Petri Nets for Game Plot

Cyril Brom and Adam Abonyi
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, Prague 1 – 118 00, Czech Republic
brom@ksvi.mff.cuni.cz adam.abonyi@gmail.com

Abstract

We have developed a technique for authoring a nonlinear plot and for managing a story according to the plot in an interactive story-based virtual reality application. The technique exploits Petri Nets that alter reactive plans controlling individual actors. Its main advantage is that it allows for iterative design and copes with large virtual worlds inhabited with tens of actors. In this paper, we describe the technique and a prototype application.

1 Introduction

This paper concerns itself with a problem of managing a story in an interactive application featuring a large virtual world inhabited by intelligent virtual humans. Above all, we think of a role-playing computer game underpinned by a narrative plot. A framework for driving virtual humans is already developed, as well as a prototype application for story management.

Several aspects need to be stressed. First, by *virtual human* (v-human in what follows) we mean a piece of software that *imitates* behaviour of a human in a virtual world and that is equipped with a virtual body. Even if the body is typically visualised by a graphical viewer, we will not consider graphical issues here. We are focused on *intelligent* v-humans, by which we mean that they carry out more complicated tasks than just walking, object grasping or chatting in an ELIZA-like manner.

Second, by *large virtual world* (v-world) we mean a really *large* artificial environment – not a single room, but a village or a region. There are tens or hundreds of v-humans acting in a large v-world. The story should last for tens of minutes.

Third, by *interactive* we mean that there is at least one user interacting with the simulation. Obviously, this will cause an interactive–narrative tension. The user can be embodied through an avatar, or can act within the v-world in an alternative way.

Fourth, a story is pre-given by a *plot specification.* The plot is not linear. It means that the story can evolve in several different ways according to a user intervention, and it can take place in several different locations at the same moment – not only in the location observed by the user.

We aim at developing a large long-lasting interactive computer game emphasising a story and intelligent artificial actors. From the artificial intelligence point of view, anyone with this goal must tackle at least the following issues:

1) How to control behaviour of an individual v-human?
2) How to simulate efficiently a large v-world when it is not possible to simulate it in its entirety due to enormous computational cost?
3) How to manage an interactive simulation according to a plot specification?
4) How to merge independent solutions of 1 – 3 into a single application?

We developed a toolkit for prototyping behaviour of individual v-humans (Bojar *et al.*, 2005) and, based on its concepts, a simulator of large v-worlds called IVE (*intelligent virtual environment*, Fig. 1) (Brom *et al.*, 2006). IVE presents an augmentation of Bryson's hierarchical reactive planning (2001), BDI theory (1987), and Gibson's theory of affordances (1979). These three main points help us to achieve several things. First, the creation of v-humans with relatively complicated behaviour is possible. Second, a level-of detail (LOD) technique for "AI" of v-humans and topology of virtual world (not for computer graphics) can be used; this allows for an automatic smooth simulation simplification on places unimportant at a given instant. Third, new objects and locations can be added into v-worlds as plug-ins and v-humans are able to adapt to them without using any machine-learning algorithm (similarly to The Sims). This feature facilitates a design.

IVE has been tested with a scenario comprising about 100 v-humans acting in four virtual villages; each with a pub, 5 mines, and 12 houses. The results

Presented at AISB 06, at "Narrative AI and Games" workshop
Published in the proceedings of AISB printed by the Society for the Study of Artificial Intelligence and the Simulation of Behaviour

have allowed us to conclude that the issues 1 and 2 have been solved. However, as the v-humans of IVE are driven only by prescripted reactive plans and schedules, no interesting story can emerge. Thus, the logical next step has been to address the issues 3, 4. In other words, our recent goal is to augment IVE with a story management module driving the simulation according to a high-level plot specification.

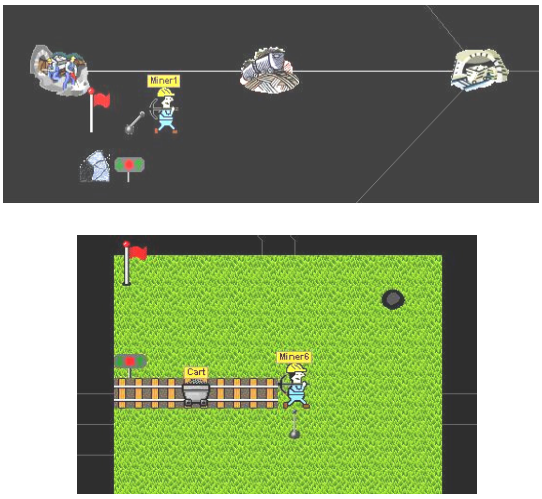


Figure 1: Mine scenario in IVE. The mine comprises three sublocations: the lower part, the tunnel and the upper part. There are two miners – one in the upper and one in the lower part. The lower part and the tunnel are not expanded (above), contrary to the upper part (bellow). The scenario illustrates an activity taking place in locations with different LOD value (*i.e.*, mining). The cart leaves the pit, which is simulated in less detail (LOD is 4), and enters the upper part, which is simulated in more detail (LOD 5).

Solving the issue 3 we have developed a technique for authoring the plots and for managing the stories, which uses Petri Nets. We have not fully augmented IVE with the technique yet, hence the issue 4 remains on the list. However, a test application (called TEST) for prototyping the plots and verifying the technique is already developed. In this paper, we describe the technique. We start with setting the ground of related works and detailing the requirements on the technique in Section 2. Section 3 introduces our technique, and describes TEST and a case-study story prototyped in TEST. In Section 4, a formal description of our method is given. Finally, we evaluate the technique and discuss directions for future research.

2 Related work and problem detailed

This section details requirements we had on the technique and discusses related work. With respect to our final goal, we were seeking a technique that:

a) copes with large v-worlds,
b) allows for describing high-level plot specifications, and yet allows for user interaction and for some degree of autonomy of v-humans,
c) copes with stories unfolding in several different places at the same instant,
d) allows for prototyping of stories.

The requirement b) means that we wanted a story manager to alter behaviour of individual v-humans, but not to drive them step by step. At the level of individual actors, we wanted the autonomy to be retained. Thus, our approach can be seen as a compromise between top-down scripting and emergent narrative. The requirement d) means that the method should allow a user to verify a plot in a test application that just *models* the course of the story. There should be no v-human and no 3D graphics in the application; it should serve just as a tool for testing the plot.

To be clear, we were not interested in automatic story generation, so the results of Cavazza (2002), who, generally saying, used an HTN planner for generating dramatically interesting sitcom-like episodes, is not what we were looking for. Similarly, we could not follow the approach of Aylett *et al.* (2005). In a nutshell, they used a continuous partially-ordered planner for generating episodes according to a high-level plot in an anti-bullying educational application. Notice, that we do not disregard the role of planners in storytelling applications. However, we think that it is hard to use them for large virtual worlds (issue a)) because of their exponential complexity.

We were interested, similarly to Mateas (2001), in authoring of plots and in keeping an unfolding story as close to an optimal plot as possible. Unfortunately, we could not use his solution, because it was not clear how to scale it to deal with the issues a), b), and d). Neither could we use the method of Szilas *et al.* (2003), because we did not find out how it could cope with issue a), b).

There is a branch of techniques used for specifying plots that exploits finite-state machines (FSM). Each state is a story episode, and a transition is a trigger that detects an end of the episode and starts a next one. FSM were described for example in (Sheldon, 2004) and used by Silva *et al.* (2003). Natural advantage of FSM is that they are formal, and yet graphical (Fig. 2), which facilitates a story design. However, a classical FSM is not suitable for us, since they cannot cope with issues c) and d). First, a (deterministic) FSM has just one active state at a given time, second, its triggers test for v-world events, which actually can not occur in a prototyping application for there is no v-world in it. Instead, we needed something like a FSM with more active states (to tackle c)) and with triggers that would

have tested for events caused by the plot specification itself (not occurrences in a v-world – to tackle d). We found that this "something like a FSM" is a Petri Net (PN). They have been already used to describe story plots in (Natkin and Vega, 2003), however, only to a retrospective analyse of a story (of a computer game). We are interested in the reverse issue.

To recapitulate, there were 4 main requirements we had on the plot description technique. We found out that we could not use state-of-art work, especially FSM. We have realised that a PN similar to the PN used in (Natkin and Vega, 2003) is the solution. The method is introduced in Section 3 and detailed in Section 4.

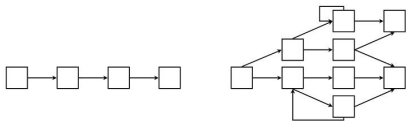


Figure 2: Story plots as finite-state machines. The linear plot is on the left, the nonlinear on the right.

3 PN-model and Story example

In this section, we introduce our technique, describe TEST and present an example of a story-plot prototyped in TEST. We recommend the reader to be familiar with PN – see (Natkin and Vega, 2003) for an introduction.

3.1 PN-model

To address the issues a) – c) (Sec. 2) we have refined a kind of PN, so called *timed coloured PN*, to serve for describing high-level plot specifications. To set the terminology, we say that *PN-model* is our type of PN, and *PN-plot* is a specification of an individual plot by means of the PN-model.

To address the issue d), we have developed two kind of PN-models, so called *draft PN-model* and *final PN-model*. The former serves for specification of a *draft PN-plot* and the latter for specification of a *final PN-plot*. Every draft can be verified by TEST, but can not be used for a real story manager. Every trigger of a draft PN-plot tests only for events caused by the draft itself, but not for any v-world events. Drafts serve for verifying the plots. Additionally, we have developed a method for converting a draft PN-plot to a final PN-plot. This conversion is to be carried out by the story designer after the draft is verified. Some triggers of the final PN-plot tests also for the occurrences of the v-world. The final PN-plot is aimed for a real story manager.

To summarise, our methodology for a specification of a plot is following: First, design a draft PN-plot. Second, verify the draft in TEST and adjust it. Third, convert the adjusted draft to a final PN-plot. In fact, this method is an example of iterative design. Section 4 details draft PN-model. Section 5 describes the conversion method.

3.2 TEST

We have developed a test application TEST, where a story is unfolded in an abstract manner according to a given PN-plot. TEST works as follows: First, a draft PN-plot is loaded from an XML-file. Second, when the simulation is started, abstract events start to occur according to the PN-plot. Third, a user can alter the story at real-time.

The purpose of the application is twofold. It serves as a "proof-of-concept" of our PN-model, and it allows for prototyping and verifying stories, including modelling of a user interaction. Actually, we are working on a real story manager of IVE.

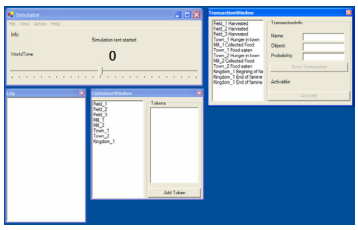


Figure 3. A screenshot of TEST.

3.3 Story example

Here, we present an example of a story we have prototyped in TEST. Notice, that the story has no artistic value, it serves only for a demonstration purpose. The story is a simple fantasy narrative set in a village in an evening. It is a rather small story, just an episode from a larger tale. There are the following actors and groups of actors:

- MAGICIAN: a user-actor. She needs a puppet (for whatever reason).
- GUARDS: four bum-bailiffs. One of them is a friend of the magician; he has given her a note that there will be a puppet theatre coming this evening.
- PUPPET THEATRE: four artists. One of them is a twin of a guy who used to steal in the village several years ago, was arrested but managed to escape.
- ROBBERS: a band having pilfered in the village for a few weeks.
- VILLAGERS: citizens of the village. Some of them will mistakenly recognise the twin as his robber-brother.

Two things are going to happen in parallel. First, the robbers are going to pilfer this evening and the guards will try to capture them. Second, the troupe is going to perform a piece in a pub, and in the

course of the play, some villagers will mistake the twin as his robber-brother and a brawl will flare up in the pub. Consequently, a fire might start, that would destroy the theatre and all the puppets.

Nothing is fixed, the events occur in a probabilistic manner. It is not sure whether or not the guards capture the robbers, or even notice them; whether or not somebody mistakes the twin; and whether or not the fire breaks out.

The magician is allowed to influence the story in the following ways: She can buy a puppet (before the theatre is reduced to ashes), or steal one in the course of the brawl. If she helps the artists in the brawl, she will be given a puppet for free, provided that no fire has broken out.

She can call the guards during the brawl using a spell. In this case, the guards interrupt the chasing and come to the pub. This reduces the probability of the fire breaking out significantly, especially if guards have just caught the robbers. She can also put out the fire immediately by casting a spell.

A part of the PN-plot for this story is depicted in Figure 4. The most important features of the portrayal are described in Section 4. The complexity of the drawing is discussed in Section 6.

4 Petri net model

This section gives the formal description of the draft PN-model and describes the most important features of the graphical representation of the PN-plot depicted in Figure 4. Notice however, that portrayals are informative only; what is important is a background formal specification. An algorithm for driving a story according to a PN-plot (both draft and final) is also presented.

A typical PN consists of places, tokens, transitions and transition function. In our model, we talk about containers (which are places – "the circles": ○, ◎, ⬡, ⬢), tokens (which are "the pellets": ●), actions (which are transition – "the rectangles": ▭) and triggers (more or less, they correspond to a transition function – to "the arrows": ⟶, ⋯⟶). The purpose of a trigger is to fire an action according to tokens' location in containers, or to add or to remove tokens from containers. Hence, tokens' locations evolve in time.

Tokens. Every *token* has a name, a colour, age, and a state. For every colour, there is a set of corresponding states. Let us denote T_{ALL} a set of all possible tokens, N_{ALL} a set of all possible names, B_{ALL} a set of all possible colours, and S_b a set of all possible states for a colour $b \in B_{ALL}$.

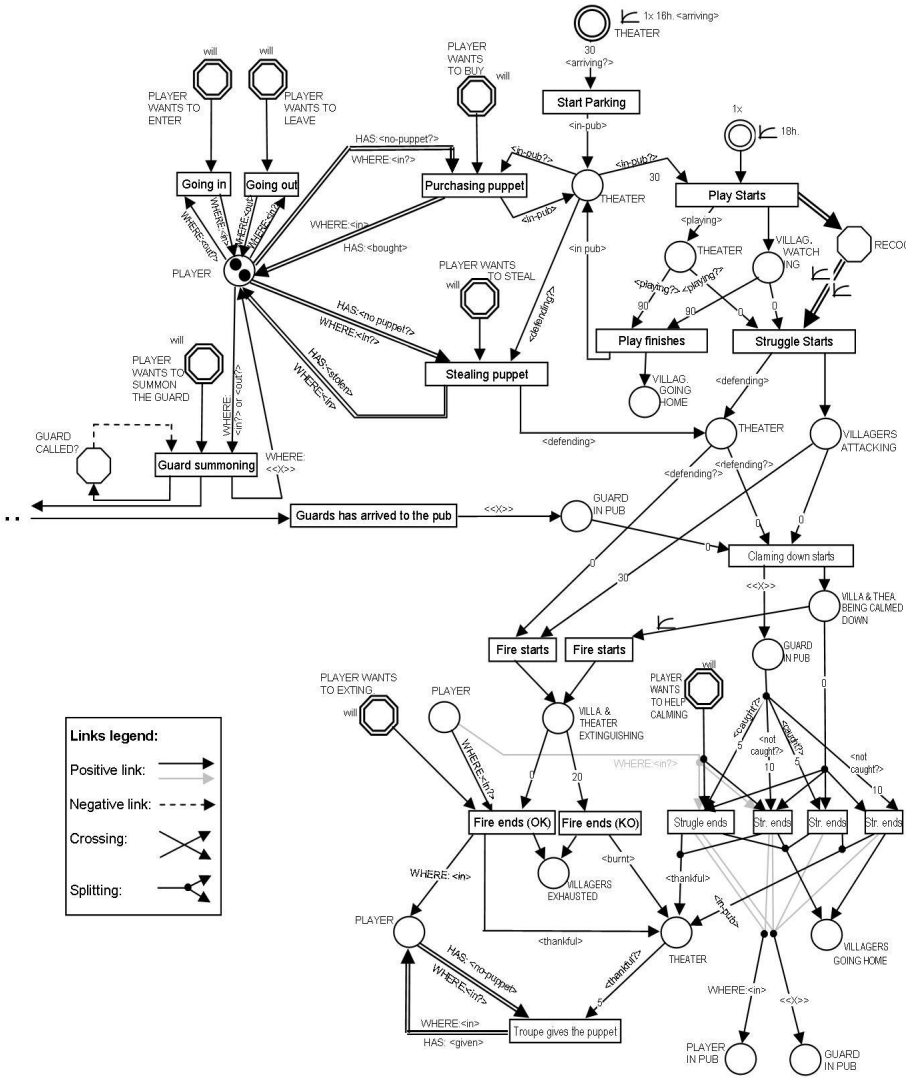


Figure 4: The "pub-plot" in the initial state is depicted. Notice that a plot can be much larger and several PN can run in parallel. In particular, in our case, the "catching-plot" is connected to the "pub-plot" on the left (indicated by "...").

Presented at AISB 06, at "Narrative AI and Games" workshop
Published in the proceedings of AISB printed by the Society for the Study of Artificial Intelligence and the Simulation of Behaviour

Then, we say that *colouring* is a function: β: $T_{ALL} \rightarrow B_{ALL}$ – it gives a token's colour – and *token–status* is a function (defined for each colour): σ_b: $T_{ALL} \rightarrow N_{ALL} \times S_b \times \mathbf{N}$ – it gives a token's name, state, and age. If an S_b is empty, we say that the token is stateless. A token can be located in a container. The age stands for how long the token is located there. As Fig. 4 depicts the PN-plot before the simulation is started, only the initial tokens are shown. After the start, other tokens would begin to appear or could be removed.

Containers. Every *container* has a name, a type and can be associated with a set of triggers, which are actually if-then rules. Let us denote C_{ALL} a set of all possible containers, Y_{ALL} a set of all possible types and I_{ALL} a set of all possible triggers. Then, we say that *container–status* is a function γ: $C_{ALL} \rightarrow N_{ALL} \times Y_{ALL} \times \wp(I_{ALL})$ – it gives a container's name, type and triggers. A container can contain more than one token in a given simulation time. We say that *containing* is a function κ: $C_{ALL} \rightarrow \wp(T_{ALL})$ – it provides all tokens located in a given container in a given instant.

A semantic meaning of a token in a container is a denotation either of a state of a group of actors (*i.e.*, an actor-token) or of a satisfied precondition (*i.e.*, a prec-token). In a drawing, containers are denoted according to their triggers and tokens they can contain as ○ (actor-tokens/without any trigger), ◎ (actor-tokens/with triggers), ⬡ (prec-tokens/without any trigger), ⬢ (prec-tokens/with triggers).

There are following main token types in our PN-plot; *italic* denotes the initial state:

- "magician has" (colour *m*); S_m = {*no-puppet*, bought, stolen, given}
- "magician where" (colour *i*); S_i = {*in-pub*, out-pub}
- "guard" (colour *g*); S_g = {*not-caught*, caught}
- "theatre troupe" (colour *t*); S_t = {arriving, in-pub, playing, defending, burnt, thankful}
- "villagers", "robbers" (colour *o*); S_o = {}
- a prec-token: "has-called", "has-recognised" (colour *o*)

The meaning is obvious: for example, every "magician has" token represents that a magician has stolen, or has bought, or has been given a puppet, or does not have it, respectively. Similarly, the state of "magician where" stands for a magician in a pub or out of the pub. Notice, that these tokens can be located only in PLAYER container. Notice also, that a token is not a v-human itself, it is just a representation of its state for the purpose of a story management!

Triggers. The most important primitive of the PN-model is a *trigger*. A trigger can be associated with an action (*action trigger*) or a container (*container trigger*). Basically, a trigger is an *if-p-then-c* rule, where *p* is a precondition and *c* a consequence, which is to be performed when *p* holds. There are four types of triggers (both action and container).

- A *token-generating* trigger is a trigger that has a consequence of always adding some tokens to some containers and not removing any token.
- A *token-consuming* trigger has a consequence of always removing at least one token and possibly adding tokens to containers.
- An *action-firing* trigger neither generates nor removes any tokens, but fires an action.
- A *conflict-resolving* trigger's precondition tests whether two or more actions are to be fired at the same time, and its consequence resolves the conflict.

Actions and triggers. Every *action* has a name, an action-firing trigger, a token-generating trigger and token-consuming trigger, a "ready to fire" flag, and an effect. The precondition of the last two triggers tests whether the action has just fired (*i.e.*, whether or not the flag is set). Let us denote A_{ALL} a set of all possible actions. Then, we say that *action–status* is a function α: $A_{ALL} \rightarrow N_{ALL} \times \{0,1\} \times I_{ALL}^{3}$ – it returns an action's name, whether or not its flag is set, and its triggers.

If an action-firing trigger holds, its consequence sets a "ready to fire" flag. Then, the other two triggers can be triggered. Notice the word "can". In a given instant, more actions can be ready to fire, but not all of them can be allowed to fire for there can be a conflict between their token-consuming triggers (it is not possible to remove a token that has been just removed by another trigger). A conflict can be solved by a conflict-resolving trigger, which unsets the flag. Finally, when an action is really allowed to fire, its effect is performed, and its token-generating and token-consuming triggers are triggered. See Algorithm 1 below for details.

There is no graphical primitive corresponding to a trigger. However, a container with a trigger is denoted as ◎ or ⬢, and an action trigger is depicted as a set of arrows, each from a container to an action, or vice versa. For example, ⊙→▭ denotes an action-firing trigger and a token-consuming trigger, which means "try to fire the action if there is at least one token in the container" and "consume one token when the action is fired". An example of a token generating trigger is ▭→○ (it means "generate one token when the action is fired").

Details of a precondition or a consequence are indicated schematically next to an arrow, next to a container, or by a changed shape of the arrow. Precisely, a precondition can:

Presented at AISB 06, at "Narrative AI and Games" workshop
Published in the proceedings of AISB printed by the Society for the Study of Artificial Intelligence and the Simulation of Behaviour

Algorithm 1. PN-plot driving in TEST and in a real drama manager.
Input: PN-plot.
The algorithm is performed in every time step *t*.
1) $I_{action,t} \leftarrow$ all action-firing triggers that fire in time *t*
2) Perform a consequence for all $i \in I_{action,t}$ in a random order (*i.e.*, mark the "ready to fire" flag of the respective actions)
3) $I_{conflict,t} \leftarrow$ all conflict-resolving triggers that fire in time *t*
4) Perform a consequence for all $i \in I_{conflict,t}$ in a random order (it unsets the "ready to fire" flag of some conflicting actions; it may include asking a user for selecting the action)
5) $I_{remove,t} \leftarrow$ all token-consuming triggers that fire in time *t* (they test for "ready to fire")
6) **for each** $i \in I_{remove,t}$ (take *i* in a random order) **do {**
   if all desired tokens can be still removed **do {**
      perform the consequence of *i* (it includes removing the tokens)
      if *i* is an action trigger **do** perform an effect of the action **}**
   otherwise,
      if *i* is an action trigger **do** unset the "ready to fire" flag **}**
7) $I_{generate,t} \leftarrow$ all token generating triggers that fire in time *t*
8) **for each** $i \in I_{generate,t}$ (take *i* in a random order) **do {**
   perform the consequence of *i*
   if *i* is an action trigger **do** unset "ready to fire" flag **}**

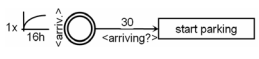
- test κ: *i.e.*, whether some containers contain (⟶) or do not contain (⋯⋯→) some tokens;
- test β and σ_β: *i.e.*, colouring (<col?>) and token-status (<state?>),
- test simulation time (HH hours),
- compare age (MM minutes) or simulation time (HH hours) to a random value generated by a given probabilistic distribution (◺),
- test whether a user will alter the simulation somehow (will),
- test whether a trigger has fired *n* times (n ×),
- test whether two or more actions are in a conflict (======).

A consequence of a token-generating trigger can generate a token of a specific colour and a state. It is indicated as <state>, <<X>> or <col> above the arrow. <<X>> means that the generated token has the same state as the token that has been just consumed (see, for example, GUARD IN PUB container).

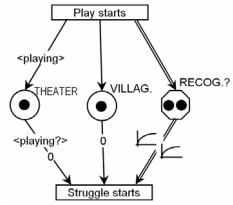
Notice that all abovementioned acts relate to the draft PN-plot itself, no v-world events are tested. This is not possible with any FSM. As TEST only models the course of a story, the effects of all actions are user notifications of an action performance.

Notice that not all conflicts are always solved in Step 4 – there might exist a conflict not recognised by any trigger. In this case, the action to execute will be chosen randomly from the set of the actions in the conflict (Step 6).

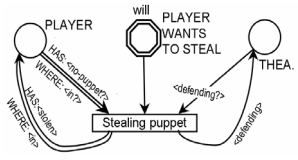
Now, consider several examples taken from Figure 4. First, focus on the container THEATRE and the action *Start parking* (in the pub). There are three triggers indicated in the drawing. The first one generates a token "theatre" with arriving state into the container once at about 4 p.m. (Step 7, 8). The second one is an action-firing trigger. It sets "ready to fire" flag when the token in the container is at least 30 minutes old (Step 1, 2). The third one is a token-consuming trigger, which removes "theatre" token when the action fires (Step 5, 6).



The next example concerns with the episode of the theatrical performance. The token-generating trigger of the *Play starts* action has generated four tokens; two "has-recognised" into the RECOGNISED? container, one "villagers" into the VILLAGERS WATCHING container, and one "theatre" with the state playing into the THEATRE container. The action-firing trigger of the *Struggle starts* action tests whether or not there is a token with the state playing in the container THEATRE, and a token in the container VILLAGERS, and two tokens in the container RECOGNISED? with appropriate age. It is determined randomly for the both tokens, whether or not the token is old enough. If the action is fired, the next trigger – the token-consuming one of *Struggle starts* – consumes all four tokens. Notice that all durative episodes are represented in a similar way using the age of the tokens. The container THEATRE is the same as in the previous figure, it is just depicted several times for convenience.



The next drawing represents the stealing of a puppet. If the magician is in the pub (denoted as WHERE: <in?>), if she does not have a puppet (denoted as HAS: <no-puppet?>), and if she wants to steal, and if the brawl has already flared up (the state of "theater" is defending), she may steal the puppet. Notice that a token "player wants to steal" is generated according to a choice of a user – this feature is achieved by checking a check box in TEST now. From the perspective of the high-level plot, the action of stealing is instantaneous: it takes no time.



Notice that there is no conflict-resolving trigger in PN-plot of this story. Notice also that all the properties that are only indicated in the portrayal, such as probabilistic distributions, are specified precisely in the XML plot specification.

5 Supplementing IVE with a story manager

In this section, we describe how to convert a draft PN-plot to final PN-plot. We first remember IVE. Then we extend the draft PN-model specified in Section 3 to the final PN-model. Finally, we describe how to convert a draft to a final PN-plot.

Presented at AISB 06, at "Narrative AI and Games" workshop
Published in the proceedings of AISB printed by the Society for the Study of Artificial Intelligence and the Simulation of Behaviour

In IVE, almost all behaviour of v-humans (say, 90%) is driven by hierarchical reactive planning, in particular by production rules organised in tree-like structures (Brom *et al.*, 2006)[1]. There are two reasons for that: the reactive planning fits well in a large, unpredictable dynamic v-world, and hierarchical nature of plans helps us to achieve LOD simulation. A classical planning technique needs to be exploited only rarely (a case in point is a path-finding).

In order to extend the draft PN-model to the final PN-model, we need to specify how a story manager can alter a v-world and how a v-world can modify a final PN-plot. The extension is founded accordingly. First, the story manager can alter the story by performing an effect of a PN-action. In the draft, the effect is just a user notification. In contrast, in a final PN-model, an effect is allowed to:

1. create a new virtual object, a location or a v-human,
2. force a v-human upon a new reactive plan, hence modify its behaviour,
3. modify an internal drive of a v-human, hence change its needs,
4. increase or decrease LOD in a given part of a v-world.

For example, in Fig. 4, the effect of the action *Struggle starts* would be the case 2. Second, we allow preconditions of triggers in the final PN-model to test also circumstances of v-world. For example, each trigger of "will" would correspond to an event caused by a user actor.

Having the final PN-model, we can describe how to convert a draft PN-plot to a final PN-plot. Suppose we have a draft PN-plot of a story episode.

1. Replace every precondition of a trigger of "will" from the draft by a final precondition that tests whether or not a user has just caused a respective event.
2. Replace every effect of an action beginning the story from the draft by an effect from the list of possible final effects (above).
3. Test in the final application, whether the story episode begins as expected.
4. From the draft, take all action-firing triggers, conflict resolving triggers, token-generating triggers and actions not beginning the story. Take them one by one in the order of their presupposed time of firing: if it is a trigger, ponder on whether its precondition should be replaced; if it is an action, ponder on whether its effect should be replaced. If it should, replace it.
5. Test the final application.

Since we do not have a final story manager yet, we have converted some of our draft PN-plots verified in TEST (including the example from Sec. 3) to the final PN-plots by hand. Notice that this procedure is intended to be automated partially.

We have realised several things. Preconditions testing an age of a token are usually replaced by a test of occurrence of an event in a v-world. Typically, preconditions of token-generating triggers of containers are replaced. Token-generating and token-consuming triggers of actions need to be modified only rarely. In the example from Sec. 3, about 25% of preconditions from the draft were replaced.

6 Discussion and Future work

Although our work is in progress, the results we have collected so far show that our model allow us to achieve the four requirements from Sec. 2. Notice that for a story comprising several episodes, more PN-plots can be specified and all of them can run in parallel. The magician, for example, does not have to be in the pub for the brawl to flare up. She can be involved in the "robbers catching" episode (that would decrease LOD value in the pub). Notice also that while the story manager alters the course of the overall simulation from a high-level perspective, the autonomy from the point of view of individual v-humans is retained. V-humans act according to their reactive plans, which are only "attuned" to the story. Our approach presents a compromise between pure scripting and pure emergent narrative. We have also realised that it is extremely useful to have an application for a story prototyping. During the creation of a plot one can make a flaw. In TEST, the flaws can be detected easily. There are also several natural advantages of PN. They can be formally validated (which helps to detect unreachable parts for example), depicted graphically and compiled at runtime.

There are also several drawbacks of our approach. First, our tokens are state-based. That means a token can have exactly one state: *e.g.*, "magician" token can have stolen state, but not both stolen and, for example, in-pub state – we need two tokens. Tokens also can not hold variables. These are merely technical limitations. We did not need these features for our plots and therefore we did not specify them in the formal model. In principle, it is possible to formalise a PN-model so that the features are included.

A more inherent problem is that a portrayal of even a small story (see Fig. 4) is fairly complicated. Assume that there are tens of such episodes; how

[1] Notice that there is a distinction between pure reactive agents employed e.g. by Brooks (1986), and v-humans driven by reactive planning. The latter can for example exhibit goal-directed behaviour and use their memory. A v-human driven by a reactive planner is not "intelligence without representation". Notice also, that a reactive plan, even though pre-given, is not a computer game script.

Presented at AISB 06, at "Narrative AI and Games" workshop
Published in the proceedings of AISB printed by the Society for the Study of Artificial Intelligence and the Simulation of Behaviour

could a story designer cope? We think that one possible approach to this problem is to extend the PN-model in a hierarchical manner. Microsoft Visio (2006) might be a tool for drawing of plots in this way. The advantage of Visio is that one can define in it new graphical components and routines for their conversion to XML specifications. These specifications are needed both for TEST and a real story manager.

Our future work concerns development of a real story manager and addressing the problem with large portrayals using a neat drawing tool. We remark that Algorithm 1 is designed to drive both TEST and a real story manager.

7 Conclusion

In this paper, we have described a method for specifying non-linear plots for a large computer game featuring intelligent virtual humans. The technique exploits Petri Nets specifications. It is designed to cope with four issues: with large virtual worlds, with stories unfolding in several different places at the same time, with allowing for some degree of autonomy of v-humans and with allowing for prototyping of stories. The natural feature of the technique is that the plots can be depicted graphically.

We have briefly described our application called TEST that serves to verify plots. We have also introduced IVE, which is our AI framework for programming large virtual worlds inhabited with tens of actors. Our future work concern supplementing IVE with a story manager that uses our Petri Nets and that is driven by the algorithm described in this paper.

Although the technique is aimed for IVE primarily, it can be used in another application as well, in particular in an application featuring a large v-world. TEST can be used independently as it is. Both TEST and IVE can be downloaded at http://urtax.ms.mff.cuni.cz/ive/.

Acknowledgements

This work is partially supported by the Program "Information Society", the project 1ET10030051; and GA UK No. 351/2006. The authors would like to thank to Joanna Bryson, Lubomír Bulej and two anonymous referees for their valuable comments.

References

Ruth S. Aylett, S. Louchart, J. Dias, A. Paiva, M. Vala. FearNot! – An Experiment in Emergent Narrative. *Proceedings of Intelligent Virtual Agents*, 305–316, 2005.

Ondřej Bojar, Cyril Brom, Milan Hladík, Vojtěch Toman. The Project ENTs: Towards Modeling Human-like Artificial Agents. *SOFSEM 2005 Communications*, Slovak Republic, 2005.

Michael E. Bratman. *Intention, plans, and practical reason*. Cambridge, Mass: Harvard University Press, 1987.

Rodney A. Brooks. Intelligence without reason. *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney, 569–595, 1991.

Cyril Brom, Jiří Lukavský, Ondřej Šerý, Tomáš Poch, Pavel Šafrata. Affordances and level-of-detail AI for virtual humans. *Proceedings of Game Set and Match 2*, Delft, 2006 (to appear)

Joanna Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents.* PhD thesis, Massachusetts Institute of Technology, 2001.

Marc Cavazza, Fred Charles, and S. J. Mead. Interacting with Virtual Characters in Interactive Storytelling. *ACM Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 318 – 325, 2002.

James J. Gibson. *The Ecological Approach to Visual Perception.* Boston: Houghton Muffin, 1979.

Michael Mateas. *Interactive Drama, Art and Artificial Intelligence.* Ph.D. Dissertation. Department of Computer Science, Carnegie Mellon University, 2002.

Microsoft Corporation. Microsoft Office Visio 2003 *www.microsoft.com/office/visio/* [14th January 2006]

Stéphane Natkin and Liliana Vega. Petri Net Modelling for the Analysis of the Ordering of Actions in Computer Games. *Proceedings of Game-ON*, 82–92, 2003.

Lee Sheldon. *Character Development and Storytelling.* Chapters 7 and 14. Thompson Course Technology, 2004.

André Silva, Guilherme Raimundo, and Ana Paiva. Tell Me That Bit Again... Bringing Interactivity to a Virtual Storyteller. *Proceedings of Virtual Storytelling II*, 146–155, 2003.

Nicolas Szilas, Olivier Marty, Jean-Huges Réty. Authoring highly generative Interactive Drama. *Proceedings of 2nd International Conference on Virtual Storytelling*, 2003