operator in the plan and replace each occurrence of such an entity with the role that it was mapped from (we currently assume that there was only one). We do so to "forget" about the planner's temporary assignments in advance of performing delayed role assignment later on. Applying these two steps to our example plan would yield: "..., distract(Villain, Hero, Flowers), ..., eat(Villain, Hero), ...". Third, we must check the preconditions of every operator that has a role in its parameters. If any precondition that involves roles is satisfied by the world's initial state (which could happen via the role's mapped counterpart), then we must add that precondition as a new constraint on each of the involved roles. In our example plan, the distract operator has the precondition Knows(Villain, Flowers) (recall Listing 3), which, given the planner's temporary mapping of Villain to Wolf, was satisfied by the initial state. As a result, the third post-processing step would add the constraint Knows(Villain, Flowers) to the outline. If this was not done, the role of the Villain could be assigned at runtime to the Ogre (which satisfies CanEat(Villain, Hero) in the outline), and the plan would fail because the Ogre does not know about the Flowers (failing requirement 3b).

4.1.5 Execution & Delayed Assignment. Given a post-processed, semi-abstract plan, execution can begin. Similarly to Thue & Halldórsson's approach to execution [13], we keep track of a frontier of operators in the plan; any operator whose preconditions are both satisfied by the current world state and independent from any prior, unexecuted operators are executed in the story's world. However, before a given operator's preconditions can be checked, any unassigned roles that the operator involves must be assigned. To do so, we create a constraint satisfaction problem with the outline's roles and constraints (the latter of which may have been increased during post-processing) as one input and the current story world as the other. To ensure assignment-soundness, we must solve the CSP for *all* of the outline's roles every time that an individual role's assignment becomes necessary. Once one or more solutions are obtained, assignments for the roles in question can be selected using any feasible method. In our example plan, the role of the Villain would be assigned immediately before checking the preconditions of the distract operator. Assuming that the Ogre had not come to know about the Flowers since the story started, the available assignments would be: $\{\text{Wolf}, \text{Troll}\}$.

5 Soundness

We are interested in our approach being sound in the two ways that we presented in Section 2. First, the plans that are generated must be plan-sound, that is, it must be possible to assign all abstract roles that appear in any operator in the plan. Secondly, we require the incremental solution of the constraint satisfaction

problem to be assignment-sound, that is, no step of the assignment can cause the remaining steps to become impossible to execute. We argue here that our approach ensures both forms of soundness.

Proposition 1 (Plan-Soundness). *A legal plan composed of operators augmented as described in Section 4 is plan-sound; that is, there exists a mapping $M : R \to C$ from abstract roles $r \in R$ to concrete entities $c \in C$, such that for every abstract role r appearing in any step of the plan or the goal condition, exchanging every instance of r with $M(r)$ leads to a valid plan involving only concrete entities that achieves the (concrete) goal.*

Proof. (Sketch) The preconditions of all augmented operators require Bound(r) for every abstract role r appearing as a parameter of the operator. The only operator resulting in Bound(r) is map(r, c), which requires that c be a concrete entity that is similar to r in the sense that whenever r is in a predicate with some other abstract role r' and r' has been mapped to c', c must be in the same predicate with c'. Thus c can safely be replaced for r at the time when map(r, c) would be executed, because it fulfills all the same conditions as r, provided that all other roles that r is in predicates with will be replaced in the same way.

Furthermore, by the construction of the augmented operators (Section 4.1.2), all effects that apply to any entity (abstract or concrete) are also applied to all entities that are mapped to or from it. Thus, any concrete entity c is similar to the abstract role r that it is mapped from at all times during the plan following the map(r, c) operator. Thus, c can safely be replaced for r at any time after map(r, c) in the plan without making the plan invalid.

Proposition 2 (Assignment-Soundness). *Let R be the set of all abstract roles occurring in a plan p and $R_p \subset R$. Any partial assignment $M_p : R_p \to C$ of abstract roles to concrete entities occurring during plan execution (Section 4.1.5) can be extended to a full assignment $M : R \to C$ with $M_p(r) = c \Rightarrow M(r) = c$ for all r, c, such that all the constraints are fulfilled.*

Proof. (Sketch) Proof by induction over the size of M_p. Base case: $M_p = \emptyset$ (i.e., no assignment has been made). As given in the final state of the computed plan p, Mapped(r, c) is an extension of M_p to a full assignment that fulfils all constraints, because our approach requires the planner to use and find a mapping for every role $r \in R$ (Section 4.1.3).

For the inductive step, assume that M_p is the partial assignment occurring during plan execution just before an operator o that contains an unassigned role r and that there is an extension M of M_p to a full assignment fulfilling all constraints. Our online role-assignment method (Section 4.1.5) will pick one of the full extensions as computed by the constraint solver and extend M_p to $M'_p = M_p \cup \{r \mapsto M(r)\}$. Thus, M is also an extension of M'_p to a full assignment.

6 Discussion

To the best of our knowledge, this work represents the first attempt since OPIATE to combine the three ideas of plan-based story generation, authorable con-

tinuity, and delayed role assignment. Unlike OPIATE, our solution supports the generation of stories with arbitrarily different roles using a single, simply authored domain. Compared to other plan-based IS systems, our solution offers a way to author continuity for story entities without having to name specific ones, which supports increased generative variety. Because all of our augmentations to the planning domain are generated automatically with each new problem, the authoring burden of plan-based storytelling remains nearly unchanged; the only extra work comes from defining the (typically few) extra entities and constraints that outlines require. We have conducted preliminary tests with an unmodified, off-the-shelf planner (Fast Downward [4]), and length 10 plans with roughly 15 concrete entities and 10 possible operators take a few seconds to compute on a single core machine. As might be expected from our effect synchronizers, increasing the number of entities can significantly increase computation time.

In addition to its sensitivity to total entity count, our method has other limitations. Due to the closed world assumption that Fast Downward makes, our mapping operator can only correctly check positive predicates as constraints in the outlines. Due to the complexity of keeping effects synchronized across abstract roles and concrete entities, our code that generates the effect synchronizers can currently only manage binary constraints. However, increasing the arity of constraints beyond two will likely have negative effects on performance, as doing so will require more universally quantified terms. A major strength of the augmentations is that they are almost (aside from the two previous restrictions) completely transparent to the author; we currently generate full PDDL domain and problem files from a more convenient internal representation for outlines, operators, and entity sets. Although we presented our augmentations without discussing any subtypes of entities, in practice we are able to generate them with full support for differently typed entities. We only omitted these types to visually simplify the listings, which became overcomplicated when they were included.

While we have proven that our solution is sound in the sense that its plans will not fail for self-sourced reasons, we cannot generally guarantee that its plans *cannot fail*. In an interactive context, it seems likely that the techniques of narrative mediation will be needed to assist when plans go awry. If the domains of such repairs can be kept within the rough size that we have tested, it might be feasible to use our solution to find new plans in real time.

7 Conclusions & Future Work

We have made three contributions in this work. First, we performed a detailed analysis of the challenge of combining authorable continuity with delayed role assignment in a plan-based storytelling context, and we concretely specified a set of requirements for its solutions. Second, we presented a novel approach to solving this problem that meets the stated requirements while simultaneously providing new advantages over existing work. Finally, we introduced two new notions of soundness for our problem (plan-soundness and assignment-soundness) and sketched proofs to show that our approach is sound in those respects.

Given the variety of concerns that must be addressed by solutions to this challenge, many opportunities exist for further research. Beyond working to address the known limitations of our solution, further experimentation is needed to discover more of its limitations that remain unknown (e.g., how many entities can be handled within a given amount of time?). Further technical extensions are also possible, such as integrating state constraints with semi-abstract plans to make the authorable continuity more fine-grained. Changing logical representations might also be worthwhile, as the constraints of PDDL planning may ultimately be too limiting for the kinds of authoring that we wish to support.

References

1. Barber, H.: Generator of Adaptive Dilemma-based Interactive Narratives. Ph.D. thesis, Department of Computer Science, The University of York (2008)
2. Fairclough, C.: Story Games and the OPIATE System. Ph.D. thesis, University of Dublin - Trinity College (2004)
3. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Tech. rep., Dipartimento di Elettronica per l'Automazione, Uni. degli Studi di Brescia (2005)
4. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research 26, 191–246 (2011)
5. Mac Namee, B., Dobbyn, S., Cunningham, P., O'Sullivan, C.A.: Men behaving appropriately: Integrating the role passing technique into the ALOHA system. In: Animating Expressive Characters for Social Interactions. pp. 59–62. AISB (2002)
6. Magliano, J.P., Zwaan, R.A., Graesser, A.C.: The role of situational continuity in narrative understanding. The construction of mental representations during reading pp. 219–245 (1999)
7. Porteous, J., Cavazza, M., Charles, F.: Applying planning to interactive storytelling: Narrative control using state constraints. ACM Transactions on Intelligent Systems and Technology 1(2), 111–130 (2010)
8. Riedl, M.O., Saretto, C.J., Young, R.M.: Managing interaction between users and agents in a multi-agent storytelling environment. In: 2nd international joint conference on Autonomous agents and multiagent systems. pp. 741–748. ACM (2003)
9. Riedl, M.O., Stern, A.: Believable agents and intelligent story adaptation for interactive storytelling. In: Göbel, S., Malkewitz, R., Iurgel, I. (eds.) TIDSE 2006. LNCS, vol. 4326, pp. 1–12. Springer, Heidelberg (2006)
10. Swartjes, I., Kruizinga, E., Theune, M.: Let's pretend I had a sword: Late commitment in emergent narrative. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 264–267. Springer Berlin / Heidelberg (2008)
11. Thue, D., Bulitko, V., Spetch, M.: Making stories player-specific: Delayed authoring in interactive storytelling. In: Spierling, U., Szilas, N. (eds.) ICIDS 2008. LNCS, vol. 5334, pp. 230–241. Springer Berlin / Heidelberg (2008)
12. Thue, D., Bulitko, V., Spetch, M., Webb, M.: Socially consistent characters in player-specific stories. In: The Sixth Artificial Intelligence and Interactive Digital Entertainment Conference. pp. 198–203. AAAI Press (2010)
13. Thue, D., Halldórsson, K.: Opportunities for integration in interactive storytelling. In: Schoenau-Fog, H., Bruni, E.L., Louchart, S., Baceviciute, S. (eds.) ICIDS 2015. LNCS, vol. 9445, pp. 374–377. Springer Berlin / Heidelberg (2015)
14. Weld, D.S.: An introduction to least commitment planning. AI magazine 15(4), 27 (1994)