Timing Interactive Narratives

Thomas Cabioch
ENIB – Lab-STICC
Brest, France
thomas.cabioch@enib.fr

Ronan Champagnat
L3i
La Rochelle université
La Rochelle, France
ronan.champagnat@univ-lr.fr

Anne-Gwenn Bosser
ENIB – Lab-STICC
Brest, France
bosser@enib.fr

Jean-Noël Chiganne
Incarna
Paris, France
jn@incarna.co

Martin Dieguez
ENIB – Lab-STICC
Brest, France
martin.dieguez@enib.fr

***Abstract*—Research in Computational Narratives has evidenced the need to provide formal models of narratives integrating action representation together with temporal and causal constraints. Adopting an adequate formalization for narrative actions is critical to the development of generative or interactive systems capable of telling stories whilst ensuring narrative coherence, or dynamic adaptation to user interaction. It may also allow to verify properties of narratives at design time. In this paper, we discuss the issues of interactive story design, verification, and piloting for a specific genre of industrial application, in the field of interactive entertainment: in the games we consider, teams of participants in a Virtual Reality application are guided in real time through a narrative experience by a human storyteller. Like in an escape game, the interactive experience is timed: it should be long enough to provide satisfaction to the players, but come to a conclusion before the game session is over in order to provide closure and a sense of achievement to them. We describe how we integrate narrative time in the story design and use it to the verification of temporal properties of scenarios, building on previous work using Linear Logic and Petri Nets.**

I. Introduction

Interactive Storytelling (IS) research focuses on the challenges posed by a new form of media, which would allow the audience to influence the unfolding of a story while experiencing it. Foretold by Murray [1], the first fully integrated Interactive Storytelling System in Virtual Reality was reported by Cavazza et al. [2]. With the popularization of Virtual Reality entertainment, and as research in IS matures, ideas make their way into the entertainment computing industry, meanwhile raising new practical issues. In this paper, we describe how we apply and adapt results from research in IS to the problem of authoring time-constrained interactive narratives for providing virtual reality narrative experiences for teams of players. More specifically, we describe how we have extended a prototype authoring tool and authoring method to support the validation of time-constrained narratives. We use Linear Logic [3], [4], with time annotations, to model narrative actions, and verify temporal properties using a translation into Petri nets.

II. A Structural Approach to Narrative Verification

Interactive Storytelling research has often been inspired by Classical (Structural) Narratology [5]. Building on the work of the Russian structuralists, theorists like Barthes [6] have put an emphasis on the underlying structures of narratives. The conceptual distinction between the raw material of the narrative (characters, objects, narrative actions, ...) and the order and manner in which it is conveyed to the audience has helped frame a number of approaches in Computational Narratives. In this paper, we consider the former, the *story* (the latter being commonly designated as *discourse* in IS), following terminology borrowed from narratologists like Genette [7] or Chatman [8]. Another important take away from narrative theories for research in IS has been the centrality of narrative actions (such as in the *schema actanciel* [9]). This has made action description a fundamental element of formal models of narratives and the work presented here makes no exception to this.

In this respect, specifying an interactive narrative requires defining a story (characters and objects, states of the narrative world, and how narrative actions impact this world), and underlying each unfolded narrative is an implicit causal structure of narrative actions (often referred to as *plot*). AI planning has been the technique of choice, for many researchers in IS, since Young [10] proposed to specify narrative actions as planning operators and to generate causally sound narratives by tweaking planning engines with narrative goals. To this day, many successful IS systems rely on planning techniques, integrating the effects of user interaction through re-planning. Planners are typically built to counter combinatorial explosion and as such, endeavour to reduce the search space: the level of generativity of IS systems has become an issue in itself for assessing the performances of systems [11].

A dual approach, less concerned with on-the-fly dynamic generation, is to treat story specification at the logical level and use formal techniques to establish properties of the possible narrative discourses: Bosser et al. [12] have shown how Intuitionistic Linear Logic can be used to specify a story and adequately account for narrative causality phenomena. It relies on the use of Linear Logic implication to describe narrative actions as consuming and producing resources (and has even led to the design of a linear logic based programming language particularly well suited to narrative description for generation purposes [13]). For instance, Bosser et al. [14] use the Coq proof assistant to establish high level properties of possible narratives specified in Linear Logic. This can be compared with the work by Colle et al. [15] for computer games, where a Linear Logic based specification of game-play

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

was automatically transformed into Petri nets for verification purposes. These previous work were based on formalisms and tools which may not be author-friendly, so for this project, we chose to build on the work of Dang [16], which provides the specification of a higher-level authoring tool based on linear logic. Dang et al. [16] describe a tool to model and analyze a narrative based on Linear Logic description and a translation into Petri net. This authoring tool front-end is based on state presentation and then uses model transformation to derive a LL model and PN to analyze. The tool provides an analysis based on a coverability graph unfolding. However it faces limitations, such as infinite branches: we describe in section IV-D how we addressed this issue. Our contribution here is thus twofold:

- we extend the expressiveness of this tool to account for the verification of temporal properties using ideas from the work of Champagnat et al. [17];
- after a translation into Petri Nets following Pradin's methodology [18] which allows verification in the spirit of the work of Colle et al. [15], we use the algorithm developped by Reynier et al. [19] to verify the story specification with regards to time constraints on the unfolding.

III. Escape games flavoured with VR Narrative Experiences

Our work takes place in an industrial context: the development of immersive narrative experiences designed for small teams of players: participants, each equipped with a Virtual Reality Head-Mounted Display, enter a shared virtual world where they experience and act in a narrative. Players are protagonists and part of the narrative so the user-interaction is diegetic (following the terminology used by Cavazza et al. [20]). Contrarily to role playing games, participants are experiencing the application as themselves and do not play a role, and whilst interaction and communication is possible within the virtual environment, the players share the same point of view of the narrative and witness and trigger narrative actions as one entity: this provides them with a shared view of the narrative experience in which they can communicate.

By contrast, a human operator (which we will call here the narrator) driving the narrative experience is non-diegetic and the players should be oblivious to her influence on the narrative unfolding. The role of the human operator is to ensure that the players enjoy the experience, which involves dynamic adjustments to ensure that the level of challenge and implication provided corresponds to the players' skills. This is indeed, an essential component of a well-balanced user centered game-design (see for instance [21]). An important part of this task is to ensure that the players will be able to understand the narrative in the time imparted, but also to make sure the narrative is not unfolded too quickly for the player to not be disappointed by the duration of the session.

This requires the design of dedicated *piloting* tools, equipping the narrator with ways of monitoring the plot that has unfolded so far, compare possible plot trajectories and narrative actions left to be experienced to the remaining time, and means to interact with the system in order to nudge the players on the path to narrative closure by tying "loose ends" of the narrative. Authoring such narratives also require dedicated tools and methods including for the validation of duration properties: we propose such a solution here, whilst accounting for the requirement of the future piloting tools (such as keeping track of the causal relations between narrative actions through an explicit representation of the plot, and ways to compute the duration of the remaining possible narratives), which we intend to develop in the future.

Because a team of players behaves as a unit from a narrative perspective, the running example we are using in this paper to illustrate our approach considers one protagonist, and its available actions correspond to a modified version of the Interactive Fiction *Silver Hair and the three bears* [1] (an ancient version of the tale featuring Goldilocks). Silver Hair is a witch living in the woods close to the house of three bears. One morning, while the bears prepare for a walk, Silver Hair came to visit but the three bears didn't invite her into their house. To take revenge, she decides to go and vandalize their cottage during their walk.

IV. Authoring and Verifying Time Constrained Narratives

We extended the tool described by Dang [16], which uses Linear Logic for story description, with the possibility to specify an aimed duration for the generated narratives, as well as minimum/maximum time durations for narrative actions [17].

These modifications led us to the UML model shown in figure 1.

A *Scenario* is composed of a set of States, EventActions, Choices and Outcomes :

- **States** are used to define the story world, they represent what can be True in the world at a given time, they all have a name and a short text describing their use, they can be divided in three sub-parts :
  - **GameStates** represents internal states that are changed internally in the system, they can be set to be a part of the initial world.
  - **GameInputs** represents states that truth values are given by the game engine when a certain action is made.
  - **GMInputs** represents states that becomes true when the game master wants them to be true, allowing him to control the scenario in real time.
- **EventActions** are all the actions that can happen during the scenario, they have a name and a description. However the duration of some actions depend on players and may vary from a team of players to another so we use estimated intervals, which can be set during the authoring process, and later refined using expertise gained

[1] https://alyssalandry.com/files/295271/silver-hair-and-the-three-bears-june-2017.html

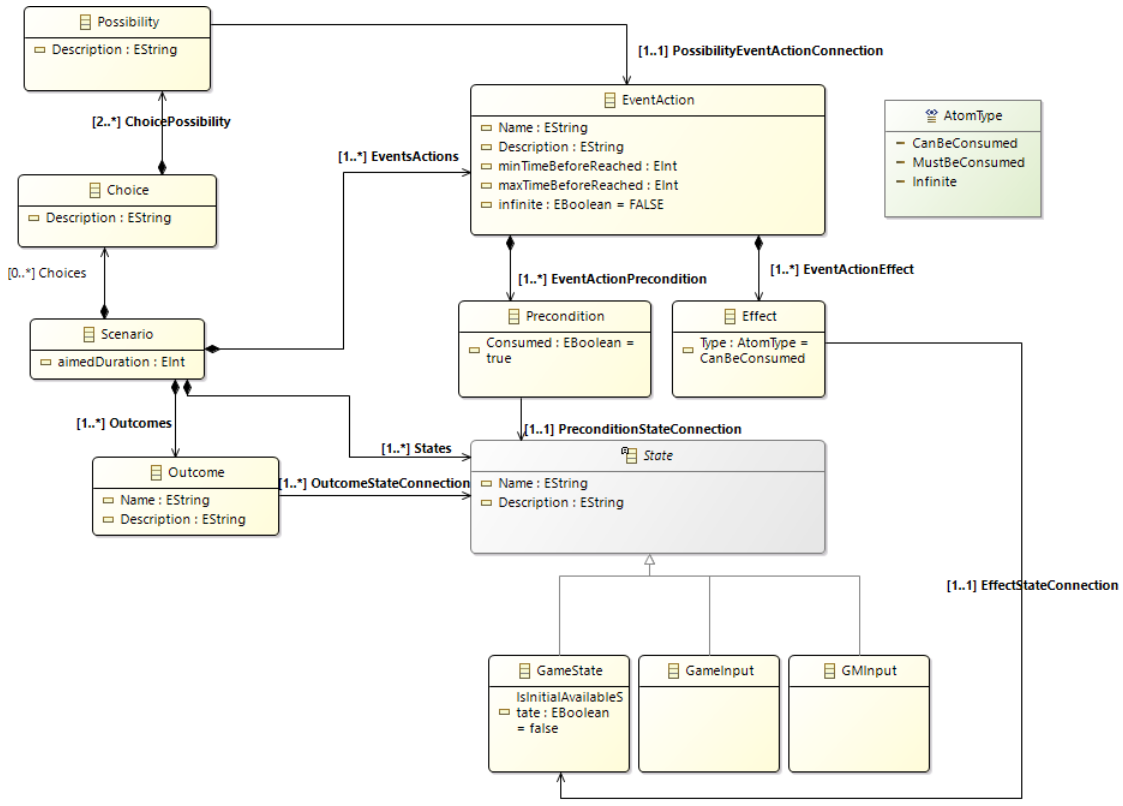

Fig. 1: Story model based on the work of Dang et al. [22], taking in account time duration

from running play sessions on various player profiles. This allows us to estimate scenarios durations during the analysis process. In addition, most actions are made to be used one time, once they are used, they can't be done again, however, for some specific kind of actions such as displacement actions, it is necessary to allow them to be used multiple times, thus, the model allows us to set an action as an infinite one, meaning that it can be done multiple times. An action is composed of :

  - **Preconditions :** A set of *States* that are necessary for the action to be possible, when the action is made, all the states associated to the preconditions are consumed.
  - **Effects :** A set of *GameStates* that are created when the action has been undertaken.

- **Choices** represent choices between different actions, when one of the action is undertook, all the actions linked to the *Possibility* that are parts of the choice are disabled for the rest of the story.
- **Outcomes** represent the desired outcomes that we want for our *Scenario*, they are linked to a set of *GameStates* that we want to be true at the end of the story.

We give here the equivalence between the model and Linear Logic, which we hope will be sufficient for the reader to grasp the meaning of the connectors for understanding the modelling in the next section. It may be useful to keep in mind that in Linear Logic, all formulas are treated as resources, and not infinitely available as permanent statement as in Classical Logic. For instance, the linear implication formula $A \multimap B$, when used, means that A has been consumed (is not available anymore) and that B is now available: you get B with the price of A. This contrast with the meaning of $A \rightarrow B$ in classical logic which means that whenever A, then B. This makes this logic suitable to represent dynamical systems [4]. Details and examples of similar usage for narrative encoding can be found in [22]) :

- Atoms correspond to states and inputs.
- $\otimes$ formulas correspond to a set of states in preconditions/effects for an action
- $\oplus$ formulas correspond to choices between actions, or between outcome possibilities.
- $A \multimap B$ correspond to an action, A is the precondition required, and on its right, there is the set of effect.
- ! before an action means that it is infinitely available.

The authoring process then consists in defining a knowledge base for our scenario using the editor we created derived from this model (see figure 2). We refer the reader to [12] for the underlying details on using LL for narrative encoding.

A. Silver Hair and the Three Bears Story Specification

We start by defining the different possible states in the story. Silver Hair can move in her home or in the Bear's house which contains several places with different available narrative actions, so we have a state for each. For example, the state SHKitchen models the fact that Silver Hair is in the bear's kitchen.

We also keep track of the bears: they can be preparing for a walk (BPrepare), walking (BWalk) or back to their house (BReturned). Once all states have been created, we use them to describe the possible *Narrative Actions*. A narrative action is encoded using the linear implication and describes how resources or states are *consumed* when new ones are *produced* in a narrative environment. Most of the actions are supposed to happen only once, and this is the default encoding provided by linear logic which considers formulae as resources. However, a more classical specification is sometimes desirable for encoding actions which are available a number of times, such as moving between places, and this is possible to define this in a controlled manner using the ! connector. We can define exclusive alternatives too using the choice connector: when the bears are back and that the player (Silver Hair) is in their bedroom, she can hide under the blanket or in the closet. Here is a formula specifying the narrative action *Hiding under a Blanket*:

$$SH_{Bedroom} \otimes B_{Returned} \otimes C_{Blanket} \multimap SH_{Blanket} \otimes B_{Returned}$$

The last step is to define *Goal* states specifying the desired outcomes of the narratives. We define two possible outcomes: (1) Silver Hair is in the bedroom hidden under the blanket and the bears eat her ; (2) Silver Hair is in the bedroom hidden in the closet and the bears catch her.

$$SH_{HideCloset} \otimes B_{Returned} \oplus SH_{HideBlanket} \otimes B_{Returned}$$

We can now define a scenario, a specification of all possible plots, using a LL sequent:

$$InitialStates, NarrativeActions \vdash Goal^{\oplus}$$

where $InitialStates$ corresponds to a set of atoms giving the initial state, and the other multisets respectively specify all possible narrative actions and the goal state description. A given proof of this sequent, by giving a partial order for the decomposition of formulae encoding narrative actions, corresponds to a plot. Proving a sequent consists in rewriting the sequent by replacing connectors by their introduction rules until it remains only the meta-connector ",". A sequent is valid if all the finishing path of the proof tree are of identity rules. One proof corresponds to an unfolding of the story. Writing all the proofs gives the scenarios.

B. Adding Time Annotations

In order to reason about time in Linear Logic, we added annotations to atoms depending on their production date [18]. In our case we consider duration that may vary according to players' actions. As a consequence we used intervals that are set during the authoring process.

Time annotations are calculated during the construction of the proof graph, the atoms present in the initial marking are considered to be produced at the date 0. We first simplify times connector to the left part of the sequent. As a consequence the production date of the tokens may differ. A set of tokens with different production date may enable a transition or a set of transition. When we go through a transition (linear implication), we look at the time annotations from all the present states, and we add the maximum values of the minimum and maximum creation time (associated with the linear implication) from the state to the time estimations associated to the actions. It is then possible to use a (max, +) algebra to determine the duration of possible unfoldings of the story (see example end of section IV-D .

C. Deriving Petri Nets

Pradin et al. [18] also propose a method for deriving Petri net from linear logic sequents:

- Each atom corresponds to a place. An atom true means a marked place.
- A linear implication formula corresponds to a timed transition in the net.
- The left side of the sequent is made up of implication formulae and propositional formulae. It represents the set of transitions to be fired and the initial marking.
- The propositional formulae on the right side of the sequent corresponds to the final marking.

Figure 3 represents the Petri net we obtain from the specification of *Silver Hair*.

D. Verification of Temporal Properties using Petri nets

The main properties that we want to verify are the validity of the possible narratives. In order for a scenario to be valid, all its instances must lead to one of the desired end, and the aimed duration must be within the bounds defined at authoring time. A naive approach exploring all the paths is impossible here because loops can be created by those narrative actions which have been defined as always available. A way to counter this problem is by avoiding states that have already been visited, this allows to prune the infinite branches. This is not sufficient to help with the combinatorial explosion, due to the fact that a lot of paths correspond to the simple reordering of narrative actions: in *Silver Hair*, the player can drink the milk or eat the porridge of the bears in any order, creating two paths that should ideally be seen as being the same story as there is no causal relationship between these two actions.

To solve this problem, we applied to our Petri Net the Karp and Miller algorithm with pruning described by Reynier et al. [19]. The algorithm used is based on Miller et al. [23], it takes a Petri Net N in input (*P* represents the set of places, *T* the set of transitions, *I* is the bacwkard incidence mapping between places and transitions, *O* is the inward incidence mapping and *m0* is the initial marking of the Petri net) and constructs a tree in which nodes are labelled by ω

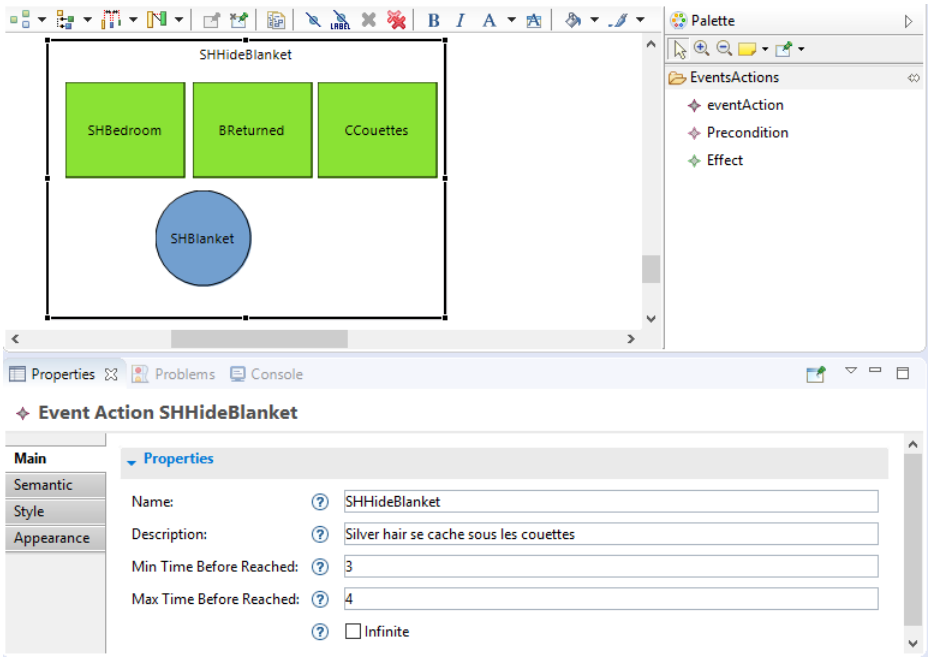

Fig. 2: Scenario editor view, showing how actions are created, the left part shows the states that are needed for our action to occur (in green) and the states that are created by it (in blue) for our example and the right part is the palette, allowing to add new actions, their preconditions and their effects.

Algorithm 1 Monotone Pruning Algorithm for Interactive Scenario based Petri Nets

Require: A Petri Net $N = (P, T, I, O, m_0)$, a scenario S that associates to T the corresponding actions.

Ensure: A labelled tree $C = (X, x_0, B, \Lambda)$ and a partition $X = Act \uplus Inact$ such that $\Lambda(Act) = MCS(N)$

Let x_0 be a new node such that $\Lambda(x_0) = m_0$;
$X := \{x_0\}; Act := X; Wait := \{(x_0, t)\} A(x_0) \rightarrow$ t $\cdot|\Lambda(x_0)\}; B = \emptyset$;
while Wait $\neq \emptyset$ **do**
  Pop (n', t) from $Wait$
  if $n' \in Act$ **then**
    $m := Post(\Lambda(n'), t)$
    Let n be a new node such that $\Lambda(n) = Acc(\Lambda(Ancestor_c(n') \cap Act), m)$;
    X += $\{n\}$; B += $\{(n', t, n)\}$;
    if $\Lambda(n) \not\leq \Lambda(Act)$ **then**
      Act -= $\{x | \exists_y \in Ancestor_c(x).\Lambda(y) \leq \Lambda(n) \wedge (y \in Act \vee y \notin Ancestor_c(n))\}$;
      Act += $\{n\}$;
      $Wait$ += $\{(n, u)|$ n $\rightarrow$ u $\cdot\}$ if $u \in n.availableActions()$;
    end if
  end if
end while
return $C = (X, x_0, B, \Lambda)$ and $(Act, Inact)$

markings and edges represents transitions, which, in our case, corresponds to an action that can be conducted by players or by the game master. The monotone pruning consists in deactivating branches where loops occurs, when we reach a state that has already been visited and replacing markings with omega-markings to represent the loop in the Petri Net. The main difference with the original K&M algorithm is that when we compute the marking of new nodes, we also calculate time labels associated as described in the previous section. Due to the nature of our representation of Interactive Scenarios, we want some actions to be deactivated when they are used or if they are part of a Choice that occurred, thus, we added to the algorithm a way to prune those actions from happening by verifying if it is still available according to the rules we defined. Only if the action corresponding to the transition is not available, we don't add it to the list of tuple Node/Transition that we want to visit (Wait).

Figure 4 shows the graph obtained after applying the algorithm on our Petri Net. We can see that Silver Hair can go from the kitchen to the bedroom and if she drank the bear's milk, she can chose to soil the bear's blanket before their return. In this example, we associated to each action the following duration bounds denoted by Action(minDuration, maxDuration), bounds are expressed in seconds : VisitBears (60-70), GoKitchen (5-10), GoBedroom (10-15), BearsReturn (10-15), HideCloset (10-15), HideBlanket (10-15), MSoilBlanket (30-40), DrinkMilk (10-30). To give a simple example of how time is calculated, we will consider the Petri net represented in Figure 3.

For our example, we will represent tokens by the name

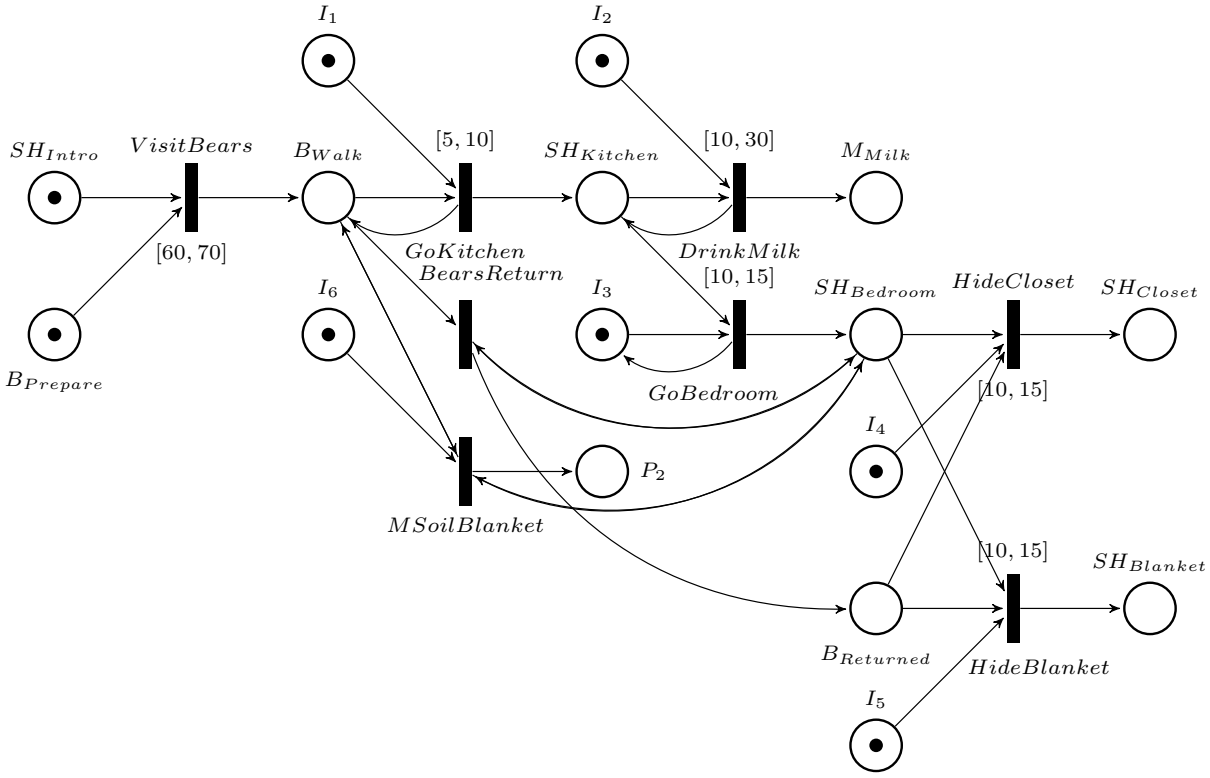

Fig. 3: Petri Net obtained by applying the method used by Pradin-Chezalviel et al. [18] on the Linear Logic sequent specifying the story of *Silver Hair*. Some labels have been omitted for brevity.

of the place they are in and their time annotations *placeName(min, max)*. At the beginning of a scenario, tokens that are already present are considered to have their time annotation to 0, thus we can represent the initial token of SHIntro by *SHIntro(0,0)*. In our example, the only transitions that can be taken at the beginning is VisitBears, in order to execute it we need to consume the token $SH_{Intro}(0, 0)$, and the token $B_{Prepare}(0, 0)$, VisitBears will then produce a new token in B_{Walk}. Because the action associated to this transition has estimated duration bounds equal to (60, 70), we need to set the annotation of our new token according to that, because the token with the highest min and max bounds in consumed tokens are 0, we add the time estimations of the action to the annotations of the token and obtain a new Token $B_{Walk}(60, 70)$, the transition GoKitchen becomes available, to use it we need to consume the Token $I_1(0, 0)$ and the newly created Token $B_{Walk}(60, 70$. Because our highest min and max bounds corresponds to (60, 70) we add it to the estimated durations of GoKitchen (5, 10) and obtain a new Token $SH_{Kitchen}(65, 80)$. The algorithm uses this process during all the transitions in order to compute the final estimation of the branch, if a place with normal tokens is replaced with a one with omega-markings when a loop occurs, the omega-marking annotation calculations follows the same principle, however, because an omega-marking represents a loop, this calculation only represents the lower bounds of time estimation and considers that the action is done only once, thus, we need to assume that players won't repeat an action many times in order for our calculation to be accurate. A solution to this problem would be to define an average or a min and max bound of times an infinite action is done by players, and change our calculation of time associated to omega-markings and infinite actions to take this into account. In practice, this sort of problem link to erratic user behavior will be solved during the play session by the narrator, whose task is to gently put back stray players on the narrative path.

For this algorithm we need to create a tree *C* (*x0* is the initial node of the tree and $\Lambda(n)$ represents the function that associate a marking to a given node *n*, we need to keep track of the lists of active nodes (*Act*) that can be visited and inactive nodes (*Inact*) that have been pruned, we also need to create a list of transitions that we want to visit (*Wait*), and a list that record which transition have been fired to reach a node from another one (*B*).'

The first step of the algorithm is to create a node that is labelled with the initial marking (x_0) of the Petri Net (line 2), we add this node to the set of nodes of the tree, we then add to Wait all the fireable transitions from the node. Once this initialization step is done, we need to iterate while the list Wait is not empty.

This loop on line 3 consists in poping the first tuple node(n) / transition(t) from Wait (line 4), we then verify that the node n is in the list of active nodes (line 5), if it is not the case, we iterate to the next tuple, if the node is still active, we

create a new marking that corresponds to the one associated to n after the transition has been fired, which correspond to the marking obtained when tokens needed for the transition are consumed, and new ones are created with corresponding time annotations (line 6). We then apply to this marking the Acceleration function to modify the marking, this function replaces some markings with omega-markings as described by Miller et al. [23], we create a new node (n) with this marking that we add to list of nodes from the tree, we also save the transition to B (line 7-8).

We then check if the marking of the node is not inferior or equal to one of the active nodes[2], and that all non infinite-transitions that have been used before arriving to n and the active node are the same (line 9). If it is the case it means that there are markings in active nodes that are smaller than the new marking and that the same transitions are available for both of them. Thus, we can prune those nodes by making them inactive (line 10), because they correspond to the same scenario (same actions happened in a different order), or because we reached a loop on a certain transition. We then add the new node n in the list of active nodes and we add all fireable transitions from n in Wait (line 12). Because of the nature of Petri nets, transitions cannot be disabled. Thus we need to verify if the transitions we want to add in Wait have not been used before if we defined them as not infinite. To ensure that we wont use transitions that are not supposed to be used more than once that are part of a Choice in which we already used a transition. In the case where the marking of n is inferior to one of the active nodes, we do nothing and go back to the beginning of the loop while Wait is not empty. When wait is empty, we can then return the tree obtained, active leafs will correspond to all possible outcomes for our scenario, the duration of the scenario corresponds to the latest time annotations.

This allows to generate the minimal coverability set (MCS) of the Petri net. This algorithm limits the exploration by pruning nodes that have already been explored in other branches. Thus, this algorithm is particularly interesting in our case as it prunes infinite paths and simple actions reordering while still allowing to validate all the possible discourses. Once the coverability set has been generated, we look at the active leafs of the tree, check whether they correspond to a desired ending and that the calculated duration correspond to the aimed one.

From the user perspective, the developed tool allows the author to quickly identify which paths leads to undesired untying or doesn't respect requirements in terms of duration by showing them in red for example. The author can then correct the path once they identified the problem and rerun the tool to check if their correction was sufficient.

V. Related Works

The formalisation of narratives has also attracted interest from researchers for their role as knowledge structures [24], [25]. Early models of story specification and narrative generation have been based on logical formalisms [26], or classic logic programming [27], but lacked the ability to reason about state and causality afforded by Linear Logic. Martens et al. [13], [28] have addressed this shortcoming proposing to use a form of linear logic programming for specifying stories in a declarative manner and generate and pilot interactive narratives. The objective of this system is closer to planning-based approaches than ours which focuses on verification, but it nevertheless provides a uniform logical framework to account for specification and generation of causally sound narratives. More recently, Azad et al. [29] tackle the issue of scheduling multiplayer interactive narrative by using Mixed-Interger Linear Programming. They reason under temporal constraints. On the contrary of our proposed approach they compute every path even dead-end and do not reason over resource allocation mechanisms.

Petri Nets have also been previously used for the authoring and piloting of interactive narratives. For instance, Balas et al. [30] have defined a customized version of Hierarchical Petri Nets for modeling the story and staging the narrative unfolding within a game engine. However, their model does not account for narrative actions which have a duration, a common shortcoming of early generative systems. Addressing this issue in a planning-based prototype fully integrated in a game engine, Porteous et al. [11] use Temporal Planning to synchronize various agents actions by taking into account their duration which solves concurrency issues at the staging level. We believe that our approach will allow us to solve the same problem by taking into account action duration at the authoring stage.

VI. Conclusions

We have described a system for authoring and verifying time constrained interactive narratives. Whilst the applications we consider take place in Virtual Reality, this work can generalize to authored narratives with one or several participants sharing the same narrative knowledge, driven by a story narrator.

Future work will tackle the issue of the piloting tool. Whilst we plan to use a similar underlying approach for representing narratives, the visual representation of the unfolding narrative for monitoring purposes will have to be simplified from current formalisms to be more usable for narrators (as early feedback suggests). One possibility is to rely on the structures described by Trabasso et al. [31], which evidences mental models of story understanding which may be more intuitively understood by untrained operators.

Acknowledgments

We would like to thank Alyssa Landry[3] Narrative Director at Incarna, for providing the Interactive Fiction piece *Silver Hair and the Three Bears* which we modified for our case study. Martín Diéguez is funded by the ANR ASTRID-Maturation Project STRATEGIC. Thomas Cabioch is funded by the DGA in France, for work in cooperation with the UK DSTL.

[2]This means that the amount of tokens in each place must be inferior or equal to the one of its corresponding place in the other marking

[3]https://alyssalandry.com/

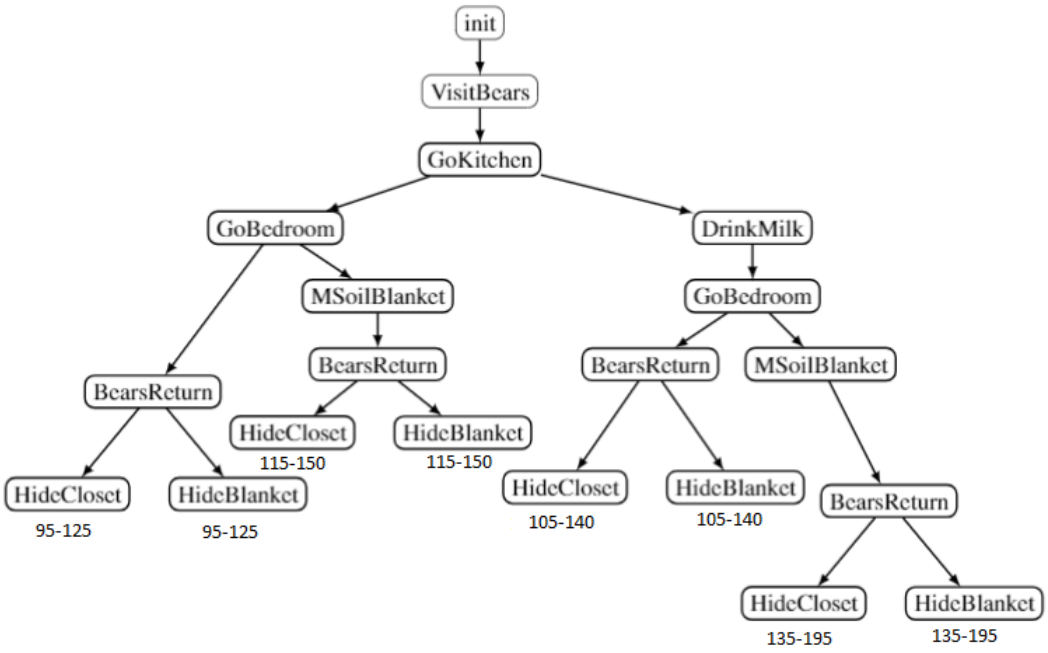

Fig. 4: Coverability graph obtained after applying the Karp and Miller with pruning algorithm from Reynier et al. [19], nodes represents narrative actions and arrows their ordering, a node followed by multiple arrows, shows that two or more actions are possible, numbers below leaves corresponds to the computed min and max duration bounds of the branch

REFERENCES

[1] J. H. Murray, *Hamlet on the Holodeck*. The MIT Press, 1998.

[2] M. Cavazza, J.-L. Lugrin, D. Pizzi, and F. Charles, "Madame bovary on the holodeck," in *MULTIMEDIA '07*, 2007.

[3] J.-Y. Girard, "Linear Logic," *Theoretical Computer Science*, vol. 50, no. 1, pp. 1–102, 1987.

[4] J.-Y. Girard, "Linear Logic: its syntax and semantics," in *Workshop on Advances in Linear Logic* (C. U. Press, ed.), 1995.

[5] M. Cavazza and D. Pizzi, "Narratology for interactive storytelling: A critical introduction," in *TIDSE'06*, Lecture Notes in Computer Science, Springer, 2006.

[6] R. Barthes, "Introduction à l'analyse structurale des récits," *Communications*, vol. 8, no. 1, pp. 1–27, 1966.

[7] G. Genette, *Narrative Discourse : An Essay in Methods*. 1979.

[8] S. Chatman, *Story and Discourse: Narrative Structure in Fiction and Film*. 1980.

[9] A. J. Greimas, *Sémantique structurale: recherche et méthode*. Larousse, 1966.

[10] R. M. Young, "Notes on the use of plan structures in the creation of interactive plot," in *Narrative Intelligence: Papers from the AAAI Fall Symposium*, AAAI Press, 1999.

[11] J. Porteous, J. Teutenberg, F. Charles, and M. Cavazza, "Controlling narrative time in interactive storytelling," in *AAMAS '11*, pp. 449–456, 2011.

[12] A.-G. Bosser, M. Cavazza, and R. Champagnat, "Linear Logic for non-linear storytelling," in *ECAI 2010*, vol. 215, IOS Press, 2010.

[13] C. Martens, "Ceptre: A language for modeling generative interactive systems," in *AIIDE'15*, 2015.

[14] A.-G. Bosser, P. Courtieu, J. Forest, and M. Cavazza, "Structural analysis of narratives with the Coq proof assistant," in *ITP'11*, 2011.

[15] F. Collé, R. Champagnat, and A. Prigent, "Scenario analysis based on Linear Logic," in *ACM SIGCHI Advances in Computer Entertainment Technology (ACE)*, ACM press, 2005.

[16] K. D. Dang and R. Champagnat, "An authoring tool to derive valid interactive scenarios," in *Intelligent Narrative Technologies*, 2013.

[17] R. Champagnat, "Optimisation d'une séquence de franchissement de transitions dans un réseau de petri t-temporisé," in *MOSIM'O3 (MOdlisation et SIMulation)*, pp. 94–100, 2003.

[18] B. Pradin-Chezalviel, R. Valette, and L. A. Kunzle, "Scenario durations characterization of t-timed petri nets using linear logic," in *PNPM '99*, pp. 208–217, IEEE Computer Society, 1999.

[19] P.-A. Reynier and F. Servais, "Minimal Coverability Set for Petri Nets: Karp and Miller Algorithm with Pruning." Jan. 2011.

[20] M. Cavazza and R. M. Young, "Introduction to interactive storytelling," *Handbook of Digital Games and Entertainment Technologies*, 2017.

[21] M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," *Entertainment Computing*, vol. 2, no. 4, pp. 205 – 213, 2011.

[22] K. D. Dang, S. Hoffmann, R. Champagnat, and U. Spierling, "How authors benefit from linear logic in the authoring process of interactive storyworlds," in *ICIDS'11*, pp. 249–260, Springer-Verlag, 2011.

[23] R. Miller and M. Shanahan, "Narratives in the Situation Calculus," *Journal of Logic and Computation*, vol. 4, pp. 513–530, 1994.

[24] A. Kakas and R. Miller, "A simple declarative language for describing narratives with actions," *Journal of Logic Programming*, vol. 31, pp. 157–200, 1997.

[25] R. Reiter, "Narratives as programs," in *KR'00*, Morgan Kaufmann Publishers, 2000.

[26] R. R. Lang, "A declarative model for simple narratives," in *Narrative Intelligence: Papers from the AAAI Fall Symposium*, AAAI Press, 1999.

[27] D. Grasbon and N. Braun, "A morphological approach to interactive storytelling," in *CAST'01*, 2001.

[28] C. Martens, A.-G. Bosser, J. F. Ferreira, and M. Cavazza, "Linear logic programming for narrative generation," in *LPNMR'13*, pp. 427–432, Springer, 2013.

[29] S. Azad, J. Xu, H. Yu, and B. Li, "Scheduling live interactive narratives with mixed-integer linear programming," 10 2017.

[30] D. Balas, C. Brom, A. Abonyi, and J. Gemrot, "Hierarchical petri nets for story plots featuring virtual humans," in *AIIDE'08*, 2008.

[31] T. Trabasso and L. L. Sperry, "Causal relatedness and importance of story events," *Journal of Memory and Language*, vol. 24, no. 5, pp. 595 – 611, 1985.