Generation of Obligation and Prohibition Dilemmas Using Knowledge Models

Azzeddine Benabbou, Domitile Lourdeaux and Dominique Lenne
Sorbonne universités, Université de technologie de Compiègne, CNRS UMR 7253 Heudiasyc
57 Av. Landshut – 60203 COMPIEGNE Cedex, France
{azzeddine.benabbou – domitile.lourdeaux – dominique.lenne}@hds.utc.fr

***Abstract*—Under the project MacCoy Critical, we would like to train individuals, in virtual environments, to handle critical situations such as dilemmas. These latter refer to situations where there is no "good" solution. In other words, situations that lead to negative consequences whichever choice is made. Our objective is to develop a Scenario Orchestration System that generates dilemma situations dynamically without having to write them beforehand. The approach consists in using Knowledge Models to extract necessary properties for dilemmas to emerge. In this article we present this approach and expose a proof of concept of the generation process.**

Keywords— Scenario Orchestration, Dilemma, Knowledge Models, Virtual Environments

I. INTRODUCTION

Sometimes individuals face critical situations where internal (e.g. tiredness, lack of concentration) and external (e.g. extreme weather conditions, incompetence of colleagues) factors make them difficult to handle. In order to prevent disastrous consequences from happening, individuals need to be trained for such situations. They must be confronted to various situations where they will have to understand their environment and act, sometimes urgently, to develop the needed competencies.

A. Virtual Environments for training

While training for "normal" situations (e.g. repairing a car engine) in genuine conditions can be an easy exercise to reproduce, training for critical situations represents a serious challenge. Virtual Reality can address this issue. It provides tools and techniques that enable the systems to simulate such complex situations especially when cost, accessibility and dangerousness prevent learners from being put in genuine situations. In order to foster the development of new skills, the situations have to be adapted for each learner. It means that the criticality of the situations has to be adjusted according to his/her actions. Also, the situations must involve not only the mastered skills but also new skills that are close to the ones already acquired [1]. To support this kind of learning, it is necessary to generate a broad spectrum of scenarios. A first approach consists in writing all the possible scenarios taking into consideration all the parameters (e.g. actions, events, user's profile). This insures a total control of the simulation. However, as far as complex domains are concerned (large number of objects, agents, possible actions etc…) this approach leads to what is called Authoring Bottleneck [2]. In order to address this issue, it is necessary to put in place a Scenario Orchestration System capable of creating adaptable environments without having to define, explicitly, all the possible scenarios. We defines an Orchestration System as a system that is composed of one or several orchestration languages used to model the scenario content and/or the scenario goals. It is also composed of an Orchestration Engine that manages dynamically the realization of the scenario. We make the hypothesis that it is possible to generate situations dynamically using Knowledge Models that underlay the simulation.

B. Context and objectives

Under the project *MacCoy Critical*, we are interested in generating critical situations in order to train for non-technical skills[3]. Dilemma is one of these situations. According to Oxford Dictionary, a dilemma is a "*situation in which a difficult choice has to be made between two or more alternatives, especially ones that are equally undesirable*". Dilemmas are faced on workplaces on a daily basis. In healthcare for example, nurses, are confronted to difficult situations where there is no "good" solution. If they are not trained to handle such situations, they will certainly not be able to deal with them when they occur. Lecomte listed several reasons why these nurses should be trained to face what are called "ethical dilemmas" [4]. For example, she spoke about being able to step back and look at the overall picture before taking a decision, being able to argue, negotiate and make compromises. Dilemmas are also encountered in other fields such as car driving. The most common one is when the drivers have to choose between two evils, running over pedestrians or sacrificing themselves and their passengers. This dilemma has been the subject of several studies such as [5] to determine which behavior a driverless car should adopt.

Designing a Virtual Environment for training does not only involve computer scientists, but it also involves instructors, experts, ergonomists etc. Usually these actors manipulates different Knowledge Models (e.g. Tasks models, World Models) according to their field of expertise. Our purpose is to design a Scenario Orchestration System capable of using these different expert models to identify potential dilemma situations and thus generate them dynamically. In this article, we present our approach of dynamic generation of dilemma situations. In the first section, we discuss some related work. In the second section, we detail some dilemma properties and propose a formalization. Then, in the fourth section, we present the global architecture of the system. After that, in the fifth section we expose the Knowledge Models used by the Orchestration System. In the sixth section, we detail our approach of dilemma generation. Finally, in the last two sections, we present a proof

of concept of our system and discuss the results of our evaluation. All along this article, our work is illustrated by examples on car driving.

II. Related work

In several works of the Institute of Creative Technologies at the University of Southern California [6], researchers implemented an Army peacekeeping scenario where the user plays the role of a U.S. Army lieutenant. He is confronted with the following dilemma: he must choose between sending his troops to help his platoon downtown and securing a landing zone for a medevac helicopter in order to assist a local boy in critical situation. In medical field, Gratch and Marsella [7] modeled the behavior of a doctor who must choose between administrating large doses of morphine to his young boy patient to reduce the pain, but hasten his death (family will) , and extending the patient's "painful" life (doctor duty). In the literature, we also find several works related to the Trolley dilemma. Its original version[1] was stated by Philippa Foot [8] in 1967. It has been subject, as well as its derivatives, to several paper studies [9], [10] and virtual environments based ones [11], [12]. In all the work listed above, the dilemmas are written in advance before the execution of the simulation. This "scripted" approach enables the authors to describe precisely and accurately the dilemma situations. Another approach consists in generating the dilemma dynamically during the simulation. GADIN [13], for example, is an interactive narrative engine that unfolds a story based on a user's response to dilemma situations. The authors propose five dilemma categories: Betrayal, Sacrifice, Greater Good, Take Down and Favor.

The scripted approach enables the authors to describe accurately the dilemma situations to present to the user. It might be relevant for social sciences experiments and training for specific and restricted situations. However, it remains unsuitable for training to various situations. In fact, the necessity of giving the learner a freedom of action and ensuring the variability of situations, makes it difficult (if not impossible) to write all the possible dilemma situations. To remedy this problem, GADIN proposes an interesting generative approach. However it suffers from two limitations: (1) the freedom of action of the user is limited and (2) the dilemma categories depend on the social relationships (friends or enemies) that the user have with the other characters. Therefore, it is impossible to generate dilemmas if the user is the only character in the environment or if s/he does not have any social relationship with the others. Unlike the "scripted" approaches, ours consists in generating dynamically the dilemma situations without having to write them in advance. It enables the system to adjust the criticality during the simulation, and therefore, to adapt the nature of the dilemma according to the learner's actions and his/her dynamic profile. Moreover, the approach relies on an algorithm that does not restrict dilemmas to categories based on the social relations between the characters.

III. Dilemma Model

A dilemma situation can be easily determined by humans. Thus, an instructor, using an appropriate user interface, can identify a dilemma situation and propose it to the learner during the training session. However, since we adopt a dynamic generation approach, this has to be automated by the Orchestration System, and this is difficult without any semantic. Therefore, it raises the following questions: How do we model semantically a dilemma situation? And how can a Scenario Orchestration System generate it dynamically using this semantic? The Oxford Dictionary definition of dilemma points out an important property about dilemma situation. It says that there are always negative consequences whatever choices are made by the agent. In the literature, some authors differentiate between *obligation dilemma* and *prohibition dilemma* [14]. The former refers to situations in which all the feasible actions are obligatory but can't all be realized. While the latter refers to cases in which all the feasible actions are forbidden but at least one has to be realized.

A. Necessary conditions

In *obligation dilemmas*, the necessary condition is that the actions can't all be performed. In other words, the choice presented to the agent is exclusive. Because if an agent is able to realize all the actions, without any negative consequence, there will be no dilemma at all. The challenge for the Orchestration System, in this case, is to prescribe a World State that guarantees this condition. Thus, we propose to generate situations that require contradictory actions. Two actions are contradictory if they are incompatible and requested to be performed at the same time (e.g. "to turn left" and "to turn right"). In this case, the agent will face a situation where he cannot achieve both actions, which makes the choice exclusive. We distinguish two types of incompatibility between the actions:

- Nomological: it refers to actions that are incompatible by nature (e.g. "to increase" vs "to decrease", "to open" vs "to close", "to move forward" vs "to move backward").
- Scenaristic/Regulatory: it refers to actions that are nomologically compatible, but, in a particular context, become incompatible due to scenario or regulatory constraints (e.g. an order, an instruction or a rule). For example, "speaking on the phone" and "driving" are two compatible actions in a nomological point of view. However, in a particular context, according to the Highway Code of certain countries, they become incompatible.

In *prohibition dilemmas*, the necessary condition is that the agent ought to do one action at least. The challenge for the Orchestration System is to push the agent to choose an action, knowing that each of them leads to negative consequences. To achieve that, we propose to generate situations where there are also negative consequences when the agent does not make a choice.

[1] The protagonist is a driver of a runaway tram which he can only steer from one narrow track on to another; five men are working on one track and one man on the other; anyone on the track he enters is bound to be killed

B. Formalization

1) Obligation dilemma

Let a_1 and a_2 be two actions that an agent *AG* can perform. For some reasons, *AG* ought to do a_1. In other words, not doing a_1 leads to negative consequences $NC_{\neg a_1}$ (discussed in section C). For some other reasons *AG* ought to do a_2. In other words, not doing a_2 leads to negative consequences $NC_{\neg a_2}$. The necessary condition in an *obligation dilemma* states, that only one action can be performed. If not so, performing the two actions leads to negative consequences too. Thus, *S* is an *obligation dilemma* situation if:

$$\neg a_1 \xrightarrow{Leads\ to} NC_{\neg a_1}$$
$$\neg a_2 \xrightarrow{Leads\ to} NC_{\neg a_2}$$
$$(a_1 \vee a_2) \wedge \neg(a_1 \wedge a_2)\ \ OR\ \ \left(a_1 \wedge a_2 \xrightarrow{Leads\ to} NC_{a_1 \wedge a_2}\right)$$

This can be generalized for n actions as follows. *S* is an *obligation dilemma* situation if:

$$\exists\ A_{nc} \subset A\ with\ A_{nc} = \bigcup_{i=1}^{n} a_i\ ,\ n \geq 2$$
$$such\ as\ \forall\ a \in A_{nc}\ , \neg a \xrightarrow{Leads\ to} NC_{\neg a}$$
$$\left(\vee_1^n a_i\ \wedge\ \neg \wedge_1^n a_i\right)\ OR\ \left(\wedge_i^n a_i \xrightarrow{Leads\ to}\ NC_{\wedge_i^n a_i}\right)$$

Where A is the set of actions, A_{nc} is the set of actions that lead to negative consequences if not performed and NC_a is the negative consequences of doing an action a.

2) Prohibition dilemma

Let a_1 and a_2 be two actions that an agent *AG* can perform. For some reasons, *AG* ought not to do a_1. In other words, doing a_1 leads to negative consequences (discussed in C). For some other reasons *AG* ought not to do a_2. In other words, doing a_2 leads to negative consequences. A necessary condition in a *prohibition dilemma* situation is that at least one action has to be performed. Thus, *S* is a *prohibition dilemma* situation if:

$$a_1 \xrightarrow{Leads\ to} NC_{a_1}$$
$$a_2 \xrightarrow{Leads\ to} NC_{a_2}$$
$$\neg a_1 \wedge \neg a_2 \xrightarrow{Leads\ to} NC_{\neg a_1 \wedge \neg a_2}$$

This can be generalized for *n* actions as follows. *S* is an *prohibition dilemma* situation if:

$$\exists\ A_{nc} \subset A\ with\ A_{nc} = \bigcup_{i=1}^{n} a_i\ ,\ n \geq 2$$
$$such\ as\ \forall\ a\ \in A_{nc}\ , a \xrightarrow{Leads\ to} NC_a$$
$$\wedge_i^n \neg a_i \xrightarrow{Leads\ to}\ NC_{\wedge_i^n \neg a_i}$$

C. Negative consequences

Let us recall that an important property of a dilemma is that there is always negative consequences that follow the choice of the agent. The consequences can be observed in terms of:

- *Severity*: degree of injury, number of victims or material damage (e.g. hit a pedestrian, a tree or another vehicle),
- *Violations*: law-breaking, value violation or non-respect of norms, rules or instructions etc. (e.g. running a red light),
- *Points*: performance points, driving license points etc.

IV. GLOBAL ARCHITECTURE

In our architecture, the *Orchestration Engine* is composed of the *Scenario Engine* module and the *Planner*. The former is responsible of generating critical situations such as dilemmas. It receives pedagogical instructions from the *Learner diagnosis* module. These instructions are composed of (1) a pedagogical intention such as "verify", "reinforce" or "destabilize" a competence and (2) a level of criticality. The instruction is transformed by the *Scenario Engine* to scenario goals that are sent to the *Planner*. The latter communicates directly with the Virtual Environment. It is responsible of performing the adequate adjustments in order to direct the simulation towards the situation that fulfills the scenario goals. The Orchestration System uses three Knowledge Models. These models are presented in the next section. The Virtual Environment as well as the *Learner diagnosis* module are managed by our partners in the project *Figure 1* gives an overview of our architecture.

V. KNOWLEDGE MODELS

Designing a Virtual Environment for training is not only the matter of software developers. It involves also several actors such as instructors, psychologists, ergonomists etc. Thus, the description of the domain (e.g. objects, their properties, actions, events) should be independent from the graphical representation of the Virtual Environment. In fact, these experts should manipulate adequate models, according to their field of expertise, in order to facilitate a collaborative conception of the training environment. Ideally, it should use at least three models. A *World Model* that describes the domain entities, their properties and the relations between them. A *Tasks Model* that describes the set of tasks to be performed, their properties and the relations between them. And finally, particularly when training for critical situations, a *Causality Model* that describes

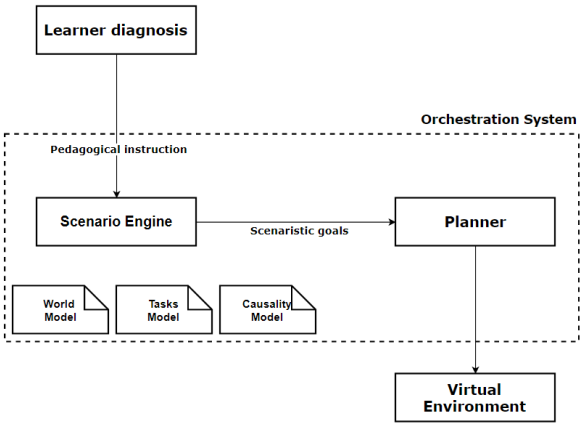

Figure 1 - System architecture

the causal links between the World events. These models are presented in the next three sections.

A. World model

The *World Model* is an ontology intended to be filled by experts of the domain. It is a set of concepts connected with subsumption and/or semantic links. This representation is intelligible and interpretable by a computer system. It provides a means to represent the World entities (e.g. objects, actions, events), their properties and the relations between them at different levels of abstraction. It facilitates the generation of a variety of situations. Moreover, this semantic representation enables the Scenario Orchestration System to query the World base in order to retrieve any relevant information (e.g. what are the traffic lights that are situated within a 100m radius?). This ontological representation is paired with rules in order to control the evolution of the World. *Figure 2* shows a fragment of a *World Model* composed of an instance of a Traffic light.

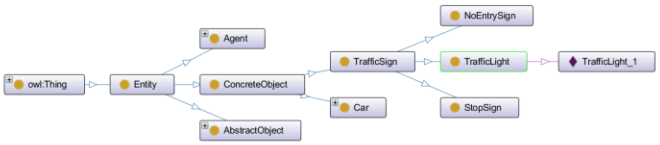

Figure 2 - Fragment of a World Model

B. Tasks model

The *Tasks Model* is to be filled by ergonomists according to the observed activity in the field. It expresses the cognitive perception of the tasks by the operators. Thus, it would be relevant to use a representation that operationalizes cognitive ergonomics principles. For example, in certain hierarchical representations, the tasks are composed of subtasks on several levels. These latter are connected by *Constructors* that inform about the logical and temporal relations between the tasks. The tasks have preconditions and postconditions. They are aggregate of assertions formulated as triples: "Subject Predicate Object" (e.g. Light has-color Red). This formulation is relevant since we use ontological representation in the *World Model*. The preconditions are the conditions that make the task realization relevant and/or favorable. The postconditions, also called satisfactory conditions, are the conditions that have to be satisfied by the World in order to consider a task as achieved. *Figure 3* shows a fragment of a *Tasks Model* that describes the task "Handle_red_light" composed of two subtasks. The task is relevant only if there is a red traffic light (precondition), and it is considered achieved if the vehicle stopped.

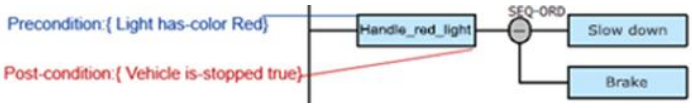

Figure 3 - Fragment of a Tasks model

C. Causality model

The *Causality Model* is an acyclic and oriented graph that expresses the causality chains of the World that are relevant for the scenario. The nodes refer to specific events or actions that are defined in the *World Model*. They can be tagged with any relevant information such as semantic (e.g. severity = 3). They are connected by causality and/or subsumption links. Thus, we can easily represent the causality chains that leads to the negative consequences presented in section III.C. The model is composed of *AND* and *OR* gates that enable the authors to model events caused by more than one action or event. Another component of the *Causality Model* is the Prevention Barriers. They are means that prevent an event from happening (e.g. "Stop at red traffic light" prevent from having a "Contravention"). With this barriers representation, an Orchestration Engine can identify which actions lead to negative consequences if not realized. Figure 4 shows a fragment of a *Causality Model* that illustrates the fact that there could be a Highway Code violation if the agent runs a Stop Sign or a Red Traffic Light. This can be prevented by the barriers "Handle_stop" and "Handle_red_light". The events are represented by rectangles while the actions are represented by rounded rectangles.

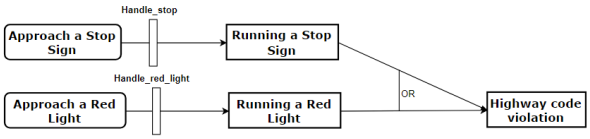

Figure 4 - Fragment of a Causality model

VI. DILEMMAS GENERATION

We described above the different types of dilemmas and stated the necessary conditions that these situations have to satisfy. We also presented the different models used by the Orchestration Engine to generate such situations. In this section, we detail the different steps of the dilemma generation process (*Figure 5*). We explain how the Orchestration Engine uses the Knowledge Models to extract the necessary properties, and thus enabling dilemma situations to emerge.

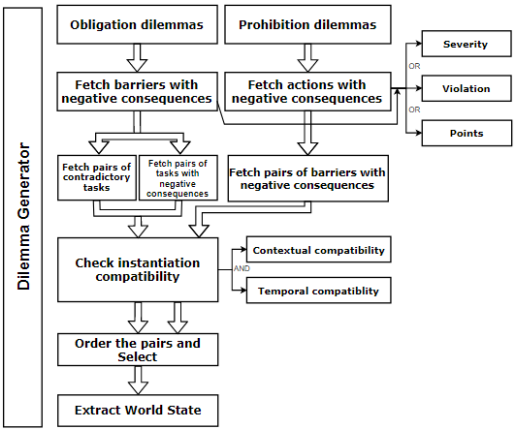

Figure 5 - Dilemma generation steps

A. Fetch actions/barriers with negative consequences

The *Causality Model* enables us to represent the World's events and their consequences (positive, negative or neutral). The first step consists in identifying the relevant actions for each dilemma type. With regards to obligation dilemmas, we are interested in identifying the actions that, if not performed, lead to negative consequences. In other words, the ones that lead to the nodes "Severity", "Violation" and/or "Points". These actions correspond to the Barriers. In fact, if a Barrier is not put in place

by the agent, it triggers the events posterior to it. Thus, we select all the barriers which posterior events could lead to one or more type(s) of negative consequences. There has to be no other barrier between the selected one and the consequence nodes. Otherwise, the agent will be able to avoid the consequences. In this case, it is the last barrier that is selected. Regarding prohibition dilemmas, we are interested in identifying the actions that leads to negative consequences. For that, we list all the actions nodes that are linked to the nodes "Severity", "Violation" and/or "Points". Same in this case, there has to be no Barrier between the action node and the negative consequences nodes. Otherwise the agent will be able to avoid the consequences. For both types of dilemmas, at this stage, we can already refine the lists of actions according to the pedagogical instructions (e.g. severity > 2). To illustrate what was said above, let us take *Figure 6* as an example.

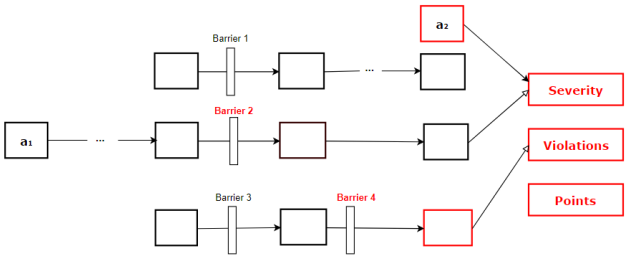

Figure 6 – Barriers and actions selection

In this example, Barrier 1 is discarded because its posterior events don't lead to any of the negative consequences nodes. Barrier 3 is also rejected because there is another Barrier between it and the negative consequences nodes. At the end, only the Barriers 2 and 4 are retained. Same things applies for the actions. Thus, only a_2 is retained.

B. *Contradictory pair of taks (For Obligation dilemma)*

The extraction of potential contradictory tasks is done using the *Tasks Model* (Algorithm 1). We scan by pairs the set of the selected Barriers from the *Causality Model* (tasks that lead to negative consequences if not realized). For each pair of tasks, we examine their postconditions (satisfactory conditions). If they are incompatible, then the pair is nomologically incompatible. We consider that two conditions are incompatible if, for the same couple (subject predicate), the object is different (e.g. "Vehicle is-stopped true" and "Vehicle is-stopped false"). This algorithm is to be generalized for *n tasks*. We will then use lists of contradictory tasks instead of pairs.

Algorithm 1: Search for contradictory actions
Function fetchContradictoryActions(Tasks ⊂ Barriers)
Begin
PCT ← {} /* set of Pairs of Contradictory Tasks*/
For all t1 ∈ Tasks **do**
For all t2 ∈ Tasks **do**
If (t1.postCondition.subject = t2. postCondition.subject **and**
t1.postCondition.predicate = t2. postCondition.predicate **and**
t1.postCondition.object ≠ t2. postCondition.object)
If {t1,t2} ∉ PCT **then**
PCT ← PCT ∪ {{t1,t2}}
Return PCT;
End

C. *Pair of actions with negative consequences (For Obligation dilemma)*

If the pair of actions are not contradictory, the system tries to find if the actions lead to negatives consequences if performed together. Algorithm 2 details this process.

Algorithm 2: Pair of actions that leads to negative consequences
Function fetchActionsConnectedWithAND(Actions)
Begin
PA ← {} /* set of Pairs of Actions*/
For all a1 ∈ Actions **do**
For all a2 ∈ Actions **do**
If ({"AND"} ∈ CommonDescendant(a1,a2))
PA ← PA ∪ {{a1,a2}}
Return PA;
End

D. *Pair of actions with negative consequences (For Prohibition dilemma)*

The necessary condition of a *Prohibition dilemma* is that the agent has to choose one action at least. We explained that to push him/her to do that, not making a choice must be penalizing too. In other words not doing any of the feasible actions leads to negative consequences. To identify such actions, the Orchestration System uses the *Causality Model*. It fetches for barriers that are connected by an AND gate according to Algorithm 3.

Algorithm 3: Search for pair of barriers that leads to negative consequences
Function fetchBarriersConnectedWithAND(Barriers)
Begin
PB ← {} /* set of Pairs of Barriers*/
For all b1 ∈ Barriers **do**
For all b2 ∈ Barriers **do**
If ({"AND"} ∈ CommonDescendant(b1,b2))
PB ← PB ∪ {{b1,b2}}
Return PB;
End

E. *Instantiation compatibility*

To ensure that the system can instantiate a situation involving the selected pair of tasks, these latters have to be contextually compatible. It means that they must have compatible preconditions. For example, let us consider two tasks "Open_the_door" and "Close_the_door" that have respectively the preconditions "Door is-open false" and "Door is-open true" and the postconditions "Door is-open true" and "Door is-open false". These tasks are nomologically incompatible (analysis of the postconditions). Therefore, they could possibly be used for generating a dilemma. However, they are contextually incompatible (analysis of the preconditions). In fact, the instantiation of such situation requires the door to be opened and closed at the same time, which is impossible. The pair of task is then discarded.

Furthermore, the pair of tasks has to be temporally compatible. It means that the realization of one task needs to be independent of the realization of the other. In the Tasks Model, the tasks are connected by the Temporal Constructors of their parent. They can be Independent, Sequential or Parallel. Having

tasks that are sequentially related means that the realization of one task is an implicit precondition of the other. Thus, we discard the tasks that have a common ancestor with a Sequential Constructor.

F. Pair of tasks selection

At this stage, the Orchestration Engine possesses two lists of filtered pair of tasks. In order to identify which pair is the most relevant, we attribute a score to each pair according to Pedagogical and Scenario Constraints. The Pedagogical Constraints consist of: the minimum or maximum degree of Severity, the difference of Severity between the two tasks of the pair and finally the type of negative consequences. The Scenario Constraints refer to the probability of occurrence of a situation involving the tasks of the pair. Each constraint has a weight according to which the score is calculated. This latter enables the Orchestration Engine to rank the pairs and thus to be able to propose to most suitable dilemma with respect to the Pedagogical and Scenario constraints.

G. Extract partial World State

Finally, once a pair is selected, the Orchestration Engine extracts the preconditions of the tasks. These preconditions represent a partial description of the World. It corresponds to a World State that needs to be instantiated. This state is transmitted as a goal to the Planner that is in charge of directing the scenario to the given state. At the end, the instruction are sent to the Virtual Environment in order to instantiate the situation graphically.

VII. PROOF OF CONCEPT

Figure 7 - Screenshot of the virtual environment

The approach described above has been subject to a first implementation in a vehicle driving simulation. We developed a first version of our *Scenario Engine* module and a virtual environment using Unity3D (*Figure 7*). The environment is composed of city buildings, road signs, traffic lights and other virtual agents (vehicles and pedestrians). The player controls a first-person vehicle using a keyboard-mouse or a driving set configuration. At this stage of our work, there is no initial instruction given to the player. S/he is free to drive anywhere s/he wants.

We used the Tasks Model presented in *Figure 8*. The model is fairly simple for the sake of clarity. It describes three independent tasks: "*Handle_aquaplaning*", "*Handle_red_light*" and "*Handle_stop*".

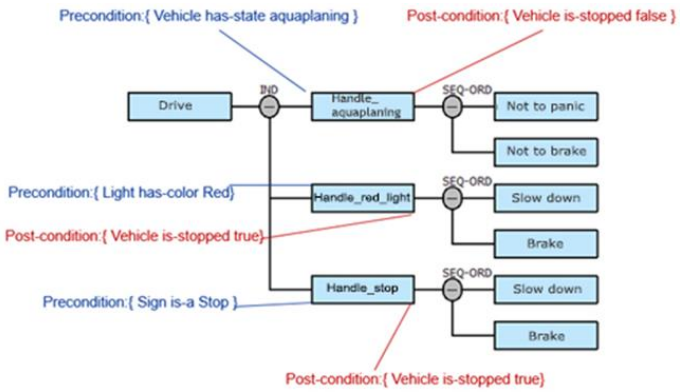

Figure 8 - Driving Task Model

We also used the Causality Model presented in *Figure 9*. It describes several events such as "*Running a Stop Sign*" and "*Running a Red Light*" that leads to a Highway Code violation if the appropriate Barriers "*Handle_Stop*" and "*Handle_red_light"* are not realized. The Highway Code violation can also be caused by "Driving" and "Answering a phone call" at the same time. The model describes also the risk of losing the control of the vehicle if the aquaplaning are not well handled. Finally another type of violation: "*Breaking a promise*" can be caused by arriving late at home or by making the mother angry.

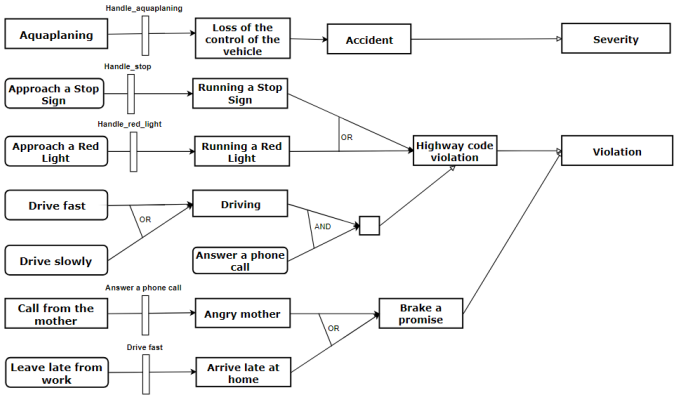

Figure 9 - Causality Model

The execution of the dilemma generation algorithm (*Figure 5*) is detailed below:

Step 1.1: *Fetch Barriers with negative consequences (Obligation dilemmas)*

According to the Causality Model, the Barriers that, if not put in place, could lead to negative consequences are: "*Handle_stop"*, "*Handle_red_light"*, "*Handle_aquaplaning", "Answer a phone call"* and *"Drive fast"*.

Step 1.2: *Fetch actions with negative consequences (prohibition dilemmas)*

According to the Causality Model, the actions that could lead to negative consequences are "*Approach a Stop sign*", "*Approach a Red light*", "*Drive fast*", "*Drive slowly*", "*Answer a phone call"* and "*leave late from work*". The first two actions are discarded because there are Barriers between them and the negative consequences nodes.

Step 2.1.a: *Fetch pairs of contradictory tasks*.

Using the list of Step 1.1, the system looks for the possible contradictory tasks. It returns the following pairs:

Pair1: {"*Handle_stop*", "*Handle_aquaplaning*"}
Pair2: {"*Handle_red_light*", "*Handle_aquaplaning*"}

In fact, the postconditions of the tasks, for both pairs, are incompatible: (*Vehicle is-stopped false* vs *Vehicle is-stopped true*). The pair {"*Handle_stop*", "*Handle_red_light*"} is rejected because the postconditions of the tasks are compatible: (Vehicle is-stopped true vs Vehicle is-stopped true).

Step 2.1.b: *Fetch pairs of actions with negative consequences*.

Using the list of Step 1.1, the system looks for actions that, if performed together, leads to negative consequences. The system returns the following pair:

Pair3: {"*Answer a phone call*", "*Drive fast*"}

In fact, "*Answer a phone call*" and "*Drive fast*" has a common descendent AND node.

Step 2.1.c: *Fetch pairs of barriers with negative consequences*.

From the list returned in Step 1.2, the system looks for pair of Barriers that leads to negative consequences. The system returns no pair. In fact, no Barriers have a common descendent *AND* node.

Step 3: *Check instantiation compatibility* (for both types of dilemma).

The common ancestor of the tasks of *Pair1* is the task "*Drive*" that has an independent Temporal Constructor (IND). The same thing applies for the tasks of *Pair2*. *Pair3* tasks are independent too, they have no temporal connection. Furthermore, the preconditions of the tasks of *Pair1* are compatible (Sign is-a Stop vs Vehicle has-state aquaplaning). The same thing applies for *Pair2* and *Pair3*. Consequently, *Pair1*, *Pair2* and *Pair3* are retained because they are temporally and contextually compatible.

Step 4: *Order the pairs and Select (for both types of dilemmas)*

At this stage, there are three candidate pairs. The dilemmas created by each pair would be the following:

- Pair1: In aquaplaning situation, should the agent brake to stop (respect the Stop sign) with the risk of losing the control of the vehicle or should s/he not to brake and runs the Stop sign (violate the Highway Code) to keep the control of the vehicle?
- Pair2: In aquaplaning situation, should the agent brake to stop (respect the Red traffic light) with the risk of losing the control of the vehicle or should s/he not to brake and runs the Red light (violate the Highway Code) to keep the control of the vehicle?
- Pair3: While driving, should the agent answer his/her mother *important* phone call (Highway Code violation) to prevent her from being angry and to arrive home at time. Should s/he stop somewhere to answer the call but arrive late. Or should s/he ignore the call to arrive home at time (Braking a promise in both cases).

For this example, the pairs are supposed equally relevant (have the same score). Thus a pair is selected randomly. We suppose that is *Pair2*.

Step 5: *Extract World State*.

Finally the World State is extracted. It consists of the preconditions of the *Pair2*: {(Vehicle has-state aquaplaning) AND (Light has-color Red)}. These preconditions correspondto a *Goal State*. It is transmitted to the *Planner* that directs the simulation to a state where there is an aquaplaning and a red traffic light. Then, the Virtual Environment, that was developed, instantiates visually this situation. A demonstration video is available in the footnote link[2].

VIII. Experiment and Results

To evaluate our *Scenario Engine*. We conducted a first evaluation without the Virtual Environment. It consists in an online questionnaire to see if the generated situations were perceived as dilemmas by the participants. Among these situations, we included "normal" situations. For this evaluation, the *Scenario Engine* used Knowledge Models that were slightly different from the ones used in the proof of concept. It generated the following pairs of tasks:

- {"*Handle_stop*", "*Handle_aquaplaning*"}
- {"*Handle_red_light*", "*Handle_aquaplaning*"},
- {"*Handle_close_car_behind*", "*Handle_stop*" }
- {"*Handle_close_car_behind*","*Handle_pedestrian*" },
- {"*Handle_close_car_behind"*,"*Handle_red_light*" },
- {"*Handle_aquaplaning*","*Handle_pedestrian*"}.

For each pair we described a little situation (two sentences at most). In addition to these, we added three "normal" situations:

- {"*Handle_red_light*", "*Handle_clear_road*"}
- {"*Handle_no_entry*", "*Follow_passenger_advice*"}
- {"*Handle_green_light*","*Handle_pedestrian*"}

The nine situations were presented randomly to the participants. They had to answer the following questions:

- What would you do in this situation?
- Did you hesitate before making your decision? Why?
- Do you think that there is a solution without negative consequences?
- Do you think that there is a "good" solution?

For this evaluation we had a totasl of 67 participants. Figure 10 shows the results.

[2] https://www.youtube.com/watch?v=Qz80sBjasfU

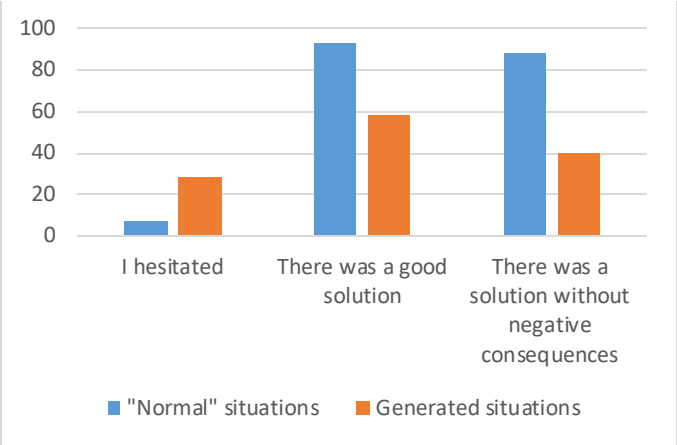

Figure 10 - Evaluation results

In comparison with "normal" situations, we noted that the participants were more hesitant in the situations generated bythe system. In theory, we expected a higher level of hesitation. After analyzing the participants' responses, it showed that this lower level of hesitation was due to certain elements that were not taken into consideration by our system, which made the participants' decision easier and immediate (e.g. "most of the cars today are equipped with technologies that prevent aquaplaning, so I will brake to stop in the red light"). In the generated situations, 60% of the participants said that there was no solution without negative consequences, while for the "normal" situations only 12% stated that. Furthermore, 93% stated that there was a good solution in "normal" situations against 58.25% for the generated situations. Therefore, we can conclude that our system was able to generate situations that, compared to "normal" ones, were more complicated in terms of decision making, were perceived to have negative effects whichever the participant choice and present no good solution. And that corresponds to dilemmas situations.

IX. Discussion and Conclusion

The dynamic generation of training situations is a solution that addresses the problem of the "Authoring Bottleneck". We adopted this approach for the generation of critical situations. In particular, we were interested in generating dilemma situations without having to write them in advance. We identified properties that characterize dilemma situations and formalized them. We proposed a dynamic generation approach that uses these properties and relies on Knowledge Models. This approach has been implemented and showed encouraging results. The work presented in this article concerns only the specification of dilemma as scenario goals (the "Scenario Engine" module). The planning part was not detailed here. It goes without saying that the role of the "Planner" is crucial. In fact, the events temporality is critical because it can jeopardize the dilemma. In the example presented in the proof of concept, the dilemma is compromised if the traffic light turns red after the player passes the light.

We plan now to take into consideration the agent's profile and uncertainties in the Causality Models. This will enable the system to present more personalized dilemmas and to propose the most probable ones. In the future, we also envisage integrating the generation of moral and ethical dilemmas. Currently, we are investigating which values model to use. The Theory of Universal Values of Schwartz [15] seems to be a serious track to consider.

Acknowledgment

The authors want to thank the French National Research Agency (ANR) for the funding of the *MacCoy Critical* project (ANR-14-CE24-0021), and all the members of the consortium.

References

[1] L. S. Vygotsky, *Mind in Society*. Cambridge, Harvard University Press, 1978.

[2] U. Spierling and N. Szilas, "Authoring issues beyond tools," in *Interactive Storytelling*, Springer, 2009, pp. 50–61.

[3] J.-M. Burkhardt, V. Corneloup, C. Garbay, Y. Bourrier, F. Jambon, V. Luengo, A. Job, P. Cabon, A. Benabbou, and D. Lourdeaux, "Simulation and virtual reality-based learning of non-technical skills in driving: critical situations, diagnostic and adaptation," *IFAC-PapersOnLine*, vol. 49, no. 32, pp. 66–71, 2016.

[4] M.-A. Lecomte, "La formation à l'éthique des étudiants en soins infirmiers (Belgique)," *Rech. Soins Infirm.*, vol. 86, no. 3, p. 4, 2006.

[5] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science (80-.).*, vol. 352, no. 6293, 2016.

[6] J. Rickel, S. Marsella, J. Gratch, R. Hill, D. Traum, and W. Swartout, "Toward a new generation of virtual humans for interactive experiences," *IEEE Intell. Syst. Their Appl.*, vol. 17, no. 4, pp. 32–38, 2002.

[7] J. Gratch and S. Marsella, "A Domain-independent Framework for Modeling Emotion," *J. Cogn. Syst. Res.*, vol. 5, no. 4, pp. 296–306, 2004.

[8] P. Foot, "The problem of abortion and the doctrine of double effect," 1967.

[9] P. Valdesolo and D. Desteno, "Manipulations of emotional context shape moral judgment," *Psychol. Sci.*, vol. 17, no. 6, pp. 476–477, 2006.

[10] M. Hauser, F. Cushman, L. Young, R. K. X. Jin, and J. Mikhail, "A dissociation between moral judgments and justications," *Mind Lang.*, vol. 22, no. 1, pp. 1–21, 2007.

[11] C. D. Navarrete, M. M. McDonald, M. L. Mott, and B. Asher, "Virtual morality: emotion and action in a simulated three-dimensional 'trolley problem'.," *Emotion*, vol. 12, no. 2, pp. 364–70, 2012.

[12] A. Skulmowski, A. Bunge, K. Kaspar, and G. Pipa, "Forced-choice decision-making in modified trolley dilemma situations: a virtual reality and eye tracking study," *Front. Behav. Neurosci.*, vol. 8, no. December, p. 426, 2014.

[13] H. Barber, "Generator of Adaptive Dilemma-based Interactive Narratives," no. October, pp. 1–18, 2008.

[14] P. Vallentyne, "Two Types of Moral Dilemmas," *Erkenntnis (1975-)*, vol. 30, pp. 301–318.

[15] S. H. Schwartz, *Les valeurs de base de la personne: Théorie, mesures et applications*, vol. 47, no. 4. 2006.