A Methodology to Derive a Valid Scenario of an Interactive Storytelling

Kim Dung Dang, Ronan Champagnat, Michel Augeraud

A Methodology to Derive a Valid Scenario of an Interactive Storytelling

Kim Dung Dang
University of La Rochelle - L3i
Avenue Michel Crépeau
17042 La Rochelle, France
(+33) (0)5 46 45 82 62
kim_dung.dang@univ-lr.fr

Ronan Champagnat
University of La Rochelle - L3i
Avenue Michel Crépeau
17042 La Rochelle, France
(+33) (0)5 46 45 82 62
ronan.champagnat@univ-lr.fr

Michel Augeraud
University of La Rochelle - L3i
Avenue Michel Crépeau
17042 La Rochelle, France
(+33) (0)5 46 45 82 62
michel.augeraud@univ-lr.fr

ABSTRACT
In previous works [3, 4], we showed how to use Linear Logic to model an Interactive Storytelling (IS). Proceeding from the achieved results, this paper introduces a methodology for authors to derive a valid scenario of an IS. In the paper, we will explain the implementation of the methodology via a detailed presentation of the steps in the process of IS modeling and illustrate those with a concrete example.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *computer-aided software engineering (CASE), user interfaces, state diagrams.*

General Terms
Management, Design, Experimentation, Human Factors, Verification.

Keywords
Game, IS, modeling, Linear Logic, model derivation, validation.

1. INTRODUCTION
Researches on IS, in general, are divided into two major families: scenario-driven approach (discourse point of view) and emergent narrative theory (character point of view) [6]. The first set of families [10, 11, 14, 9] aims to guarantee that the story development is coherent and leads to author's desired effects. And hence when a player's action deviates from the pre-computed story plan, the system either replans (gets the story back on track), or makes the player's action have no effect on the story progress. As a consequence, the player cannot direct the story unfolding in a considerable way. On the opposite, the emergent narrative theory [1, 2, 13, 12] gives complete freedom to the player, who may deeply influence, through her/his actions, the evolution of the virtual world where s/he has been immersed. This means that the story will emerge from the player's interaction with the game, and the unfolding of the story is not based on any specific structure. However, its foremost limit is the deriving quality, in terms of consistency and pertinence, which highly depends on the player and therefore cannot be guaranteed.

In previous works [3, 4], we showed how to use Linear Logic to model an IS thanks to which the strong points of both the discourse point of view and the character point of view are combined. In addition, in order to apply this approach to creating interactive video games, we have developed a system assuring a set of objectives [5]: the player does not feel constrained by the game but s/he can determine its evolution; the virtual world must provide a coherent environment that is appropriate for player's actions; the progress of the game has to respect a structure of discourse (a common structure of a discourse is made of introduction; stating problems; solving them step by step; conclusion) which has been pre-defined by an expert of the domain (author/designer). To this purpose, its architecture is composed of three components: Linear Logic model, IS controller and IS rendering (see Figure 1). The IS rendering builds in advance all the necessary interfaces (scenes) as well as directs the "rendering process" of the game (which interacts immediately with the player). The IS controller aims to manage the unfolding of the game and to ask the IS rendering to show suitable interfaces (scenes) for the game at each step, by taking into account the propositions of the Linear Logic model and player's action choices (transmitted to the IS controller via the IS rendering). The Linear Logic model stores a sequent that models the situation of the game at each moment (it is updated after each step). The automatic reasoning of the sequent (the automatic sequent proof) assists directly the IS controller in managing the game unfolding.

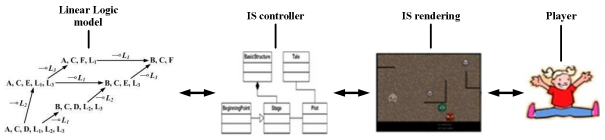

Figure 1. Architecture of the system.

The works described in this paper present a methodology for authors to create and validate the Linear Logic sequent mentioned above at the initial moment (*i.e.* it models the initial situation of the game, determines its scenario and is the input of the *Linear Logic model* component). In other words, the goal of the paper is to propose a methodology that allows authors to derive a valid scenario of an IS. Indeed, we have developed a tool which is an implementation of the foregoing methodology by a model driven approach to help authors derive a valid scenario even when they do not have any knowledge of Linear Logic. It is composed of two components: Editor of scenario and Analysis module.

- The editor of scenario is a graphical editor using a scenario metamodel thanks to which authors can model an IS as simply as possible: give a set of states, of player's action choices and of outcomes, specify associations between these elements via a set of events/actions together with their preconditions and effects.
- Based on the result received from the editor of scenario, the analysis module builds automatically (1) a reduced schema of discourses which shows an overall view on the generated scenario and thereby helps authors validate it, and (2) a Linear Logic sequent which is the input of the *Linear Logic model* component via two model transformations.

The paper begins with a brief introduction to Linear Logic. Then we gives the methodology to derive a valid scenario of an IS regarding both the discourse point of view and the character point of view. After that, we explain the implementation of this methodology by a detailed presentation of the steps in the process of IS modeling and illustrate those with a concrete example (these are also the main contribution of the paper).

However, above all, we define some important notions that are used in our approach:

- A *story* (a *game*) is a set of entities, of events/actions and of constraints that solves a set of problems, describes an evolution concerning a set of characters and/or of objects. It consists in starting from an initial situation, then in solving the given set of problems in order to reach a final situation that corresponds with one of satisfactory endings of author's goals.
- A *discourse* is an ordered sequence of events/actions that is a possible unfolding of a story. Therefore a same story can generate various discourses. This consists in scheduling the events/actions corresponding with the story.
- A *scenario* is a set of all the possible discourses for a story. If we change anything in the story then we will receive a new scenario.

2. BRIEF INTRODUCTION TO LINEAR LOGIC

Linear Logic is an executable formal model that has been introduced by Girard [7] as a "closer logic genre" than Classical Logic, where the *Contraction* and *Weakening* rules are "forbidden". In addition, it also considers atoms and formulas as resources that are consumed and/or produced. As a result, in Linear Logic, two instances of an atom (or of a formula) are different from one instance. Unlike Classical Logic, Linear Logic is not applied to determine whether an assertion is true or not, but it is employed to represent the validity of how resources are used when proving an assertion. In other words, we are interested in writing the proof and in analyzing the choices made during this phase. Those make Linear Logic well suited to modeling systems with resource sharing, to controlling processes, as well as to automatically reasoning on the logic of discourse, in particular when it embeds concepts of high relevance to storytelling, such as causality.

In order to model an IS, within the framework of this paper, we do not employ all the features of Linear Logic, but just the following connectives:

- $\multimap$*: linear implication (imply)*, expresses the possibility of deduction. Example: "1$ $\multimap$ 1kg strawberries" means that we can give 1$ to buy 1kg strawberries.
- $\otimes$*: multiplicative conjunction (times)*, expresses a set of "synchronous" resources. Example: "1$ $\multimap$ 1kg strawberries $\otimes$ 1kg tomatoes" means that we can give 1$ to buy 1kg strawberries and 1kg tomatoes.
- *&: additive conjunction (with)*, expresses an external choice to the system (for instance coming from a player) if it is in the left part of the sequent. Example: "1$ $\multimap$ tea & 1$ $\multimap$ coffee" means that we (the player) can choose tea or coffee when we give 1$ to an automatic machine.
- $\oplus$*: additive disjunction (plus)*, expresses an internal choice to the system (for instance coming from an IS controller) if it is in the left part of the sequent. Example: "1$ $\multimap$ tea $\oplus$ 1$ $\multimap$ 1$" means that it is the automatic machine which decides it will give us tea or return to us 1$ depending on the availability of tea in the machine, if this formula is in the left part of the sequent. If the connective $\oplus$ is in the right part of the sequent, it is only used to connect distinct consequents. For example, if the right part of a sequent is "tea $\oplus$ coffee", this means that if the sequent is provable, then from the left part of the sequent, we can receive either tea or coffee.

A sequent is an expression $\Gamma \vdash \Delta$, where Γ and Δ are sequences of atoms and/or of formulas; $\vdash$ (turnstile) is used to separate its left (antecedents/available resources) and right (consequents/conclusions) parts. For example, "A $\otimes$ (A $\multimap$ B) $\vdash$ B" (or "A, A $\multimap$ B $\vdash$ B") means the possibility to produce a copy of "B" by consuming the available resources "A" and "A $\multimap$ B" (we can substitute the connective $\otimes$ between two atoms, between two formulas, or between an atom and a formula in the left part of a sequent by the comma "," to be briefer). From the left part of a sequent, we may lead to many valid conclusions/consequents, and at the same time, a proof (how to reach a conclusion/consequent) is not unique, meaning that there exist many ways of reaching a same conclusion/consequent. Proving a Linear Logic sequent consists in rewriting the sequent, by making a substitution of one of its formulas at each step until the left part is identical to one consequent in the right part of the sequent. Thus for a same sequent, there may be several successful proofs, as well as several unsuccessful ones. As a result, the proof strategy becomes crucial in using Linear Logic to reason on the logic of discourse and on the resource allocation mechanisms for a story.

3. LINEAR LOGIC MODEL TO IS

This section describes how to model an IS by means of Linear Logic. First we give a model to represent the IS, it has been inspired from the Greimas' analysis [8] where an event/action (which modifies the state of something) is expressed by an abstract formula, namely, the narrative program. We have based our model on (player and non-player) characters' states, states of the story, player's action choices and outcomes as well as associations between those elements thanks to a set of events/actions. This model has offered a metamodel to develop the foregoing editor of scenario, its details are as follows:

- Player and non-player characters are modeled by atoms. An atom corresponds to a state of a character considering a

certain point of view. Therefore a character at a moment can be modeled by various atoms which constitute the character's situation at that moment and are put in a state vector corresponding with the character. Thus, the size and the component of the state vector of a character may vary during the unfolding of the story. Similarly, the states of the story are also modeled as atoms. These atoms represent the discourse point of view (author's desired effects) in the modeling process. In the list of states (of the characters and of the story), we have to show which ones are available at the initial moment of the IS.

- Player's action choices are expressed by inputs. This means that the player decides her/his occurrence in the unfolding of the story by entering the inputs. These inputs are modeled as atoms and will become available after being entered into the story by the player.
- An outcome (goal/conclusion/consequent) of the story is an author's desired ending. It corresponds to a state, or a set of states.
- An event/action may modify the situation of a (or some) character(s) and/or the states of the story. In the model, we are only interested in when (under which conditions) an event/action takes place (*Precondition* of the event/action) and in the received result after the event/action is executed (*Effect* of the event/action).

Then we transform this model into a sequent by applying the notions of Linear Logic accordingly (the transformation will be described in detail in the following):

- The right part of the sequent only includes the outcomes of the story which are connected by the connective ⊕.
- The available states at the initial moment of the IS correspond to the available atoms in the left part of the sequent.
- A multiplicative conjunction formula expresses a set of "synchronous" elements (state, input, event/action). For example, the states that constitute an outcome are connected between them by the connective ⊗.
- An additive conjunction (disjunction) formula in the left part of the sequent represents choices of the player (the IS controller) in the progress of the story.
- An event/action is similar to the working of the connective ⊸, and so it is linked to a linear implication formula.

We can find that: (1) as a proof expresses the steps to reach a consequent of a sequent (may be successful or unsuccessful), it is equivalent to a discourse which is an ordered sequence of events/actions that is a possible unfolding of a story; (2) from a sequent, we are able to build all the ways of writing proofs, therefore it corresponds to a scenario which is a set of all the possible discourses for a story.

Thus, the created sequent determines a scenario of the IS and has to be validated before it is used as the input of the *Linear Logic model* component (see more Figure 1). To do this, we have given the concept of *reduced schema of discourses* (that is explained in the next section) which allows verifying whether or not the corresponding scenario leads to satisfactory endings of author's goals.

4. PROCESS OF IS MODELING

The previous section gives a methodology to derive a valid scenario of an IS. This section introduces the implementation of the methodology via a detailed explanation of the steps in the modeling process thanks to which, authors can produce a Linear Logic sequent that represents a valid scenario of the IS. In order to help authors do those even when they do not have any knowledge of Linear Logic, we have developed a tool that is an implementation of the foregoing methodology where Linear Logic is "implicit" for authors. More concretely, the objective of the tool is to facilitate and optimize the modeling process via a model driven approach so that authors do not have to manipulate directly Linear Logic. Figure 2 presents briefly the modeling process of an IS using this tool. In the beginning, the author employs the editor of scenario to describe the story by creating the diagrams/lists of states, inputs, outcomes and events/actions. From these lists, the analysis module executes automatically two tasks:

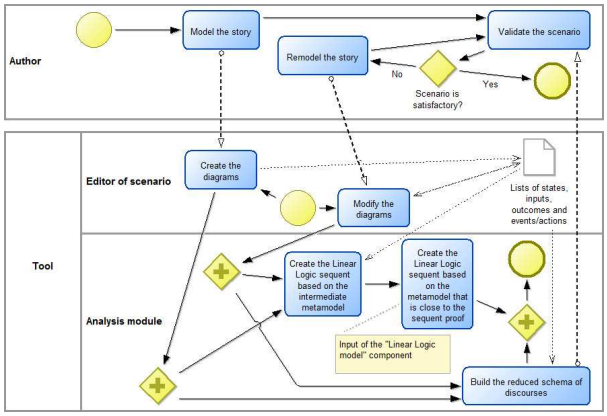

Figure 2. Modeling process of an IS (built according to the *BPMN – Business Process Modeling Notation* standard [16]).

- Firstly, it builds the reduced schema of discourses to help the author validate the generated scenario. Indeed, thanks to the reduced schema, s/he can verify if the scenario is satisfactory, if not, the author remodels the story in order to create a scenario that is the most appropriate for her/his desired goals (this process may be iterated if necessary).
- Secondly, the analysis module executes two model transformations to create in turn two Linear Logic sequents: the first represents the scenario corresponding with the produced lists of states, inputs, outcomes and events/actions (it is based on a metamodel that is intermediate to transform into the second); the second is the input of the *Linear Logic model* component (see more Figure 1), it assists the IS controller in managing the unfolding of the story via its automatic reasoning, as a result, it is based on a metamodel that is close to the sequent proof.

In the following, we will explain the implementation of the methodology via a detailed presentation of the steps in the process of IS modeling that has been briefly described above, as well as illustrate those with a concrete example. It is an extract of an educational game which warns of domestic electrical accidents (DEA game) whose objective consists in causing an electric shock for the player [4]. At first, the game designer (author) anticipates that the player, from an initial position, will go to the kitchen, where the IS controller will start the strategy of causing the electric shock for her/him, via appliances there such as a fridge, a

microwave oven, an electric cooker,… However, besides the possibility of going to the kitchen, the player may have other choices, for instance, staying at the initial position to work or going to the bathroom. What will happen?

4.1 Scenario Metamodel

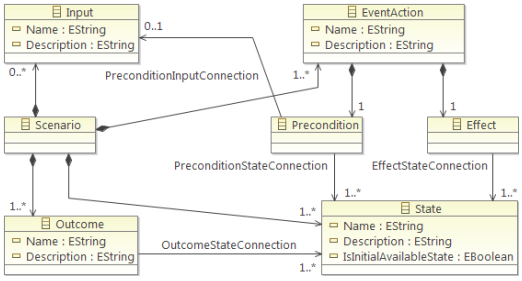

Figure 3. Scenario metamodel.

Figure 3 gives the metamodel which allows modeling the scenarios of an IS. Each instance of a class (*State*, *Input*, *Outcome, EventAction*) is distinguished by a *Name* (obligatory), it may be presented more concretely by a *Description* (not obligatory). A scenario is composed of four lists of states, inputs, outcomes and events/actions. The states have an attribute to show whether or not it is an initial available state of the story (IsInitalAvailableState = True/False, its default value is *False*). An *Outcome* is composed of one or some *State*(s). An *EventAction* includes a *Precondition* and an *Effect* which contain one or some *State*(s); if the *EventAction* is executed by a player's choice then its *Precondition* has to have an *Input*.

In order to facilitate the authors' work, we have built the editor of scenario as a graphical editor, thereby they can describe an IS by simple "pull and drop" manipulations. To do this, we have chosen GMF (Graphical Modeling Framework) [15] because it provides a generative component and runtime infrastructure for developing graphical editors. As a consequence, we have created a GMF project where the metamodel in Figure 3 is employed as a domain model (ecore model).

Now, let us see, as an example, how to describe the DEA game by means of this editor of scenario. Very simple – just build graphically (pull and drop accordingly the available elements in the editor) one after another four diagrams corresponding to four lists of states, inputs, outcomes and events/actions:

- States (of the game and of the player)

Name	IsInitial Available State	Description
Gi	True	Game's state: The game is at the initial state (this is an initial available state)
Gk	False	Game's state: The IS controller starts the strategy of causing the electric shock for the player in the kitchen
Gr	False	Game's state: The game reaches the goal (the player has got the electric shock)
Pi	True	Player's state: The player is at the initial state (this is an initial available state)
Pw	False	Player's state: The player works at the initial position
Pk	False	Player's state: The player is in the kitchen
Pb	False	Player's state: The player is in the bathroom
Pe	False	Player's state: The player has got the electric shock

We can find that the game has two initial available states: Gi and Pi.

- Inputs of the player (her/his action choices)

Name	Description
Iw	The player decides to work at the initial position
Ik	The player decides to go to the kitchen
Ib	The player decides to go to the bathroom

- Outcomes of the game: As the game's objective is to cause an electric shock for the player, it only has one outcome O including two states Pe and Gr, this means that the game's ending satisfies the author's desired goal if and only if the player's state is Pe and the game's state is Gr (endings with other states do not satisfy the author's desired goal).

Name	Description	
O	The player gets the electric shock	Pe, Gr

- Events/actions of the game

Name	Description	Pre condition	Effect
EA01	The player decides to work at the initial position by choosing Iw	Pi, Iw	Pw
EA02	The player decides to go from the initial position to the kitchen by choosing Ik	Pi, Ik	Pk
EA03	The player decides to go from the initial position to the bathroom by choosing Ib	Pi, Ib	Pb
EA04	The IS controller starts the strategy of causing the electric shock for the player in the kitchen	Pk, Gi	Pk, Gk
EA05	The player gets the electric shock	Pk, Gk	Pe, Gr

Let us consider the event/action EA01: *Precondition* contains one state Pi and one input Iw, *Effect* contains one state Pw. Therefore its meaning is: EA01 is executed if and only if the player is at the initial state (Pi) and s/he decides to work at the initial position by choosing Iw; the event/action's effect is that the player's state becomes Pw (s/he works at the initial position). The explanation for the other events/actions is similar.

After building graphically four diagrams corresponding to four lists of states, inputs, outcomes and events/actions thanks to the editor of scenario (Figure 4 is the diagram corresponding to the list of events/actions), we receive four XML files which express these lists (Figure 5 is the XML file representing the list of states).

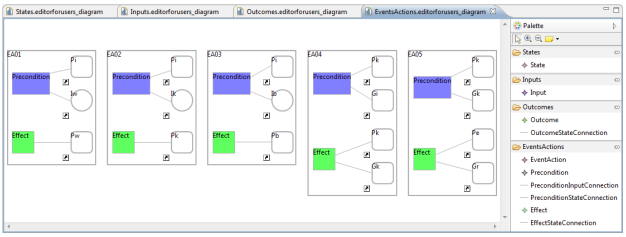

Figure 4. Diagram corresponding to the list of events/actions.

```
<Scenario>
  <State Name="Gi" IsInitialAvailableState="True" Description="Game's state: The game is at the initial state (this is an
  initial available state)"/>
  <State Name="Gk" IsInitialAvailableState="False" Description="Game's state: The IS controller starts the strategy of
  causing the electric shock for the player in the kitchen"/>
  <State Name="Gr" IsInitialAvailableState="False" Description="Game's state: The game reaches the goal (the player has
  got the electric shock)"/>
  <State Name="Pi" IsInitialAvailableState="True" Description="Player's state: The player is at the initial state (this is an
  initial available state)"/>
  <State Name="Pw" IsInitialAvailableState="False" Description="Player's state: The player works at the initial position"/>
  <State Name="Pk" IsInitialAvailableState="False" Description="Player's state: The player is in the kitchen"/>
  <State Name="Pb" IsInitialAvailableState="False" Description="Player's state: The player is in the bathroom"/>
  <State Name="Pe" IsInitialAvailableState="False" Description="Player's state: The player has got the electric shock"/>
</Scenario>
```

Figure 5. XML file representing the list of states.

4.2 Validation of Scenario Using the Reduced Schema of Discourses

The scenario validation problem, within the framework of this paper, is how to create a scenario that leads to satisfactory endings of author's goals (or how to guarantee that all the choices of the player or of the IS controller during the game unfolding lead to satisfactory endings of author's goals). To do this, after the author models the story by the editor of scenario, the analysis module builds automatically the reduced schema of discourses corresponding with the generated scenario, thanks to which the author can verify if it is satisfactory. If not, s/he remodels the story (eliminates and/or modifies the branches (paths) which do not direct toward successful endings) in order to receive the most pertinent scenario for her/his desired goals (this process may be iterated if necessary). The schema is called "reduced" because it does not contain symmetric discourses in the scenario (two discourses are symmetric if they are composed of a set of events/actions but the execution order of these events/actions is different, for instances, three discourses "EA01 → EA02 → EA03", "EA02 → EA01 → EA03" and "EA02 → EA03 → EA01" are symmetric). Each branch (path) of the reduced schema corresponds to one possibility of choice in the scenario (either the player's choice or the IS controller's choice). Thus we will receive all the paths (branches) leading to unsuccessful and/or successful endings of the goal of the story.

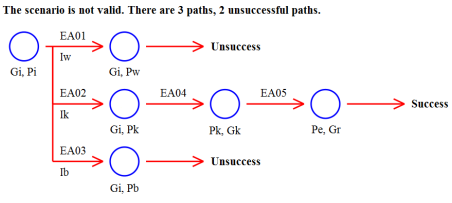

Figure 6. Reduced schema of discourses before the validation.

Let us return to the DEA game and see if the current scenario satisfies the game's goal (cause an electric shock for the player).

From four XML files representing four lists of states, inputs, outcomes and events/actions, we receive the reduced schema of discourses given in Figure 6 (it is automatically built by the analysis module) where: the first node including two states Gi and Pi corresponds with the initial situation of the game; each next node (with its states) corresponds with the effect of the execution of an event/action (an arc) which either needs or does not need an input.

We can see that there are two paths which lead to the unsatisfactory endings of the goal of the game (if the player decides to work at the initial position or to go from the initial position to the bathroom). Therefore we have two possibilities:

- either remove the actions of the player causing the unsatisfactory endings (EA01 - Working at the initial position and EA03 - Going to the bathroom), but that may restrict the player's freedom, so we do not choose this possibility;
- or enrich the contents of the plot:
  - if the player decides to work at the initial position, then the IS controller will ask him to go to the kitchen (for example, a non-player character asks him to take an apple in the fridge);
  - if the player decides to go to the bathroom, then the IS controller will start the strategy of causing the electric shock for him there (by tools such as a hair-dryer, a light bulb,…).

Thus we remodel the DEA game thanks to the editor of scenario (modify the corresponding diagrams) as follows (two lists of inputs and outcomes are unchanging):

- Add two new states to the list of states

Name	IsInitial Available State	Description
Ga	False	Game's state: The IS controller asks the player (who is working at the initial position) to go to the kitchen
Gb	False	Game's state: The IS controller starts the strategy of causing the electric shock for the player in the bathroom

- Modify the list of events/actions (Figure 7 is the diagram in the editor of scenario corresponding to the new list of events/actions)

Figure 7. Diagram corresponding to the new list of events/actions.

Name	Description	Pre condition	Effect
EA01	The player decides to work at the initial position by choosing Iw	Pi, Iw	Pw
EA02	The player decides to go from the initial position to the kitchen by choosing Ik	Pi, Ik	Pk
EA03	The player decides to go from the initial position to the bathroom by choosing Ib	Pi, Ib	Pb
EA04	The IS controller asks the player (who is working at the initial position) to go to the kitchen	Pw, Gi	Pw, Ga
EA05	The player (who is working at the initial position) goes to the kitchen according to the request of the IS controller	Pw, Ga	Pk, Ga
EA06	The IS controller starts the strategy of causing the electric shock for the player in the kitchen (the player has decided to go from the initial position to the kitchen by choosing Ik)	Pk, Gi	Pk, Gk
EA07	The IS controller starts the strategy of causing the electric shock for the player in the kitchen (the player has gone to the kitchen according to the request of the IS controller)	Pk, Ga	Pk, Gk
EA08	The IS controller starts the strategy of causing the electric shock for the player in the bathroom	Pb, Gi	Pb, Gb
EA09	The player gets the electric shock in the kitchen	Pk, Gk	Pe, Gr
EA10	The player gets the electric shock in the bathroom	Pb, Gb	Pe, Gr

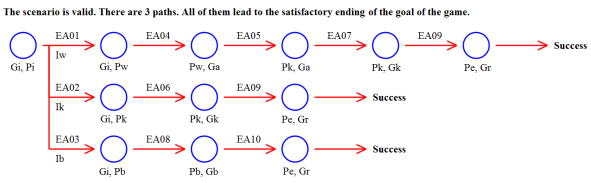

Figure 8. Reduced schema of discourses after the validation.

As a consequence, we receive the new reduced schema of discourses given in Figure 8. We can notice that all the paths lead to the satisfactory ending of the goal of the game (which means the player always gets the electric shock in any case), and at the same time her/his freedom is also guaranteed.

4.3 Creation of the Linear Logic Sequents by the Model Transformations

After modeling the story thanks to the editor of scenario (and validating the generated scenario if necessary), from the received result (four XML files representing four lists of states, inputs, outcomes and events/actions), the analysis module executes two model transformations to create in turn two Linear Logic sequents: the first expresses directly the scenario corresponding with those produced files (it is based on a metamodel that is intermediate to transform into the second); the second is the input of the *Linear Logic model* component (see more Figure 1), it assists the IS controller in managing the unfolding of the story via its automatic reasoning, as a result, it is based on a metamodel that is close to the sequent proof. The following sections describe in detail these metamodels.

4.3.1 Intermediate Metamodel

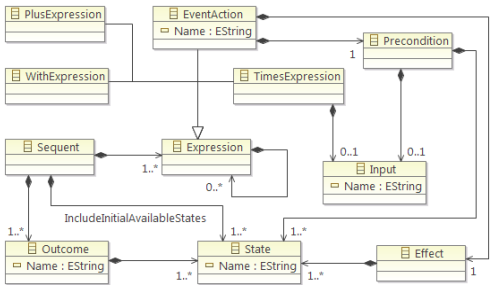

Figure 9. Intermediate metamodel.

The intermediate metamodel is given in Figure 9 where the states, the inputs, the outcomes and the events/actions are similar to the ones created by the editor of scenario. A sequent (corresponding to a scenario) is composed of initial available states, expressions and outcomes. An expression is either a *TimesExpression* or a *WithExpression* or a *PlusExpression* or an *EventAction*, it may also contain other expressions.

- A *TimesExpression* is an expression whose components are connected by the connective ⊗. If the execution of an event/action is decided by a player's choice (enter an input), then there is a *TimesExpression* between the input and the event/action. Besides, a *TimesExpression* also represents the "succession" of the expressions (execute the expressions one after another). For instance, in the DEA game, we have Iw ⊗ EA01 ⊗ EA04 ⊗ EA05 ⊗ EA07 ⊗ EA09 which means that the player decides to execute the event/action EA01 by choosing Iw, then the events/actions EA04, EA05, EA07 and EA09 are continued to execute.
- A *WithExpression* (*PlusExpression*) is an expression whose components are connected by the connective & (⊕) which expresses a choice between these components. If the states in the preconditions of two events/actions are the same, then either the player or the IS controller will decide which event/action will be executed. If it is the player's decision (there are inputs in the events/actions), then these two events/actions (with their succession expressions if any) are connected by a *WithExpression*; on the contrary (there is not any input in the events/actions), these two events/actions (with their succession expressions if any) are connected by a *PlusExpression*. For instance, in the DEA game, as all the preconditions of the three events/actions EA01, EA02 and EA03 contain the state Pi as well as there are the inputs in these events/actions (Iw, Ik, Ib), we uses a *WithExpression*

to connect three *TimesExpressions* (Iw ⊗ EA01 ⊗ EA04 ⊗ EA05 ⊗ EA07 ⊗ EA09 & Ik ⊗ EA02 ⊗ EA06 ⊗ EA09 & Ib ⊗ EA03 ⊗ EA08 ⊗ EA10).

Thus, the Linear Logic sequent representing directly the scenario of the DEA game and created automatically by the analysis module (thanks to the model transformation from the four lists of states, inputs, outcomes and events/actions) is: Gi, Pi, Iw ⊗ EA01 ⊗ EA04 ⊗ EA05 ⊗ EA07 ⊗ EA09 & Ik ⊗ EA02 ⊗ EA06 ⊗ EA09 & Ib ⊗ EA03 ⊗ EA08 ⊗ EA10 ⊢ O (hidden the contents of the events/actions and of the outcome). It is expressed by the reduced XML code segment given in Figure 10.

```
<Sequent>
  <State Name="Gi"/>
  <State Name="Pi"/>
  <WithExpression>
    <TimesExpression>
      <Input Name="Iw"/>
      <EventAction Name="EA01">
        <Precondition>
          <State Name="Pi"/>
          <Input Name="Iw"/>
        </Precondition>
        <Effect>
          <State Name="Pw"/>
        </Effect>
      </EventAction>
      <EventAction Name="EA04">...</EventAction>
      ...
    </TimesExpression>
    <TimesExpression>...</TimesExpression>
    <TimesExpression>...</TimesExpression>
  </WithExpression>
  <Outcome Name="O">
    <State Name="Pe"/>
    <State Name="Gr"/>
  </Outcome>
</Sequent>
```

Figure 10. Sequent based on the intermediate metamodel.

4.3.2 Metamodel Used for the Sequent Proof

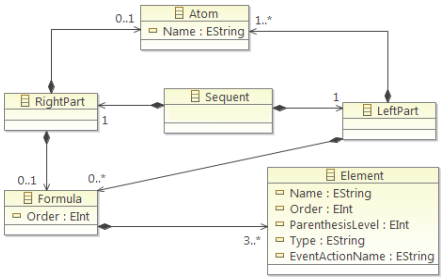

Figure 11. Metamodel used for the sequent proof.

As the aim of the Linear Logic sequent, in our approach, is to assist the IS controller in managing the unfolding of the story via its automatic reasoning, the analysis module continues to transform the sequent based on the intermediate metamodel into the sequent based on the metamodel that is close to the sequent proof. Figure 11 describes in detail this metamodel where a sequent is composed of two parts (separated by ⊢): Left part and Right part.

- The *LeftPart* includes atoms and/or formulas. The atoms in the left part represent the available states at each step in the sequent proof process (at the first step, they are the initial available states), so their *Name* attribute stores the states' name accordingly. The formulas in the left part are distinguished by their order number. Each formula corresponds with an *Expression* that is a "direct child" of the sequent (for instance, the *WithExpression* is a "direct child" of the sequent given in Figure 10, but the three *TimesExpressions* are not because they are three children of the *WithExpression*). A formula is composed of at least three elements, the element's order number expresses its position in the formula. In the left part, there are seven types of element:
  - Type = "Open Parenthesis", Name = "(", EventActionName = "", ParenthesisLevel is the level of the parenthesis;
  - Type = "Close Parenthesis", Name = ")", EventActionName = "", ParenthesisLevel is the level of the parenthesis: The transformation from an *Expression* into a formula, in several cases, needs be added some parentheses to ensure the meaning of the *Expression*, for instance, in the DEA game, Iw ⊗ EA01 is transformed into Iw ⊗ (Pi ⊗ Iw ⊸ Pw);
  - Type = "Additive Conjunction", Name = "with", ParenthesisLevel = "0", EventActionName = "": These elements are added between the components of an *WithExpression*;
  - Type = "Additive Disjunction", Name = "plus", ParenthesisLevel = "0", EventActionName = "": These elements are added between the components of a *PlusExpression*;
  - Type = "Multiplicative Conjunction", Name = "times", ParenthesisLevel = "0", EventActionName = "": These elements are added between the components of a *TimesExpression*, or between the components (state, input) in the *Precondition* and between the states in the *Effect* of an event/action;
  - Type = "Linear Implication", Name = "imply", ParenthesisLevel = "0": This element is added between the *Precondition* and the *Effect* of an event/action (as a consequence, an event/action only has a unique format: A1 ⊗ A2 ⊗ … ⊗ An ⊸ B1 ⊗ B2 ⊗ … ⊗ Bm), its *EventActionName* attribute stores the name of that event/action in the game;
  - Type = "Atom", ParenthesisLevel = "0", EventActionName = "": These elements represent the states or the inputs in the *Expressions*, so their *Name* attribute is the states' name or the inputs' name accordingly.
- The *RightPart* only includes either one atom or one formula. If the scenario (sequent) only has one outcome and this outcome only contains one state, then the right part only includes one atom which corresponds to this state, so its *Name* attribute stores the state's name. In other cases (the scenario has one outcome but this outcome contains some states, or the scenario has several outcomes), the right part includes one formula (so its *Order* attribute = "1"). This formula is composed of at least three elements, the element's order number expresses its position in the formula. In the right part, there are three types of element:
  - Type = "Atom", ParenthesisLevel = "0", EventActionName = "": These elements represent the states in the outcome(s), so their *Name* attribute stores the states' name;
  - Type = "Multiplicative Conjunction", Name = "times", ParenthesisLevel = "0", EventActionName =

“”: These elements are added between the states of one outcome;

- Type = “Additive Disjunction”, Name = “plus”, ParenthesisLevel = “0”, EventActionName = “”: If there are several outcomes, then these elements are added between them.

```
<Sequent>
  <LeftPart>
    <Atom Name="Gi"/>
    <Atom Name="Pi"/>
    <Formula Order="1">
      <Element Name="Iw" Order="1" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name="times" Order="2" ParenthesisLevel="0" Type="Multiplicative Conjunction" EventActionName=""/>
      <Element Name="(" Order="3" ParenthesisLevel="1" Type="Open Parenthesis" EventActionName=""/>
      <Element Name="Pi" Order="4" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name="times" Order="5" ParenthesisLevel="0" Type="Multiplicative Conjunction" EventActionName=""/>
      <Element Name="Iw" Order="6" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name="imply" Order="7" ParenthesisLevel="0" Type="Linear Implication" EventActionName="EA01"/>
      <Element Name="Pw" Order="8" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name=")" Order="9" ParenthesisLevel="1" Type="Close Parenthesis" EventActionName=""/>
      <Element Name="times" Order="10" ParenthesisLevel="0" Type="Multiplicative Conjunction" EventActionName=""/>
      ...
      <Element Name="with" Order="50" ParenthesisLevel="0" Type="Additive Conjunction" EventActionName=""/>
      <Element Name="Ik" Order="51" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name="times" Order="52" ParenthesisLevel="0" Type="Multiplicative Conjunction" EventActionName=""/>
      ...
    </Formula>
  </LeftPart>
  <RightPart>
    <Formula Order="1">
      <Element Name="Pe" Order="1" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
      <Element Name="times" Order="2" ParenthesisLevel="0" Type="Multiplicative Conjunction" EventActionName=""/>
      <Element Name="Gr" Order="3" ParenthesisLevel="0" Type="Atom" EventActionName=""/>
    </Formula>
  </RightPart>
</Sequent>
```

Figure 12. Sequent based on the metamodel that is close to the sequent proof.

Finally, as an example, Figure 12 gives the reduced XML code segment which represents the Linear Logic sequent (corresponding with the DEA game) based on the metamodel that is close to the sequent proof. This Linear Logic sequent assists the IS controller in managing the unfolding of the game via its reasoning and is automatically created by the analysis module (thanks to the model transformation from the sequent based on the intermediate metamodel).

5. CONCLUSION

In the paper, we have presented the methodology for authors to derive a valid scenario of an IS (even when they do not have any knowledge of Linear Logic). We have explained the implementation of the methodology via a detailed presentation of the steps in the process of IS modeling and illustrated those with the DEA game. Concerning future works to ameliorate the Linear Logic approach for IS modeling, in addition to ensuring that the received scenario leads to satisfactory endings of author’s goals, it will be validated on two aspects:

- Firstly, does the scenario follow exactly the structure of discourse that has been pre-defined by the author?
- Secondly, is the scenario “ludic”? In [3], we have proposed a new class of properties (impartiality, complexity, concurrence) which allows estimating the relevance of a scenario, and as a result, we will have to quantify theses properties for each game as well as test them by Linear Logic. Thus, we may evaluate the scenario’s quality and hence show an “interesting scenario” for a game.

6. ACKNOWLEDGMENTS

This work has been funded (in part) by the European Commission under grant agreement IRIS (FP7-ICT-231824).

7. REFERENCES

[1] Aylett, R. 1999. Narrative in Virtual Environments – Towards Emergent Narrative. In *Proceedings of the AAAI Symposium on Narrative Intelligence*. AAAI Press, Menlo Park, 83-86.

[2] Cavazza, M., Charles, F., and Mead, S.J. 2002. Character-based Interactive Storytelling. In *IEEE Intelligent Systems, special issue on AI in Interactive Entertainment*, 17-24.

[3] Champagnat, R., Prigent, A., and Estraillier, P. 2005. Scenario building based on formal methods and adaptative execution. In: ISAGA 2005 - International Simulation and Gaming Association, Atlanta, USA.

[4] Dang, K. D., Champagnat, R., and Augeraud, M. 2010. Modeling of Interactive Storytelling and Validation of Scenario by Means of Linear Logic. In: Aylett, R. et al. (eds.) ICIDS 2010. LNCS, vol. 6432, 153-164.

[5] Dang, K. D., Champagnat, R., and Augeraud, M. 2011. Interactive Storytelling Control for Video Games: an Approach Based on a Linear Logic Model. Internal report L3i-2011-001, L3i laboratory, University of La Rochelle (may be offered if necessary).

[6] Delmas, G., Champagnat, R., and Augeraud, M. 2009. From Tabletop RPG to Interactive Storytelling: Definition of a Story Manager for Videogames. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, 121-126.

[7] Girard, J.-Y. 1987. Linear Logic. *Theoretical Computer Science* 50(1), 1-101.

[8] Hebert, L. 2006. Tools for Text and Image Analysis: An Introduction to Applied Semiotics. Texto! (last accessed 05/28/2011), http://www.revue-texto.net/1996-2007/Parutions/Livres-E/Hebert_AS/Hebert_Tools.html.

[9] Magerko, B. 2005. Story Representation and Interactive Drama. In *1st Artificial Intelligence and Interactive Digital Entertainment Conference*. Los Angeles, California.

[10] Mateas, M. 2002. Interactive Drama, Art, and Artificial Intelligence. PhD Thesis, School of Computer Science, Carnegie Mellon University.

[11] Riedl, M.O. 2004. Narrative Generation: Balancing Plot and Character. PhD Thesis, Department of Computer Science, North Carolina State University.

[12] Si, M., Marsella, S.C., and Pynadath, D.V. 2009. Directorial Control in a Decision-Theoretic Framework for Interactive Narrative. In: Iurgel, I.A., Zagalo, N., Petta, P. (eds.) ICIDS 2009. LNCS, vol. 5915, 221-233.

[13] Szilas, N. 2003. IDtension: a Narrative Engine for Interactive Drama. TIDSE. LNCS 3105. Darmstadt, Germany, 183-203.

[14] Young, R.M., Riedl, M.O., Brandy, M., Martin, J., and Saretto, C.J. 2004. An Architecture for Integrating Plan-Based Behavior Generation with Interactive Game Environments. *Journal of Game Development*, 51-70.

[15] Graphical Modeling Framework (last accessed 05/28/2011), http://wiki.eclipse.org/Graphical_Modeling_Framework.

[16] Object Management Group/Business Process Management Initiative (last accessed 05/28/2011), http://www.bpmn.org.