







# Plot Composition by Mapping Situation Calculus Schemas into Petri Net Representation

Edirlei Soares de Lima<sup>1,2</sup> , Antonio L. Furtado<sup>3</sup> , Bruno Feijó<sup>3</sup> ,  
and Marco A. Casanova<sup>3</sup> 

<sup>1</sup> IADE, Universidade Europeia, Av. D. Carlos I 4, 1200-649 Lisbon, Portugal  
edirlei.lima@universidadeeuropeia.pt

<sup>2</sup> UNIDCOM/IADE, Av. D. Carlos I 4, 1200-649 Lisbon, Portugal

<sup>3</sup> Department of Informatics, PUC-Rio, R. Marquês de São Vicente 225, Rio de Janeiro, Brazil  
{furtado, casanova}@inf.puc-rio.br

**Abstract.** In this paper we propose a new plot composition method based on situation calculus and Petri net models, which are applied, in a complementary fashion, to a narrative open to user co-authorship. The method starts with the specification of situation calculus schemas, which allow a planning algorithm to check if the specification covers the desired cases. A Petri net is then automatically derived from the schemas in a second phase, guiding interactive plot generation and dramatization. The applicability of the proposed method is validated through the implementation of an interactive storytelling system capable of representing the generated Petri net models using 2D graphics and animations.

**Keywords:** Petri net · Situation calculus · Interactive storytelling · Plot generation · Dramatization

## 1 Introduction

Readers enjoy a far more pleasant experience with narratives in which they are invited to participate as co-authors. This claim is convincingly expressed by Umberto Eco [8], when talking of “open works” and “works in movement,” i.e., works that deliberately leave decisions on the meaning of specific passages to the care of the reader. However, this ideal, which is hard to satisfy in book format, only now is truly reachable through interactive narratives developed for digital environments.

The most popular approach to interactive narrative, especially in narrative-driven games (e.g., *Heavy Rain* (2010) and *Detroit Become Human* (2018) by Quantic Dream), is the branching technique (also known as branching path stories [16]). In this technique, the player makes a choice at each branching point. The writer usually builds a rigid structure of branching points through a manually authored process without any consistency check. User decisions involve choosing which way to proceed at branching points, thus leading, knowingly or not, to a subsequent outcome. To identify such branching points, designers usually use Petri nets [3, 23], a graphically structured modeling technique for dynamic systems [22].

A more suitable and robust approach is to treat a narrative as an application process where, instead of a single linear plot, the designers (authors) define a fixed repertoire of predefined event-producing operations containing preconditions and post-conditions. These conditions enforce the intended conventions and preserve consistency when the users (readers performing as co-authors) are allowed to decide. Postconditions consist of facts asserted or retracted as the consequence of executing the operator. And as we argue in this paper, situation calculus [15] is a suitable modeling strategy that can exploit backward-chaining plan generation based on the operators above mentioned (e.g., using STRIPS [11]) to show what specific plots can emerge.

This paper presents a new plot composition method that combines both modeling techniques in a *complementary* fashion. We consider Petri nets from the perspective of event logs, as proposed by Wil van der Aalst [1]. Our method starts with specifying situation calculus schemas for a chosen process application. It then automatically derives a Petri net representation from these schemas, which is informative enough to be employed for interactive plot generation and dramatization. The method is analogously applicable to information system domains, where business transactions can be treated in the same way as narrative plots.

The contribution of our work is twofold. First, we shed light on the complementarity between situational calculus and Petri nets. Second, we have taken the process of composing interactive plots to a more robust and semantically consistent level. In our approach, situation calculus is used at specification time, enforcing integrity constraints, and checking if the specification allows all desirable use cases while disallowing undesirable cases. In the final step, the automatically generated Petri net allows visualizing the processes and effectively executing them.

The paper is organized as follows. Section 2 discusses related work. Section 3 presents our approach to deriving a Petri net model from a situation calculus model in a narrative domain. Section 4 explores the application of the proposed method in a fully implemented interactive storytelling system. Finally, concluding remarks are the object of Sect. 5.

## 2 Related Work

Situation calculus [15] provides a second-order logic method to formalize state transitions caused by event-producing operations. Petri nets, in turn, are commonly utilized in Process Mining work [1] for obtaining an implicit model of an application, by discovering the partial order requirements prevailing on a significant number of traces extracted from an execution log. For example, in [14], Petri net synthesis is preceded by a preliminary activity mining algorithm.

From [1] we borrowed and used in our first experiments [19] the simple introductory case of a request processing application, represented by a Petri net. When specifying the trial by combat application that serves as a running example in the present paper, we were able to end up with a structurally analogous Petri net representation, thus favoring the claim [6] that serious and entertainment applications can be treated by the same modelling formalisms.

Research involving situation calculus and Petri net formalisms includes in special [24], which proposes a formal ontology that highlights the correspondence between a

sequence of actions starting from an initial state in situation calculus, and a sequence of transition firings starting from an initial node in a Petri net. However, the aforementioned work focuses only on the analysis of structural properties of Petri nets and situation calculus models.

The relations between Petri nets and automated planning were also explored in previous research, such as [13], where a method to transform planning graphs into Petri nets is presented and used to demonstrate that Petri net unfolding, a form of partial order reduction, can be used to recognize independent planning subproblems. The transformation of Petri net models into planning problems was also explored in [2], which is the inverse of the process discussed in this paper. Plans are modelled as Petri nets in [25] and plans are used to produce workflows in [10] (re-calling that Petri nets can be viewed as a particular form of workflow).

Petri nets were also applied in interactive storytelling contexts. Riedl et al. [23] uses a specialized type of Petri net, called colored Petri net, to allow authors to manually model interactive narratives as a process in which multiple players can navigate through different narrative scenes. During dramatization, their system uses an execution algorithm that monitors for situations in which the Petri net fails to account for player actions. When a failure situation is identified, a planning algorithm is used to generate new narrative events to restore the integrity of the Petri net. In this paper, instead of relying on manually authored Petri nets, we focus on the automatic process of mapping situation calculus schemas into Petri net models.

Petri nets are also commonly used as modelling tools to design narratives and character behaviors in games. An example of work that employs manually authored Petri nets to represent narrative plots is presented by Balas et al. [3], which uses hierarchical Petri nets to define branching narratives for games. Lee and Cho [17] also proposed a quest generation method for games where quests are modeled as Petri nets, which are activated during a game session according to characters' goals and the current world state. A similar approach is explored by El-Sattar [9], who uses Petri nets as a state-based model to design narrative plots. The use of Petri nets to model and control individual characters in an interactive storytelling context is also explored by Brom and Abonyi [4], who utilize manually designed Petri nets to represent the narrative of the game.

Although Petri nets have been previously explored as a structure to represent narrative plots, most of the previous research focus on the use Petri nets as a modelling tool to allow authors to design interactive narratives. In this paper, we follow a different approach and focus on plot composition by automatically mapping situation calculus schemas into Petri net representations that are suitable for interactive dramatization.

## 3 From Situation Calculus to Petri Net Models

### 3.1 The Basic Situation Calculus Model

Situation calculus is a logical language used to represent and reason about dynamic worlds, which has been successfully applied to a variety of domains and problems, including narrative generation [6]. According to Kowalski [15], situation calculus, as a logic program, can be compactly expressed by the following two clauses, which define

what sentences  $P$  hold in the situation  $result(A, S)$  that is the result of the transition from state  $S$  by an action of type  $A$ :

$$\begin{aligned} holds(P, result(A, S)) &\leftarrow happens(A, S) \wedge initiates(A, S, P) \\ holds(P, result(A, S)) &\leftarrow happens(A, S) \wedge holds(P, S) \wedge \neg terminates(A, S, P) \end{aligned}$$

noting that the second clause of this second-order logic formulation avoids the exponential proliferation of first-order logic clauses, which constitutes the so-called frame problem by eliminating the need to specify every class of facts ( $P$ ) that are *not* affected by the execution of an operation ( $A$ ).

In turn, these situation calculus clauses suggest an elementary plan-generator, which can be thus expressed in natural language:

- A fact  $F$  holds if it is true in the initial state;
- $F$  holds if it is added as one of the effects of an operation  $Op$ , and the preconditions of  $Op$  hold at the current state;
- $F$  holds after the execution of an operation  $Op$  if it did already hold at the current state and if it is not deleted as one of the effects of  $Op$ .

and then translated into a Prolog program:

```
holds(Fact, [start]) :- initial_state(Fact), !.
holds(Fact, [Operation | Current_state]) :- added(Fact, Operation),
                                           precond(Operation, Current_state).
holds(Fact, [Operation | Current_state]) :- not deleted(Fact, Operation),
                                           holds(Fact, Current_state).
```

Although this elementary program is able to handle overly simple cases, such as the well-known monkey-and-bananas problem, it must be considerably expanded for practical usage, such as proposed in [5, 7].

As a general starting point to apply situation calculus in storytelling domains, one must specify static and dynamic schemas, which include the classes of facts that will eventually populate states, a set of facts describing the initial state, and a fixed repertoire of event-producing operations for performing state changes in conformance with the applicable integrity constraints. Each operation is defined in terms of pre-conditions, which consist of conjunctions of positive and/or negative terms expressing facts, and any number of post-conditions, consisting of facts to be asserted or retracted as the effect of executing the operation (cf. The STRIPS model [11]).

As an illustration, we shall concentrate on a simple narrative incident taken and adapted from the film *Excalibur*, directed, produced, and co-written by John Boorman in 1981. The incident can be thus summarized:

*Sir Gawain accuses Queen Guinevere of adultery. A trial by combat is announced, there being two candidate knights to claim the Queen's innocence: Lancelot, a worthy knight, famous for his many victories, and Perceval, who would be no less reputed in the future, but at that time still had little combat experience. The Queen would be vindicated if her defender could defeat the accuser, otherwise she would be condemned. The trial could be reinitiated if a last-minute replacement of defender chanced to occur.*

The entities involved in the incident and their properties are specified through a static schema:<sup>1</sup>

```
entity(person, pn).
entity(knight, kn).
attribute(knight, strength).
attribute(knight, loyal).
entity(accuser, an).
entity(defendant, dn).
attribute(defendant, has_defender).
entity(defender, dn).
entity(challenger, cn).
entity(offense, on).
relationship(accusation, [defendant, offense]).
attribute(accusation, vindicated).
attribute(accusation, condemned).
relationship(encounter, [challenger, defender]).
attribute(encounter, winner).
```

The event-producing operations are defined in a dynamic schema. For example, the operator for the *combat* event is defined below, noting that the *V* parameter identifies the winner, who can be either the accuser or the defending knight, depending on their strength:

```
operation(combat(A, K, D, O, V)).
precond(combat(A, K, D, O, V), (accusation(D, O), challenger(A),
defender(K), strength(K, Sk), strength(A, Sa),
if(Sk > Sa, V = K, V = A))).
added(encounter(A, K), combat(A, K, D, O, V)).
added(winner([A, K], V), combat(A, K, D, O, V)).
added(defendant(D), accuse(A, D, O)).
```

In order to conduct experiments, an initial state must be introduced, indicating the instances of the entity classes and the specific initial values of their properties:

```
person('Guinevere').
knight('Lancelot').
loyal('Lancelot', true).
strength('Lancelot', 200).
knight('Perceval').
loyal('Perceval', true).
strength('Perceval', 100).
knight('Gawain').
loyal('Gawain', false).
strength('Gawain', 150).
offense(murder).
offense(adultery).
```

Described in this fashion, the well-intentioned but still immature Perceval would stand no chance to defeat Gawain when playing the role of defender. This default result is evidenced in the plan-generated plot below, in which the last parameter of the combat operation (in boldface) indicates the winner:

<sup>1</sup> A complete description of the static and dynamic schemas used in this example is available at: <http://www.icad.puc-rio.br/~logtell/petri-net/schemas-trial-by-combat.pdf>.

```
accuse(Gawain, Guinevere, adultery), enter_challenger(Gawain, Guinevere,
adultery), enter_beginner_defender(Perceval, Guinevere, adultery), com-
bat(Gawain, Perceval, Guinevere, adultery, Gawain), condemn(Guinevere,
adultery).
```

On the contrary, Lancelot had what was required to triumph, thereby establishing the Queen's innocence:

```
accuse(Gawain, Guinevere, adultery), enter_challenger(Gawain, Guinevere,
adultery), enter_worthy_defender(Lancelot, Guinevere, adultery), com-
bat(Gawain, Lancelot, Guinevere, adultery, Lancelot), vindi-
cate(Guinevere, adultery).
```

Both for serious and for entertainment applications, the situation calculus model leads to the verification, by applying plan-generation, whether the proposed specification allows all desirable use cases and effectively disallows those which transgress the intended conventions [6]. On the other hand, the Petri net model, like other workflow engines, can be designed to run in a tightly restrictive mode, with the additional asset of the explicit determination of the workable sequences and of the branching points open to the user's choice – which strongly suggests that it is particularly qualified for the dramatization of interactive narratives.

In this paper, we argue that the two models are complementary to each other, to the point that the Petri net representation can be generated from the situation calculus model, over which an execution method, analogous to the standard token-based Petri net method, can be operated.

### 3.2 Deriving a Petri Net from a Situation Calculus Model

In the semi-formal terminology employed in this paper, a Petri net is a graph with two kinds of nodes: *places* (round nodes, either empty or containing exactly one token) and *transitions* (square nodes representing operations).<sup>2</sup>

We define a Petri net *edge* as triple  $(Op_1, Pn, Op_2)$ , with two (operational) transition nodes ( $Op_1$  and  $Op_2$ ) and an intervening place node ( $Pn$ ). Petri net edges are positioned so as to express a partial ordering in the execution of events, which may follow each other in linear sequences, possibly branching to form and-forks, or-forks, and-joins, and or-joins. We shall also consider a simple case of backward loop, allowing to return to a previous position and try different branching options.

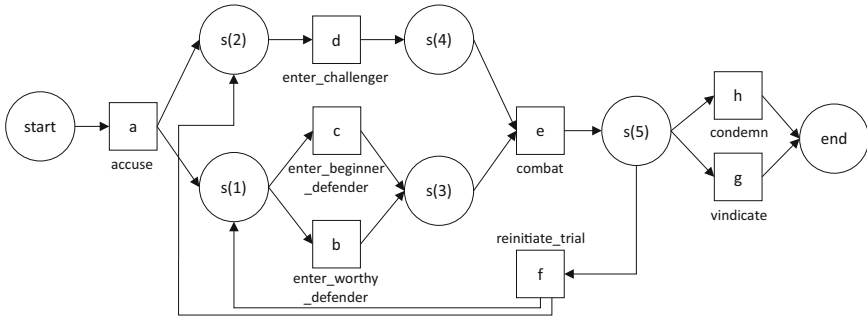
Our approach to generate a Petri net model is based on the observation that the ordering requirements of a Petri net can be derived from the situation calculus model. The first basic consideration is that there exists an edge connecting (through a  $Pn$  node)  $Op_1$  and  $Op_2$  if the post-conditions of  $Op_1$  have a non-empty intersection with the pre-conditions of  $Op_2$ . There is also an edge from  $Op_1$  to  $Op_2$  if some post-condition of  $Op_1$  cancels some post-condition of  $Op_2$  (thus causing a backward loop, whereby  $Op_2$  can be retried).

Forks occur when there are edges leading from an  $Op$  node into two or more nodes  $Op_1, Op_2, \dots, Op_n$ . A fork is an or-fork if  $Op_1, Op_2, \dots, Op_n$  contain either incompatible pre-conditions or redundant post-conditions. Incompatibility typically results from

<sup>2</sup> A comprehensive and still useful classic survey of the Petri net formalism is provided in [21].



Once the clausal representation is generated, a visual representation of the Petri net can be created, as shown in Fig. 1.



**Fig. 1.** Petri net drawn from the clausal representation derived from the situation calculus model.

Operations coming from and-forks can be executed in any order. In addition, they might be executed in parallel to simulate narrative events taking place at the same time. As a preliminary consideration, note that, by construction, or-forks stem from place nodes, and or-joins always converge to a single place node. Since place nodes can contain at most one token, or-type branching is restricted – as should be expected – to the selection of a single option. In contrast, and-forks stem from operation nodes, and and-joins converge to an operation node. Differently from place nodes, operation nodes are able to emit tokens to all outgoing place nodes (one token for each).

The generated Petri net is ready to be executed in order to establish the plot for an interactive narrative. According to the standard token-based Petri net process, executing a Petri net begins by placing a token in the start place node. The place node is then activated which signifies that the token is consumed, and the single operation node attached to the start node is enabled. In the next steps, successive place nodes are activated after receiving tokens from enabled nodes, and some operation node to which all incoming place nodes are active (i.e., contain a token) is chosen to be enabled. The process ends when some operation node connected to the end place node is reached.

An interactive trace-generation program that allows users to traverse through the Petri net generated for the trial by combat is available online at:

<http://www.icad.puc-rio.br/~logtell/petri-net/trial-by-combat/>.

## 4 Interactive Storytelling Application

In order to validate the applicability of our method, we implemented a full interactive storytelling system capable of representing the generated Petri net models using 2D graphics and animations.

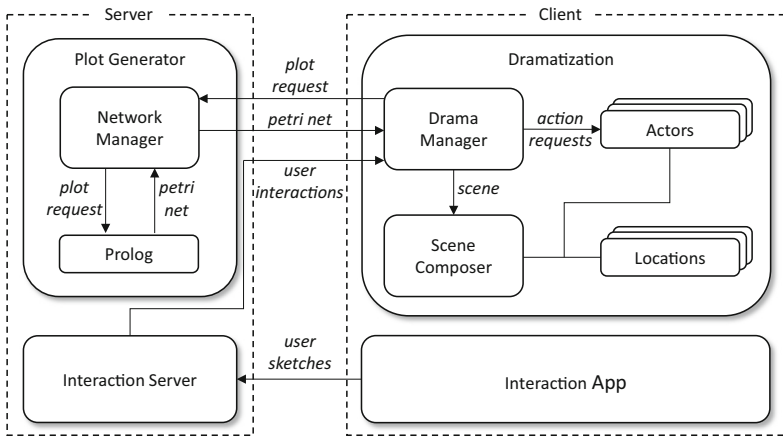
### 4.1 System Architecture

The architecture of our interactive storytelling system is based on a client-server model (Fig. 2), where the server is responsible for the generation of the plot (Petri net model) and



the client handles the dramatization of the story. On the server-side, the Network Manager receives plot requests from clients and uses the Prolog implementation described in the previous sections to generate a Petri net, which is then sent to clients for dramatization. On the client-side, the Drama Manager interprets and controls the execution of the Petri net by sending action requests to virtual Actors. The process of composing scenes for dramatization (i.e., selecting the Actors and Locations to show) is performed by the Scene Composer, which is constantly being informed by the Drama Manager about the type of scene being dramatized.

User interaction is handled by the Interaction App module, which is implemented as a mobile app that uses a Convolutional Neural Network classifier to identify hand-draw sketches (see [20] for more details about the sketch recognition process). Once a sketch is recognized, its identification class is sent to the Interaction Server through a TCP/IP network message. The Interaction Server module is responsible for receiving and interpreting the sketch classes sent by clients. Two interaction modes are supported: (1) single user mode, in which the first valid user sketch received by the system is immediately used as the interference choice to be incorporated into the story; and (2) voting mode, in which the Interaction Server collects all users' sketches during a certain time and then selects one through a voting process.



**Fig. 2.** Architecture of our interactive storytelling system.

Multiple programming languages were used in the implementation of our interactive storytelling system. As described in the previous sections, the process of generating plots in the Petri net model is implemented in Prolog. However, the Plot Generator also includes an additional module called Network Manager, which is implemented in C# and provides network communication capabilities to the system, allowing us to implement

the plot generation process as a service provided by a network server. On the client-side, the dramatization system is implemented in Lua<sup>3</sup> using the Löve 2D framework,<sup>4</sup> which provides the graphical functionalities needed to create visual representations for the story. The interaction process is implemented in Java as an Android app, which communicates with a Web service implemented in PHP. See [12] for more details about the design of the sketch-based interaction system.

The Petri net representation of the plot created by the Plot Generator consists of a directed graph  $G = (V, E)$ , where  $V$  is a set of nodes  $\{v_1, v_2, \dots, v_n\}$  and  $E$  is a set of edges  $\{e_i = (v_i, v_j), \dots, e_m = (v_k, v_w)\}$ . Each node  $v_i$  is a pair  $(id_i, ev_i)$ , where  $id_i$  is a unique name that identifies the node  $v_i$  and  $ev_i$  is an event description in a predicate format for *transition* nodes (e.g., `accuse(a, d, o)`), or the constant `nil` for *place* nodes (as described in Sect. 3.2, places are nodes that can contain tokens and transitions are nodes that represent operations).

When encoding the Petri net to be sent to the dramatization system, the graph is simplified as a set of edges, where each edge is represented in the format  $[id_i: ev_i, id_j: ev_j]$ . For example, the initial edges of the Petri net generated for the trial by combat (connecting nodes *start*, *a*, *s(1)*, and *s(2)*), as illustrated in Fig. 1, can be described as:

```
[start:nil, a:accuse(a, d, o)]
[a:accuse(a, d, o), s(1):nil]
[a:accuse(a, d, o), s(2):nil]
```

## 4.2 Interactive Dramatization

The process of dramatizing the Petri net representation of the plot involves a simple step-wise algorithm that controls the execution of the story by updating a list of active events according to a standard token-based execution approach. As described in Algorithm 1, function `Execute-PetriNet-Step` receives by parameter a Petri net `PN` and a list `C` with the nodes that were executed in the previous step of the algorithm (for the first step: `C={start}`). The algorithm performs all the operations to activate place nodes and transition nodes for a single iteration of the execution process. The narrative events associated with activated transition nodes are added to set `A`, which is returned when the execution of the iteration ends. The set of narrative events returned by a single call of function `Execute-PetriNet-Step` represents the parallel events that take place during a certain point of the narrative. When the dramatization of these events ends, function `Execute-PetriNet-Step` can be called again to obtain the next narrative events for dramatization. If an empty set is returned, the narrative ends.

<sup>3</sup> Lua is a well-known programming language developed at the Pontifical Catholic University of Rio de Janeiro, Brazil (<http://www.lua.org/>).

<sup>4</sup> <https://love2d.org/>.

**Algorithm 1.** Petri net execution algorithm.

---

```

1.  function Execute-PetriNet-Step(PN, C)
2.    A =  $\emptyset$ ;
3.    for each node V in C do
4.      if PN[V] is a PLACE then
5.        N = number of edges in PN[V];
6.        if N is greater than 0 then
7.          if N is 1 then
8.            S = first edge in PN[V];
9.          else
10.           S = get selected edge from PN[V] based on user interaction;
11.          end
12.          TA = get number of tokens available in parent nodes of PN[S];
13.          TN = get indegree of PN[S];
14.          if TA is greater or equal than TN then
15.            Consume TN tokens from the parent nodes of PN[S];
16.            Add S to A;
17.          end
18.        end
19.      else if PN[V] is a TRANSITION then
20.        for each edge E in PN[V] do
21.          if PN[E] is a PLACE then
22.            Add a token to place PN[E];
23.            L = Execute-PetriNet-Step(PN, {E});
24.            for each node W in L do
25.              Add W to A;
26.            end
27.          end
28.        end
29.      end
30.    end
31.    return A;
32.  end

```

---

All the assets used for dramatization (e.g., character animations, background images, and audio files) are defined in a library manually constructed for the domain of a specific story. The context library is a 5-tuple  $L = (\gamma, \alpha, \beta, \delta, \pi)$ , where:

- $\gamma$  is a set that defines the actors of the story. Each actor has a name and a set of actions, which are represented by animations in a sprite sheet format;
- $\alpha$  defines the locations of the story. Besides associating each location with a background image and a soundtrack, it also defines a set of waypoints where actors can be placed during the scene composition process;
- $\beta$  defines the characters' dialogs (text and audio);
- $\delta$  is a set that defines the interaction points of the story. Each interaction point is associated with a set of interactive objects, which are represented by the classes of sketches that can be used by users to interact at each interaction point. The interaction points also include a set of instructions to guide the user during the interaction;
- $\pi$  establishes values for the variables present in the events of the Petri net.

In our implementation, the context library is defined in an XML file. The library used for the trial by combat example is available at: <http://www.icad.puc-rio.br/~logtell/petri-net/context-trial-by-combat.xml>.

During the dramatization of the story, our system generates 2D animations in real time according to the actions performed by the virtual actors. An automatic virtual camera

maintains the active actors always centered in the image frame while they move around the virtual world. When more than one actor is involved in the action, the camera will target the center of the scene, which is calculated based on the positions of all characters that are participating in the event.

User interaction occurs at or-fork nodes of the Petri net. When a node of this type is activated, users are instructed by the virtual characters to interact by drawing specific objects in the interaction app. The instructions are defined in the context library and comprise a set of phases (text and audio), which are repeated until the user draws a valid object (single user interaction mode) or during a certain time frame (voting interaction mode). When the user's choice is identified, the corresponding transition node is selected to be activated (as indicated in line 10 of Algorithm 1). An example of a user interaction moment for the trial by combat is illustrated in Fig. 3, showing the user's decision whether to help Gawain or Perceval, in the combat by drawing a spear or a sword (i.e., the weapons used by each character). A complete video demonstration of the trial by combat example is available at:

<https://www.youtube.com/watch?v=qI2TeBrhycc>.



**Fig. 3.** User interaction moment in the trial by combat: (a) shows the dramatization system instructing the user to draw a spear to assist Gawain or a sword to help Perceval; and (b) shows that the user chose to draw a sword in the interaction app.

## 5 Concluding Remarks

We claim that our research thus far has already revealed the advantages of the complementary use of situation calculus and Petri nets. The situation calculus model is most convenient to start with, allowing to investigate through a planning algorithm the appropriateness of the initial specification.

On the other hand, mapping the situation calculus schemas into the graphic structure of a Petri net permits the identification of the points where the narrative process proceeds along branching sequences, so as to recognize and explicitly annotate the occurrence

of forks, joins and loops. This kind of information is most helpful to guide interactive plot generation/dramatization and is indispensable if dramatization is done by putting together video-recorded sequences [18], given that scene transition often poses nontrivial adjustment problems that cannot be left to be solved at runtime.

In addition, one must recall the relevance of this method to game design [16], where multiple-ending and branching path storytelling mark increasingly advanced stages in the interactivity spectrum, with remarkable examples, such as the Mass Effect trilogy (BioWare, 2007–2012) and The Witcher trilogy (CD Projekt RED, 2007–2015). Due to their predictability, handcrafted branching narrative structures are still dominant in the game industry. However, we believe that more open approaches to interactive storytelling, such as our method, can expand the boundaries of game narratives towards new forms of interactive experiences. The situation calculus used at the specification stage of our approach gracefully deals with unpredictability because the complete sequence of outcomes is not explicit in the set of operators but can be easily verified.

Much work, however, remains to be done. As a proof of concept, we initiated this project working upon an oversimplified example. Accordingly, we do not claim that the current prototype can handle all problems associated with more complex applications and intricate Petri net schemes. For instance, Petri net loops caused by iterative actions have not been considered. Also, we could enrich the information kept at each Petri net node by collecting user behavior data during a run and analyzing them to regulate the branching options. Another essential future investigation is to address non-deterministic events, i.e., events that can have more than one outcome. Finally, we also plan to explore authoring systems to support story writers, and to conduct comprehensive user satisfaction tests involving writers (i.e., authors) and players (i.e., co-authors) in future works.

**Acknowledgements.** We want to thank CNPq (National Council for Scientific and Technological Development) and FINEP (Funding Agency for Studies and Projects), which belong to the Ministry of Science, Technology, and Innovation of Brazil, for the financial support.

## References

1. Aalst, W.V.D.: Process mining. *Commun. ACM* **55**(8), 76–83 (2012). <https://doi.org/10.1145/2240236.2240257>
2. Agostinelli, S., Maggi, F.M., Marrella, A., Mecella, M.: Verifying petri net-based process models using automated planning. In: *Proceedings of the 2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*, pp. 44–53. IEEE Press, New York (2019). <https://doi.org/10.1109/EDOCW.2019.00021>
3. Balas, D., Brom, C., Abonyi, A., Gemrot, J.: Hierarchical petri nets for story plots featuring virtual humans. In: *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'08)*, pp. 2–9. AAAI Press, Menlo Park (2008)
4. Brom, C., Abonyi, A.: Petri-nets for game plot. In: *Proceedings of AISB Artificial Intelligence and Simulation Behaviour Convention* **3**, pp. 6–13 (2006)
5. Ciarlini, A.E.M., Barbosa, S.D.J., Casanova, M.A., Furtado, A.L.: Event relations in plan-based plot composition. *Computers in Entertainment* **7**(4), 55 (2009). <https://doi.org/10.1145/1658866.1658874>

6. Ciarlini, A.E.M., Casanova, M.A., Furtado, A.L., Veloso, P.A.S.: Modeling interactive storytelling genres as application domains. *J. Intelligent Inf. Systems* **35**(3), 347–381 (2010). <https://doi.org/10.1007/s10844-009-0108-5>
7. Ciarlini, A.E.M., Pozzer, C.T., Furtado, A.L., Feijó, B.: A logic-based tool for interactive generation and dramatization of stories. In: *Proceedings of the International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, pp. 133–140. ACM Press, New York (2005). <https://doi.org/10.1145/1178477.1178495>
8. Eco, U.: *The Open Work*. Harvard University Press, Cambridge (1989)
9. El-Sattar, H.K.H.A.: A new framework for plot-based interactive storytelling generation. In: *Proceedings of the 2008 Fifth International Conference on Computer Graphics, Imaging and Visualisation*, pp. 317–322. IEEE Press, New York (2008). <https://doi.org/10.1109/CGIV.2008.50>
10. Fernandes, A., Ciarlini, A.E.M., Furtado, A.L., Hinchey, M.G., Casanova, M.A., Breitman, K.K.: Adding flexibility to workflows through incremental planning. *Innovations Syst. Softw. Eng.* **3**(4), 291–302 (2007). <https://doi.org/10.1007/s11334-007-0035-y>
11. Fikes, R.E., Nilsson, N.J.: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* **2**(3–4), 189–208 (1971). [https://doi.org/10.1016/0004-3702\(71\)90010-5](https://doi.org/10.1016/0004-3702(71)90010-5)
12. Gheno, F., Lima, E.S.: História viva: a sketch-based interactive storytelling system. In: *Proceedings of the XX Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2021)*, pp. 116–125. SBC, Porto Alegre (2021)
13. Hickmott, S., Rintanen, J., Thiebaut, S., White, L.: Planning via petri net unfolding. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1904–1911. AAAI Press, Menlo Park (2007)
14. Kindler, E., Rubin, V., Schäfer, W.: Process mining and petri net synthesis. In: Eder, J., Dustdar, S. (eds.) *BPM 2006*. LNCS, vol. 4103, pp. 105–116. Springer, Heidelberg (2006). [https://doi.org/10.1007/11837862\\_12](https://doi.org/10.1007/11837862_12)
15. Kowalski, R., Sadri, F.: Reconciling the event calculus with the situation calculus. *J. Logic Programming* **31**(1–3), 39–58 (1997)
16. Lebowitz, J., Klug, C.: *Interactive Storytelling for Video Games: A Player-Centered Approach to Creating Memorable Characters and Stories*. Focal Press, Waltham (2011)
17. Lee, Y.-S., Cho, S.-B.: Dynamic quest plot generation using Petri net planning. In: *Proceedings of the Workshop at SIGGRAPH Asia (WASA '12)*, pp. 47–52. ACM Press, New York (2012). <https://doi.org/10.1145/2425296.2425304>
18. de Lima, E.S., Feijó, B., Furtado, A.L.: Video-based interactive storytelling using real-time video compositing techniques. *Multimedia Tools Appl.* **77**(2), 2333–2357 (2017). <https://doi.org/10.1007/s11042-017-4423-5>
19. Lima, E.S., Furtado, A.L., Feijó, B., Casanova, M.A.: A note on process modelling: combining situation calculus with petri nets. Technical Report 01/2022, Department of Informatics, PUC-RIO, Rio de Janeiro (2022). <https://doi.org/10.17771/PUCRio.DIMcc.59758>
20. Lima, E.S., Gheno, F., Viseu, A.: Sketch-based interaction for planning-based interactive storytelling. In: *Proceedings of the XIX Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2020)*, pp. 348–356. IEEE Press, New York (2020). <https://doi.org/10.1109/SBGames51465.2020.00029>
21. Peterson, J.L.: Petri nets. *ACM Comput. Surv.* **9**(3), 223–252 (1977). <https://doi.org/10.1145/356698.356702>
22. Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Upper Saddle River (1981)
23. Riedl, M., Li, B., Ai, H., Ram, A.: Robust and authorable multiplayer storytelling experiences. In: *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'11)*, pp. 189–194. AAAI Press, Menlo Park (2011)

24. Tan, X.: SCOPE: A situation calculus ontology of Petri Nets. In: Proceedings of 6th International Conference of Formal Ontology in Information Systems, Toronto, Canada, pp. 227–240 (2010)
25. Ziparo, V.A., Iocchi, L., Nardi, D., Palamara, P.F., Costelha, H.: Petri net plans - a formal model for representation and execution of multi-robot plans. In: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 79–86 (2008)