Scénarisation d'environnements virtuels : vers un équilibre entre contrôle, cohérence et adaptabilité

Camille Barot

Par **Camille BAROT**

Scénarisation d'environnements virtuels : vers un équilibre entre contrôle, cohérence et adaptabilité

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC

Soutenue le 24 février 2014
Spécialité : Technologies de l'Information et des Systèmes

D1615

Thèse
pour l'obtention du grade de

Docteur de l'Université de Technologie de Compiègne
Spécialité : Technologies de l'Information et des Systèmes

Scénarisation d'environnements virtuels.

Vers un équilibre entre contrôle, cohérence et adaptabilité.

par
Camille Barot

Soutenue le 24 février 2014 devant un jury composé de :

M. Ronan CHAMPAGNAT	Maître de Conférences (HDR) Université de la Rochelle *Rapporteur*
M. Stéphane DONIKIAN	Directeur de Recherche (HDR) INRIA Rennes-Bretagne *Rapporteur*
M. Stacy MARSELLA	Professor Northeastern University *Examinateur*
Mme Indira MOUTTAPA-THOUVENIN	Enseignant-Chercheur (HDR) Université de Technologie de Compiègne *Examinatrice*
M. Nicolas SZILAS	Maître d'Enseignement et de Recherche Université de Genève *Examinateur*
M. Dominique LENNE	Professeur Université de Technologie de Compiègne *Directeur*
Mme Domitile LOURDEAUX	Maître de Conférences (HDR) Université de Technologie de Compiègne *Directrice*

heudiasyc cnrs utc Université de Technologie Compiègne PICARDIE LA RÉGION

Résumé

Ces travaux traitent de la **scénarisation d'environnements virtuels**, définie comme la spécification des déroulements possibles ou souhaitables d'une simulation, et la mise en place de mécanismes permettant de contrôler son déroulement effectif de manière dynamique.
Nous visons pour cette scénarisation un ensemble d'objectifs, souvent considérés comme contradictoires : la **liberté** et la **capacité d'action** de l'utilisateur, l'ampleur, le caractère dynamique et l'efficacité du **contrôle** exercé sur le scénario, la **cohérence** des comportements présentés et l'**adaptabilité** du système, nécessaire pour la variabilité des scénarios.
Nous proposons **SELDON**, un modèle basé sur le contrôle centralisé et indirect d'une simulation émergente à partir de modèles du contenu scénaristique. L'environnement est peuplé de personnages virtuels autonomes et l'utilisateur y est libre de ses actions. La scénarisation est réalisée en deux étapes : des objectifs dynamiques sont déterminés à partir de l'activité de l'utilisateur, puis un scénario est généré en fonction de ces objectifs et exécuté au travers d'ajustements sur la simulation. Le moteur **DIRECTOR** permet de générer et de réaliser ce scénario. DIRECTOR utilise les modèles qui sous-tendent la simulation pour prédire son évolution, et guide cette évolution au travers d'un ensemble d'ajustements indirects, qui influencent les réactions des systèmes techniques et les prises de décision des personnages. Il utilise un **moteur de planification** pour générer des scénarios composés d'étapes de **prédiction**, dont il suit la réalisation dans la simulation, et d'**ajustements**, qu'il déclenche. Les objectifs scénaristiques pris en compte sont à la fois dynamiques, au travers de situations prescrites et proscrites et de contraintes sur les propriétés globales du scénario, et statiques, sous la forme d'espaces de scénario. Le contenu scénaristique et les objectifs scénaristiques statiques sont représentés à l'aide de l'ensemble de langages que nous avons proposé : **DOMAIN-DL**, **ACTIVITY-DL** et **CAUSALITY-DL**. Ces contributions permettent d'allier la réactivité et la variabilité permises par des simulations émergentes et la pertinence des scénarios offerte par les systèmes basés sur de la planification.

Abstract

Orchestration of virtual environments:
Balancing control, coherence and adaptability.

This work addresses the **orchestration of virtual environments**, defined as the specification of possible or wanted unfoldings of events in a simulation and the implementation of control mechanisms over the actual unfolding.
We aim at a set of objectives often considered contradictory: user **freedom**/**agency**, range, dynamicity and efficiency of **control** over the scenario, behavioural **coherence**, and system **adaptability**, which is essential for scenario variability.
We propose **SELDON**, a model based on centralised and indirect control of an emergent simulation. Scenario content is represented by domain and activity models in the **DOMAIN-DL** and **ACTIVITY-DL** languages. The virtual environement is populated by autonomous characters, and the user's choices of actions are unconstrained. Orchestration is carried out in two steps: first, a set of dynamic objectives is determined from the user's profile and activity, then a scenario is generated from these objectives and executed through adjustements on the simulation.
The **DIRECTOR** engine performs scenario generation and execution. DIRECTOR uses the simulation's models to predict its evolution and guide it through a set of indirect adjustements that influence technical systems' reactions and characters' decisions. It uses **planning** to generate scenarios made up of **prediction** steps that DIRECTOR monitors in the simulation and **adjustements** that it triggers in the virtual environment. Scenario objectives can be defined dynamically through prescribed/proscribed situations and constraints on global scenario properties or statically as a space of scenarios of interest represented in the **CAUSALITY-DL** language.

Remerciements

Je tiens avant tout à exprimer ma profonde gratitude à Domitile Lourdeaux, qui m'a encadrée, conseillée et soutenue tout au long de ce doctorat. Merci, Domitile, d'avoir toujours cru en moi et de m'avoir encouragée à viser plus haut. Merci aussi de m'avoir permis de trouver mes marques et de m'intégrer dans une approche existante sans jamais m'imposer un point de vue, de m'avoir laissé poursuivre mes idées même quand tu n'y croyais pas au départ. Merci enfin de m'avoir montré que l'on pouvait vraiment s'éclater en faisant de la recherche. Tu as fait de ton équipe une vraie famille, avec son arbre généalogique accroché au mur de ton bureau, et je suis heureuse d'en faire partie. Je suis fière du travail accompli à tes côtés.

Je remercie également Dominique Lenne d'avoir co-encadré cette thèse. Merci de m'avoir fait confiance, et de m'avoir poussée aussi, durant la thèse certes mais également bien avant puisque c'est toi qui, dès IA01, m'a incitée à envisager la voie de la recherche.

Je souhaite remercier Ronan Champagnat et Stéphane Donikian d'avoir accepté de rapporter ma thèse, et je remercie Nicolas Szilas d'avoir accepté d'en être examinateur. I thank (in English!) Stacy Marsella, for not only accepting to suffer the slings and arrows of visioconference to attend my defense, but also for the faith he put in me during our short time working together. Je remercie de tout cœur Indira Mouttapa Thouvenin, qui a accepté de présider mon jury de thèse, mais m'a surtout soutenue et encouragée depuis plusieurs années. Merci à tous pour vos critiques, vos remarques et vos questions qui ont fait de ma soutenance un moment d'échanges passionnants.

Cette thèse n'aurait pas pu voir le jour sans le soutien de la région Picardie et du FEDER au projet ARAKIS, et je les en remercie. De même, je remercie toutes les personnes impliquées dans ce projet, mais aussi dans les projets V3S et NIKITA auxquels j'ai pu participer, en particulier les groupes du CEA, du LATI et d'Emissive. Merci donc à Emmanuel Guerriero, Fabien Barati et Antoine Ferrieux avec qui j'ai été ravie de travailler avant comme pendant la thèse. Un grand merci également à Jean-Marie Burkhardt, avec qui les échanges ont été particulièrement enrichissants. Enfin, je suis très reconnaissante à Fabrice Camus d'être venu avec moi affronter les dangers des dépôts pétroliers pour réaliser les analyses terrain, et de m'avoir encouragée sans relâche pendant toute la durée de cette thèse.

Mes pensées vont à l'équipe HUMANS, et à tous ceux qui ont pu graviter autour de celle-ci à un moment où à un autre : Margot Lhommet, Pierre-Yves Gicquel, Vincent Lanquepin, Kahina Amokrane, Antoine Vincent, Dorine Dufour, Hazaël Jones, Jocelyn Thiery, Lydie Edward, Lucile Callebert, Juliette Lemaitre, Loïc Fricoteaux, Jérôme Olive, Alistair Jones, Kevin Wagrez, Thibaut Arribe... mais aussi la bande de Reviatech — Romain Lelong, Mehdi Sbaouni et Morgan Fraslin — et celle des déjeuners à l'IMI — Benjamin Diemert et Thomas Bottini. On a vécu de grands moments, et vous avez été bien plus que des collègues à mes yeux. J'espère que les traditions des journées à thèmes du mardi et des bières au Shamrock qui font rater le train perdureront, et que de nouvelles viendront s'y ajouter.

Kevin Carpentier, mon acolyte de toujours, mérite son paragraphe rien qu'à lui : tu as été un binôme fantastique durant toutes ces années, et ça va beaucoup me manquer de travailler avec toi. J'espère qu'on y remédiera très vite et qu'on finira par ouvrir ensemble un grand parc d'attractions autour du storytelling, de la génération procédurale et de la synesthésie.

Je remercie également le laboratoire Heudiasyc et l'équipe ICI pour leur accueil, ainsi que les étudiants de l'UTC que j'ai pu encadrer autour de différents projets pour l'équipe HUMANS et qui ont été d'une aide précieuse : Barthélémy Arribe, Yacine Badiss, Clément Boissière, Marion Cayla, Jonathan Denonain, Pierre-Henri Fricot, Thibaud Huet, Ronan Kervella, Pierre-Alexandre Kofron,

Julien Mazars, Aloïs Nolin, Vincent Palancher, Thierry Tang... sans oublier Diane Wakim bien sûr !

Je voudrais aussi saluer le travail du personnel de l'UTC, en particulier Nathalie Alexandre et Marion Kaczkowski qui m'ont sauvé la mise plus d'une fois.

Merci à mes amis, la MDB et tous les autres, d'avoir toujours su me changer les idées et me faire relativiser. Merci en particulier à Elise, qui est mon rayon de soleil, et qui maintenant va se sentir obligée de lire cette thèse en entier.

Enfin, merci à mes parents, mon frère, ma sœur, ainsi qu'à toute ma famille, de m'avoir supportée (dans les deux sens du terme !) dans des moments qui ont été très difficiles pour moi comme pour eux. C'est grâce à leur soutien inconditionnel que je peux présenter aujourd'hui ce travail.

Table des matières

Chapitre 1

Introduction

Sommaire

Les travaux présentés dans cette thèse traitent de la scénarisation d'environnements virtuels. Leur objectif est de fournir un modèle pour la scénarisation dynamique de ces environnements, qui permette de générer et de mettre en place des scénarios pertinents, en particulier dans le cas de simulations complexes, le tout en conservant la cohérence des comportements présentés.

1.1 Scénarisation et environnements virtuels

Ces travaux s'inscrivent dans le contexte des environnements virtuels : des systèmes interactifs permettant à un ou plusieurs utilisateurs d'interagir avec un monde artificiel, généralement en trois dimensions, par le biais d'un ensemble de techniques informatiques couvrant une ou plusieurs modalités sensorielles [Burkhardt et al., 2003]. Plutôt que de considérer ces environnements sous l'angle des périphériques utilisés et des différents degrés d'interaction et d'immersion induits sur le plan sensori-moteur, nous nous intéresserons aux événements ayant lieu dans le monde artificiel et à l'activité de l'utilisateur en termes cognitifs, en suivant la définition de la réalité virtuelle donnée par [Fuchs et al., 2006] :

> “ La finalité de la réalité virtuelle est de permettre à une personne (ou à plusieurs) une activité sensori-motrice et cognitive dans un monde artificiel, créé numériquement, qui peut être imaginaire, symbolique ou une simulation de certains aspects du monde réel. ”

La scénarisation est un processus qui vise à spécifier les événements devant se produire dans un cadre donné, qui peut être celui d'un film, d'un jeu vidéo, d'une session d'apprentissage, etc. Or, selon le cadre considéré, le terme de scénarisation, ainsi que la notion de scénario lui-même, ne revêtent pas le même sens.

Dans le domaine cinématographique, un scénario est un document écrit décrivant les différentes scènes d'un film. On distingue deux types de scénarios : le script, ou screenplay, décrit l'enchainement des événements (actions, dialogues...) — *“what to shoot"*, tandis que le découpage technique, ou shooting script, inclut également des indications sur la manière de mettre en scène et de porter ces événements à l'écran (découpage en séquences et plans, points de vue...) — *“how to shoot it"* [Van Rijsselbergen et al., 2009]. La scénarisation désigne alors le processus d'écriture de ce scénario.

De même, dans le domaine de l'apprentissage, la scénarisation est souvent considérée comme une activité de modélisation (voir par exemple [Ouraiba, 2012]). Elle consiste alors en la spécification, de manière prescriptive, du déroulement futur d'une activité pédagogique, en décrivant à la fois les intentions des enseignants et les situations d'apprentissage prévues pour leurs enseignements à l'aide d'un langage de scénarisation pédagogique [Abedmouleh, 2012].

Pour Pernin et Lejeune [Pernin and Lejeune, 2004], la spécification du scénario n'est au contraire que la première étape de la scénarisation. Ils distinguent ainsi quatre phases dans le cycle de vie d'un scénario d'apprentissage :

- la conception, qui consiste à spécifier l'activité prescrite dans le langage de scénarisation ;
- l'opérationnalisation, durant laquelle ces spécifications sont transformées en composants exécutables sur des plateformes informatiques ;
- l'exploitation, qui consiste à “jouer" le scénario, et durant laquelle les acteurs (apprenants et tuteurs) doivent pouvoir observer et ajuster les situations mises en place.
- le retour d'usage, où les résultats obtenus lors de la phase d'exploitation sont évalués

Si de nombreux travaux sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH) et Environnements Virtuels pour l'Apprentissage Humain (EVAH) se concentrent sur la phase de conception des scénarios et sur les langages de scénarisation [Martel et al., 2007] [Laforcade et al., 2007], la phase d'exploitation revêt une importance capitale : à partir du moment où on utilise l'outil

informatique, l'exécution, la supervision et la régulation du scénario ne sont plus intégralement le fait du formateur humain, mais relèvent en partie de l'outil lui-même [1].
Ainsi, dès lors que l'on se place dans un contexte interactif, la scénarisation ne consiste plus uniquement en la modélisation d'un scénario, mais également en la mise en place des mécanismes nécessaires à la réalisation de ce scénario. Selon les travaux, cette partie dynamique du processus de scénarisation sera désignée par les termes de pilotage [Delmas, 2009], drama management [Mateas and Stern, 2005], directorial control [Si et al., 2009], ou encore story mediation [Magerko, 2007].
C'est autour de cette problématique que se place le domaine de la narration interactive (interactive storytelling). La narration interactive permet aux utilisateurs de participer activement à une histoire, soit de manière intradiégétique, en incarnant l'un des personnages, soit de manière extradiégétique, en exerçant un contrôle sur le déroulement des événements [Si, 2010]. Les travaux effectués dans le domaine de la narration interactive visent à trouver un équilibre entre l'intérêt narratif et le respect des intentions de l'auteur d'un côté, et la liberté d'action du ou des utilisateurs de l'autre [Magerko, 2005].
Lorsqu'il s'agit de concevoir un environnement virtuel ou un système de narration interactive, il existe deux approches : la scénarisation peut être considérée comme partie intégrante du processus de conception, ou bien elle peut être vue comme une étape supplémentaire de cadrage d'un environnement existant.
Dans le premier cas, la scénarisation va consister à définir l'ensemble des objets de l'environnement virtuel, leurs règles de fonctionnement, les actions possibles de la part de l'utilisateur et des personnages virtuels, le comportement de ces derniers, etc. Les scénarios peuvent être définis de manière plus ou moins explicite, allant du script exhaustif à la modélisation des comportements individuels, les scénarios effectifs émergeant alors des interactions entre le ou les utilisateurs et le monde virtuel. L'environnement virtuel scénarisé est ici vu comme un tout. C'est le cas par exemple dans [Mateas and Stern, 2005] ou [Cavazza et al., 2002].
Dans le deuxième cas, la scénarisation va consister à réduire l'espace des scénarios possibles, en sélectionnant, ordonnant ou priorisant des sous-ensembles d'événements en vue d'une utilisation particulière de l'environnement virtuel. Ainsi, pour N. Mollet, "la scénarisation consiste à utiliser l'environnement et agencer les interactions afin de réaliser une tâche particulière" [Mollet, 2005].
La modélisation du scénario est alors séparée en deux parties : la spécification du contenu scénaristique d'un côté, et celle des objectifs scénaristiques de l'autre.
Si l'on reprend la métaphore du jeu de rôle (non numérique), souvent utilisée en narration interactive (voir par exemple [Aylett et al., 2011], [Delmas, 2009]), on peut identifier trois composantes principales au sein du jeu avec lequel les joueurs interagissent :

- le corpus de règles qui décrit l'univers et les mécanismes du jeu,
- la trame préparée en amont par le meneur de jeu (game master), qui peut être plus ou moins détaillée et plus ou moins dirigiste,
- les interventions du meneur de jeu, qui décrit la situation initiale, interprète et applique les règles du jeu, fait intervenir des personnages non-joueurs, déclenche des événements et met en scène le tout.

L'activité du meneur de jeu est assimilable au processus de pilotage de l'environnement virtuel, tandis que le corpus de règles correspond au contenu scénaristique et la trame aux objectifs scénaristiques. On constate ainsi que, pour un même contenu, de nombreux scénarios différents pourront être créés en fonction des objectifs définis, et que la spécification du contenu et celle des objectifs scénaristiques ne sont pas forcément réalisées par les mêmes personnes (ainsi le créateur du jeu de

1. Il est nécessaire de distinguer ici la scénarisation des activités pédagogiques et celle des situations d'apprentissage (parfois appelée orchestration [Luengo, 2009]). La première va situer l'utilisation de l'environnement virtuel comme outil de formation dans le cadre plus large d'un scénario pédagogique, au même titre que d'autres activités pédagogiques comme des cours magistraux. Le passage d'une activité à une autre peut alors relever de la responsabilité du formateur, ou être déclenché automatiquement par un système informatique. La seconde va poser des contraintes sur le déroulement de l'activité même de l'utilisation de l'environnement virtuel, contraintes qui devront être appliquées sur la simulation par un système de scénarisation, afin que le scénario se déroule quelles que soient les actions de l'apprenant.

rôle définit le système de règles, et peut également fournir des trames pré-écrites, mais le meneur de jeu peut choisir d'utiliser une trame de sa propre invention).
On retrouve cette même approche dans de nombreux environnements virtuels de type simulateurs, utilisés pour la formation : ces environnements sont construits autour d'un noyau générique, qui contient les règles de fonctionnement de la simulation (ex : le fonctionnement d'une automobile, le code de la route, etc.) auquel vient s'ajouter une couche d'instanciation pour un objectif pédagogique donné (ex : "conduite sur autoroute avec un trafic dense").
Les objectifs scénaristiques peuvent ainsi être spécifiés de manière très large en donnant simplement les conditions initiales d'une simulation. Le terme de "scénario" est alors utilisé pour désigner ces conditions initiales, par exemple la disposition de l'environnement virtuel (settings) dans [Hullett and Mateas, 2009]. Le "scénario" en tant qu'objectif scénaristique peut, selon les travaux, être modélisé à différents niveaux de granularité, et de manière plus ou moins explicite : contraintes sur les conditions initiales [Hullett and Mateas, 2009], ensemble de séquences d'événements exemplaires [Si, 2010], contraintes d'ordonnancement sur des événements clés [Porteous and Cavazza, 2009], contraintes de haut niveau sur des fonctions d'évaluation du scénario [Weyhrauch, 1997], etc. Certains parlent au contraire du "scénario" d'un environnement virtuel pour désigner le domaine d'application [Szilas, 2007]. D'autres utilisent ce même terme pour parler d'une séquence d'événements particulière, instanciée lors d'une session d'interaction [Champagnat et al., 2005] — il peut alors s'agir d'une séquence prescrite ou effective. C'est cette acception que nous utiliserons dans le reste de ce mémoire, et nous parlerons alors d'"espace de scénarios" lorsqu'il s'agira de se référer à un ensemble de possibles.
Nous adopterons ainsi dans cette thèse les définitions suivantes :

Scénarisation

La scénarisation est un processus comprenant à la fois la spécification du ou des déroulements possibles ou souhaitables de la simulation, et le contrôle (exécution et/ou suivi et correction) du déroulement des événements en temps-réel.

Système de scénarisation

Un système de scénarisation est composé d'un ou plusieurs langages de scénarisation, permettant de modéliser le contenu scénaristique et/ou les objectifs scénaristiques, et d'un moteur de scénarisation permettant de gérer de manière dynamique la réalisation du scénario.

1.2 Motivations

La scénarisation d'environnements virtuels a des applications dans de nombreux domaines, parmi lesquels on peut citer la formation [Burkhardt et al., 2006], l'aide à la décision [Bourdot et al., 2006], la thérapie [Klinger et al., 2006] et le divertissement, notamment via le domaine de la narration interactive [Bates, 1992]. Dans le cadre de ce mémoire, nous nous intéresserons surtout aux deux premiers. Les Environnements Virtuels pour l'Apprentissage Humain (EVAH) constituent en effet une des applications majeures de la réalité virtuelle, se démocratisant de plus en plus depuis les premiers développements remontant aux années 90. Les avantages de la réalité virtuelle pour la formation sont nombreux [Lourdeaux, 2001] : non seulement l'utilisation d'un environnement virtuel permet de se détacher des contraintes pouvant exister dans les formations en conditions réelles (dangerosité, contraintes d'accessibilité au site ou au matériel, coût, etc.), mais elle permet également d'enrichir ces situations à des fins pédagogiques, via par exemple des fonctions de rejeu ou de segmentation de l'apprentissage.
Ces environnements concernent à présent de nombreux domaines d'application, que ce soit au niveau

de la formation professionnelle — formation au geste technique [Da Dalto, 2004], aux procédures de maintenance [Gerbaud, 2008] ou de conduite de machineries [Crison et al., 2005] [Rickel and Johnson, 1998], formation à la maîtrise des risques [Fabre et al., 2006] [Barot et al., 2013] [Querrec and Chevaillier, 2001], etc. — ou bien de l'éducation [Mikropoulos and Natsis, 2011] — compréhension de concepts scientifiques [Dede et al., 1996] [Windschitl and Winn, 2000], travaux pratiques virtuels [Baudouin, 2007], apprentissage des langues [Schlemminger, 2013], etc. Ils peuvent être utilisés dans différentes configurations techniques et avec différentes stratégies pédagogiques : présence ou non d'un formateur, environnement mono ou multi-apprenant, apprenant spectateur ou acteur de l'interaction avec l'environnement virtuel, etc. [Burkhardt et al., 2005].
Les environnements virtuels sont également utilisés pour des applications d'aide à la décision, que ce soit dans l'industrie [Bourdot et al., 2006] ou dans le domaine médical [Rizzo et al., 2012], mais aussi pour des applications de planification urbaine [Roupé, 2013], de gestion des flux d'immigration [Hedge, 2011], etc. La simulation informatique d'un système socio-technique peut en effet permettre au décideur d'appréhender rapidement les conséquences induites par ses choix, lui donnant une meilleure visibilité, en particulier lorsque les décisions doivent être prises dans l'urgence et dans un contexte flou [Castagna et al., 2001]. De tels simulateurs sont aussi utilisés en amont comme outils d'aide à la décision pour la conception de systèmes [Robotham and Shao, 2012], ou encore pour la maîtrise des risques, afin de permettre de détecter les facteurs d'accidents possibles [Gounelle et al., 2007].
Pour de nombreux environnements, dont la visée est de simuler un système technique dans une situation précise et cadrée, un faible niveau de scénarisation est suffisant. Les concepteurs définissent dans ce cas les règles d'évolution des systèmes simulés, sans chercher à exercer de contrôle supplémentaire sur le déroulement des événements. Ces environnements peuvent servir de support d'apprentissage — l'apprenant expérimentant avec la simulation à la manière d'un chercheur scientifique [De Vries and Baille, 2006] — ou d'entrainement, comme c'est le cas pour de nombreux simulateurs de conduite dits "pleine échelle" [Joab et al., 2006].
D'autre part, certains environnements font l'objet d'une scénarisation pédagogique réalisée au travers de rétroactions extradiégétiques, c'est à dire qui ne correspondent pas à des événements du domaine simulé (elles se situent ainsi en dehors de la diégèse). Ces rétroactions peuvent apporter des modifications à l'environnement virtuel au sein d'une même situation d'apprentissage, sous la forme d'assistances [Lourdeaux, 2001] : enrichissement de l'environnement via l'ajout de symboles visuels ou sonores, restriction des déplacements ou manipulations de l'utilisateur, etc. Elles peuvent également permettre de naviguer dans le scénario pédagogique en passant d'une situation d'apprentissage à une autre [Luengo, 2009] ; la simulation est alors arrêtée et relancée à partir d'une situation différente.
Cependant, dans le domaine de la formation et de l'aide à la décision, peu de travaux s'intéressent à la scénarisation des environnements virtuels sur le plan intradiégétique. Ce type de scénarisation consiste à contrôler le déroulement des événements au sein même de la simulation, sans couper l'immersion de l'utilisateur dans l'environnement virtuel. La visée ici n'est pas de remplacer les systèmes de scénarisation pédagogique qui ont fait leurs preuves pour la formation initiale et l'entrainement d'apprenants, mais de se positionner de manière complémentaire, en adressant d'autres besoins et d'autres types de formation.
Ainsi, dans le cadre de la formation continue d'apprenants experts, ces derniers doivent être confrontés à des scénarios particuliers, plus rares, mettant en jeu des situations de travail dégradées. La scénarisation peut alors permettre d'amener à la réalisation d'un événement précis, comme le déclenchement d'un accident donné pour une formation à la maîtrise des risques. Pour des applications d'analyse de risque, cela permettrait ainsi de rejouer des scénarios accidentels, ou de déterminer les facteurs de risque d'un accident donné sans avoir besoin de procéder à des ajustements progressifs des conditions initiales de la simulation (voir par exemple [Bosse and Mogles, 2013]).
De même, alors que les situations de travail se complexifient, il devient nécessaire de former non seulement aux compétences techniques, mais également aux compétences non-techniques, comme le travail en groupe ou la gestion du stress. Les environnements virtuels seront alors peuplés de per-

sonnages virtuels, dont le comportement devra lui-aussi être pris en compte par le système de scénarisation : les personnages pourraient ainsi tenter de minimiser ou de maximiser les risques liés à une procédure collaborative, aider l'apprenant quand il est en difficulté en lui donnant des conseils ou en réalisant des actions à sa place, ou au contraire le mettre en situation de stress.
Cette scénarisation adaptative permettrait ainsi de contrôler le niveau de tension et de difficulté de manière à ce qu'ils restent adaptés au profil et à l'activité de l'apprenant. Elle permettrait également de motiver ce dernier par le biais d'une mise en scène, en favorisant son implication émotionnelle dans une histoire, à la manière de la narration interactive. Il serait alors possible de transposer les jeux de rôles réalisés dans le cadre des formations sur des personnages de l'environnement virtuel, évitant ainsi de devoir faire intervenir plusieurs formateurs.

1.3 Objectifs

De nos motivations concernant la scénarisation d'environnements virtuels destinés à la formation ou à l'aide à la décision découlent un certain nombre d'objectifs pour notre système de scénarisation. Ces objectifs touchent à la fois aux différents degrés de liberté pour l'utilisation et le pilotage de l'environnement virtuel, au contenu et à la qualité des scénarios produits, et aux possibilités offertes pour la phase de conception de ces environnements virtuels scénarisés.

1.3.1 Déroulement des scénarios

Le premier groupe d'objectifs concerne les propriétés dynamiques de notre système de scénarisation, c'est à dire à la fois l'utilisation qui sera faite par l'apprenant de l'environnement virtuel en temps-réel, et la manière dont le système de scénarisation va pouvoir diriger le scénario de l'environnement virtuel (quoi, quand et comment).
Nous déterminons ainsi quatre objectifs :

- liberté d'action de l'utilisateur,
- degrés de contrôle du scénario,
- caractère dynamique du contrôle,
- résilience.

Liberté d'action de l'utilisateur

Pour promouvoir un apprentissage par essai-erreur, ou pour permettre de tester différents cas dans un cadre d'aide à la décision, il est nécessaire de laisser à l'utilisateur une certaine liberté d'action. Par utilisateur, nous désignons ici la personne qui est immergée dans l'environnement virtuel et joue le rôle de l'un des personnages de l'environnement — dans le cas d'un environnement virtuel pour la formation, nous considèrerons qu'il s'agit de l'apprenant — et non les hypothétiques personnes qui pourraient exercer un contrôle extérieur sur l'environnement en temps réel — par exemple, un formateur.
Ainsi, laisser à l'utilisateur une certaine latitude dans le choix de ses actions peut lui permettre d'expérimenter et de comparer les conséquences des différents choix qui s'offrent à lui. Il faut cependant distinguer le sentiment de liberté d'action que peut avoir l'utilisateur et sa capacité d'action réelle, c'est à dire l'impact réel de ses décisions sur le déroulement des événements (*user agency*) [Thue et al., 2010]. Si la liberté d'action perçue permet d'intéresser l'utilisateur en répondant à ses désirs en termes d'interaction [Wardrip-Fruin et al., 2009], seule une réelle capacité d'action peut lui permettre d'adopter une posture expérimentale vis à vis de l'environnement virtuel. De plus, la capacité d'action affecte également l'implication de l'utilisateur dans l'environnement virtuel [Si, 2010].

Possibilités de contrôle du scénario

En plus de la capacité d'action de l'utilisateur sur le scénario (*user agency*), il est nécessaire de considérer également la capacité d'action du système de scénarisation lui-même (*global agency*).
Ainsi, parmi les objectifs scénaristiques qui vont pouvoir être pris en compte par le système, il faut tout d'abord permettre de spécifier des situations à atteindre. Par situation, nous entendons ici une combinaison particulière d'états de l'environnement virtuel, pouvant être décrite à plus ou moins haut niveau, par exemple "Il y a une fuite au niveau de la vanne n°32" ou "Il y a une anomalie au niveau d'une vanne". Pour les environnements virtuels pour l'apprentissage se situant dans un cadre d'apprentissage situé, il est considéré que la situation dans laquelle se développent les apprentissages est primordiale et que les apprentissages doivent être contextualisés. La situation est une composante intégrale de la connaissance qui se développe [Rogalski, 2004]. On va donc chercher à placer l'apprenant dans une situation d'apprentissage donnée, afin de mettre en jeu des connaissances et des compétences, et de générer ces apprentissages.
A l'inverse, les situations à *ne pas* atteindre doivent également pouvoir être spécifiées. Cela est notamment utile dans le cas où un apprenant ne possède pas encore les ressources nécessaires pour gérer une situation, ou au contraire s'il a déjà rencontré cette situation par le passé.
Enfin, il s'agit de ne pas seulement s'intéresser aux situations ponctuelles mais aussi aux propriétés plus globales du scénario. En effet, une situation d'apprentissage peut être plus large qu'une situation ponctuelle dans l'environnement virtuel et inclure une dimension temporelle : augmentation du rythme de travail ou présence d'un collaborateur récalcitrant, par exemple. Contrôler l'agencement des situations, le rythme ou encore la complexité globale du scénario permettrait ainsi de pouvoir gérer une augmentation de la difficulté et de faire disparaître progressivement les assistances offertes par l'environnement virtuel [Burkhardt, 2010].

Caractère dynamique du contrôle

En parallèle du *quoi* contrôler, il faut aussi s'intéresser au *quand*, car si la définition des objectifs scénaristiques en amont permet déjà de spécifier un scénario, il peut être utile dans de nombreux cas de re-définir ces objectifs de manière dynamique, afin d'offrir un scénario qui soit pertinent par rapport au profil et à l'activité de l'utilisateur de l'environnement virtuel.
On veut ainsi pouvoir adapter les objectifs scénaristiques d'une session d'utilisation à l'autre afin de prendre en compte ce qui a été réalisé précédemment par l'utilisateur : introduire de nouvelles situations, augmenter la difficulté, éviter de rejouer des scénarios déjà vus, etc.
De plus, il peut s'avérer intéressant dans certain cas d'aller plus loin, en proposant une adaptation dynamique du scénario au cours d'une même session d'utilisation de l'environnement virtuel. En effet, de même que l'environnement est modifié du fait des actions de l'apprenant, ce dernier évolue lui aussi au fur et à mesure de ses interactions avec la simulation. L'analyse de son activité en temps-réel permettant d'inférer sur ses connaissances et compétences, ces informations peuvent être utilisées pour déterminer les nouvelles situations auxquelles il doit être confronté. Le système de scénarisation doit ainsi être capable de prendre en compte de nouveaux objectifs scénaristiques de manière dynamique.

Résilience

Par résilience, nous désignons la capacité du système de scénarisation à réaliser un scénario possédant les propriétés souhaitées en dépit des perturbations extérieures. En effet, l'utilisateur étant libre d'agir dans l'environnement virtuel, ses actions peuvent aller à l'encontre du scénario prévu : distraction des personnages virtuels, modification des états des objets, etc. Le système de scénarisation doit être capable de gérer ces déviations, soit en limitant les conséquences de l'action de l'utilisateur sur le scénario (*intervention*), soit en modifiant le scénario afin de les prendre en compte

(*accommodation*) [Riedl et al., 2003].
Cette propriété est particulièrement importante dans le cas où les objectifs du système de scénarisation et ceux de l'utilisateur sont opposés. Dans les environnements virtuels pour la formation à la maîtrise des risques, par exemple, le système de scénarisation va chercher à confronter l'apprenant à une situation accidentelle, tandis que ce dernier cherchera à l'éviter.

1.3.2 Contenu des scénarios

Le second groupe d'objectifs a trait aux propriétés des scénarios générés via l'interaction de l'utilisateur et du système de scénarisation lors de sessions d'utilisation de l'environnement virtuel.
Nous déterminons ainsi trois objectifs pour ces scénarios :

- validité écologique,
- explicabilité,
- variabilité.

Validité écologique

Pour qu'il puisse y avoir un transfert d'apprentissage entre l'environnement virtuel et la situation de travail réelle, ou que les conclusions tirées de la simulation pour l'aide à la décision soient transposables, il faut que la situation simulée corresponde par certains aspects à la situation réelle.
La reproduction parfaitement fidèle de la réalité n'est pas nécessairement un objectif en soi : dans certains cas, la situation d'apprentissage autorise une déformation ou une exagération de la réalité pour faire mieux comprendre la complexité de la situation [Lourdeaux, 2001]. Au contraire, selon la visée de la formation ou le type de système simulé, les environnements virtuels auront différents besoins en termes de réalisme. Pour des environnements virtuels destinés à la formation au geste technique, par exemple, il sera nécessaire de veiller à la *fidélité perceptive* de la simulation, c'est à dire à la création d'une expérience perceptive qui serait crédible si elle était vécue dans le monde réel [Burkhardt et al., 2003]. A l'inverse, lorsqu'il s'agit de former à des compétences de plus haut niveau (apprentissage de procédure, formation à la réaction dans des situations d'urgence, etc.), on va privilégier au réalisme visuel la *fidélité psychologique* de la simulation, c'est à dire la proportion dans laquelle la tâche simulée engendre une activité et des processus psychologiques identiques à ceux de la tâche réelle [Burkhardt et al., 2003].
Plutôt que de réalisme, on va préférer parler de validité écologique des situations. La notion de validité écologique est issue du domaine de la psychologie de la perception, et désigne la proximité entre une situation expérimentale et la situation réelle à laquelle elle fait référence. Les situations écologiquement valides sont traditionnellement mises en opposition avec les situations artificielles très contrôlées et contraintes désignées comme "laboratoires" [Loomis et al., 1999].
Puisque nous nous intéressons à des formations à des compétences de haut niveau, incluant des aspects facteurs humain, nous viserons donc la génération de scénarios rendant compte de la complexité des comportements humains dans les situations de terrain, et induisant de la part des utilisateurs des comportements équivalents à ceux rencontrés en situation réelle.

Explicabilité

En plus de correspondre par certains aspects à la situation réelle, ces scénarios doivent également être explicables. Il s'agit ici de pouvoir expliquer non seulement les chaînes de causalité (conséquences des actions, chaînes accidentelles, etc.), mais aussi les comportements individuels des personnages.
Sans aller toutefois jusqu'à rechercher la modélisation des processus décisionnels humains à la manière cognitiviste, nous visons à intégrer des aspects facteurs humains dans la modélisation comportementale des personnages virtuels, et à assurer la cohérence de leurs comportements par rapport

à ce modèle. Il a en effet été montré dans [Si et al., 2010] que les contradictions dans les motivations des personnages virtuels ont un impact négatif sur la compréhension qu'a l'utilisateur de ce qui se passe dans l'environnement virtuel.
Cette explicabilité est donc nécessaire pour que l'utilisateur puisse comprendre le déroulement des événements, ainsi que pour produire des traces et tirer des conclusions dans le cas de systèmes d'aide à la décision.
De même, pour que l'utilisateur ressente un sentiment de présence dans l'environnement virtuel, il doit pouvoir se former un modèle mental du monde, et pouvoir anticiper les résultats de ses actions ainsi que de celles des personnages virtuels : l'environnement doit alors réagir de manière constante et prévisible [Slater and Usoh, 1993]
La notion d'explicabilité est très liée à celle de cohérence, dans le sens de la cohérence perçue par un utilisateur. [Riedl et al., 2003] définit la cohérence d'un scénario comme la capacité par un utilisateur de comprendre les relations entre les événements du scénario, à la fois au niveau des événements du monde de la simulation, et au niveau des événements qui lui sont directement présentés.

Variabilité

Notre objectif est de cibler de la formation à des situations de travail complexes ou de la formation continue, plutôt que de l'entraînement à un geste technique ou à une procédure limitée. Dans cette optique, il faut que le système de scénarisation puisse générer un panel varié de scénarios, allant des scénarios les plus courants où le système fonctionne en mode nominal et les personnages virtuels réalisent la procédure prescrite, aux scénarios les plus dégradés et les plus rares.
Cette variabilité peut découler d'un système permettant l'émergence de scénarios inédits, ce qui est particulièrement intéressant pour des systèmes d'aide à la décision en maîtrise des risques, où l'on va utiliser la simulation pour juger si des situations accidentelles peuvent émerger des règles de fonctionnement du système technique et des procédures associées.

1.3.3 Conception des scénarios

Enfin, le troisième groupe d'objectifs concerne le processus de conception des scénarios, c'est à dire la modélisation du contenu scénaristique faite en amont par des auteurs humains. On va chercher ici à limiter l'effort nécessaire à la conception (*authoring*) de ces environnements.
Nous déterminons ici trois objectifs principaux :

- passage à l'échelle,
- maintenabilité,
- réutilisabilité.

Passage à l'échelle

La création du contenu scénaristique est aujourd'hui l'un des principaux verrous identifiés dans le domaine de la narration interactive [Spierling and Szilas, 2009]. La création du système le plus connu à l'heure actuelle, Façade, a par exemple nécessité le travail à plein temps de deux ingénieurs pendant 5 ans, dont 3 ont été passés uniquement sur la création du contenu scénaristique, et ce pour une expérience interactive de 20 minutes environ à chaque session [Mateas and Stern, 2005].
Ce problème se fait d'autant plus ressentir lorsqu'il s'agit d'opérer un passage à l'échelle, c'est à dire d'augmenter fortement la taille, la complexité et la variabilité des scénarios couverts par l'environnement virtuel. Si l'on souhaite que le système de scénarisation puisse offrir une grande variabilité en termes de scénarios générés, il faut qu'il soit conçu de manière à pouvoir augmenter le nombre de scénarios possibles sans pour autant augmenter exponentiellement la taille du contenu scénaristique nécessaire à la génération de ces scénarios.

Maintenabilité

Dans les systèmes de narration interactive, la personne qui crée le contenu scénaristique et celle qui crée le moteur de scénarisation sont souvent en réalité la même personne [Spierling and Szilas, 2009]. Dans le cas des environnements virtuels pour la formation ou l'aide à la décision, ce constat pose problème, notamment pour ce qui est de la maintenabilité de l'environnement.
En effet, si le système technique change, ou si les procédures sont mises à jour, les formateurs ou les décideurs doivent être capables d'adapter l'environnement virtuel, sinon ce dernier devient obsolète. Il faut donc, d'une part, que les connaissances soient représentées de manière explicite et séparées de l'implémentation du moteur, et d'autre part qu'elles soient représentées dans un formalisme qui soit relativement accessible à des non-informaticiens.

Réutilisabilité

La séparation du moteur de scénarisation et des connaissances utilisées par ce moteur est également nécessaire pour la réutilisabilité de ces connaissances.
Un même environnement virtuel pourrait ainsi être utilisé pour la formation de novices, pour la formation d'experts et pour de l'aide à la décision, en modifiant les objectifs scénaristiques ou en n'utilisant qu'une partie du contenu scénaristique. De même, il serait possible de réutiliser une partie d'un environnement virtuel pour une formation à une procédure similaire sur un autre site ou sur une machinerie différente.

1.3.4 Bilan sur les objectifs

Ces dix objectifs peuvent être regroupés autour de quatre axes :
- la **liberté d'action** de l'utilisateur,
- le **contrôle dynamique** du scénario, qui regroupe les objectifs de possibilités de contrôle, de caractère dynamique du contrôle, et de résilience du système de scénarisation,
- la **cohérence** du scénario global et des comportements individuels, qui regroupe les objectifs de validité écologique et d'explicabilité des scénarios,
- l'**adaptabilité** de l'environnement virtuel, qui regroupe les objectifs de variabilité, de passage à l'échelle, de maintenabilité et de réutilisabilité des scénarios.

L'objectif de cette thèse est donc de proposer un **système de scénarisation** d'environnements virtuels — un moteur de scénarisation et un ensemble de formalismes pour la modélisation du contenu et des objectifs scénaristiques — qui concilie ces quatres axes, en permettant d'assurer à la fois la **liberté d'action** de l'apprenant, le **contrôle dynamique** de la simulation, la **cohérence** des comportements présentés et l'**adaptabilité** de l'environnement virtuel.
Si ces objectifs découlent de nos motivations concernant les applications particulières que sont les environnements virtuels pour l'apprentissage ou pour l'aide à la décision, nous estimons cependant qu'ils sont suffisament larges pour qu'un système de scénarisation ainsi conçu puisse être utilisé dans d'autres contextes.

1.4 Problématique

L'objectif de cette thèse est de proposer un **système de scénarisation** d'environnements virtuels qui permette d'assurer à la fois la **liberté d'action** de l'apprenant, le **contrôle dynamique** de la simulation, la **cohérence** des comportements présentés et l'**adaptabilité** de l'environnement virtuel.
Le verrou lié à ces travaux naît de l'incompatibilité entre ces différents objectifs : le contrôle s'oppose

à l'adaptabilité, la liberté d'action va à l'encontre du contrôle, l'alliance du contrôle et de l'adaptabilité met en péril la cohérence, et ainsi de suite.
Le domaine de la narration interactive oppose ainsi traditionnellement le contrôle et l'adaptabilité des environnements virtuels, en classifiant les approches *orientées scénario*, qui pilotent l'environnement virtuel au niveau global, d'un côté, et *orientées émergence*, qui font émerger les scénarios à partir de comportements individuels, de l'autre.
Dans le domaine du jeu vidéo, on parle du **paradoxe narratif** pour désigner l'opposition fondamentale entre l'interactivité et la narration [Juul, 1998], c'est à dire que le fait de donner davantage de capacité d'action au joueur va venir entraver le récit prévu par l'auteur [Aylett, 1999].
De plus, ces jeux sont rarement adaptables, et la variabilité des scénarios n'est atteinte qu'au prix d'un gros travail de conception, puisque la plupart utilisent un modèle de narration arborescente, où tous les déroulements possibles doivent être décrits explicitement [Genvo, 2002].
Cet effort nécessaire pour le passage à l'échelle de scénarios cohérents et précisément contrôlés constitue ce qu'on appelle l'**authoring bottleneck** [Spierling and Szilas, 2009], et montre la nécessité de mettre en place des systèmes de scénarisation permettant de créer des environnements adaptables, sans avoir à définir explicitement l'intégralité des scénarios possibles.
Cependant, il est fréquent que les systèmes qui proposent de concilier contrôle et adaptabilité se bornent à apposer une surcouche de contrôle sur une simulation composée d'entités autonomes, et que les interventions de ces systèmes perturbent la cohérence de l'environnement en modifiant à la volée des états de la simulation.

1.5 Approche

Pour conserver la liberté d'action de l'utilisateur et assurer l'adaptabilité des scénarios, nous avons choisi d'opter pour une approche émergente et modulaire. Le contenu scénaristique est représenté au travers d'un ensemble de modèles — le modèle du domaine qui décrit les objets, les actions possibles et le fonctionnement des systèmes, et le modèle de l'activité qui décrit les comportements humains dans la situation considérée. L'utilisateur est libre d'effectuer n'importe quelle action permise par le modèle du domaine, et l'environnement est peuplé de personnages virtuels autonomes, capables de s'adapter aux changements dans l'environnement, dont le comportement est généré à partir du modèle de l'activité.
Ces modèles, ainsi que les moteurs qui vont permettre de faire évoluer l'état du monde et de générer les comportements des personnages virtuels, vont constituer le noyau de nos environnements virtuels. A ce niveau, les scénarios émergent des interactions libres de l'utilisateur, des personnages virtuels autonomes, et des systèmes simulés. Les scénarios ainsi créés sont explicables, puisqu'ils se conforment aux modèles, et valides écologiquement, les-dits modèles étant créés par des experts à partir d'analyses terrain.
Afin de pouvoir exercer un contrôle supplémentaire sur le déroulement des événements, on ajoute à ce noyau une sur-couche de scénarisation, à travers un module en charge de la gestion des objectifs scénaristiques. Ce module va, en premier lieu, utiliser les traces d'activité et le profil de l'apprenant pour déterminer de manière dynamique, à partir d'un ensemble de règles pédagogiques et scénaristiques, les objectifs scénaristiques à atteindre dans l'environnement virtuel. Dans un deuxième temps, il va chercher à générer et à réaliser un scénario qui respecte ces objectifs scénaristiques. Afin de maintenir la cohérence de la simulation, le module de scénarisation va utiliser les modèles qui la sous-tendent afin de prédire son évolution, et va chercher à guider cette évolution au travers d'un ensemble d'ajustements possibles, qui vont influencer les réactions des systèmes techniques et les prises de décision des personnages virtuels de manière indirecte, sans nuire à leur explicabilité.

1.6 Contributions

J'ai proposé dans le cadre de cette thèse un ensemble de contributions, qui sont résumées dans cette partie, avant d'être présentées en détail dans le reste du mémoire. Mes contributions majeures sont :

- SELDON, un modèle pour la scénarisation d'environnements virtuels,
- DIRECTOR, un moteur de planification et réalisation de scénarios prédictifs,
- CAUSALITY-DL, un langage permettant de représenter des modèles de la causalité.

J'ai également participé à la proposition de deux méta-modèles, dans le cadre d'un travail commun avec mon équipe de recherche :

- ACTIVITY-DL, un langage permettant de représenter des modèles de l'activité,
- DOMAIN-DL, un langage permettant de représenter des modèles du domaine.

1.6.1 Modèle pour la scénarisation d'environnements virtuels – SELDON

J'ai proposé le modèle SELDON, pour ScEnario and Learning situations adaptation through Dynamic OrchestratioN, qui permet d'adapter le scénario d'un environnement virtuel en fonction de l'activité de l'utilisateur en conciliant les problématiques de contrôle, de cohérence et d'adaptabilité.

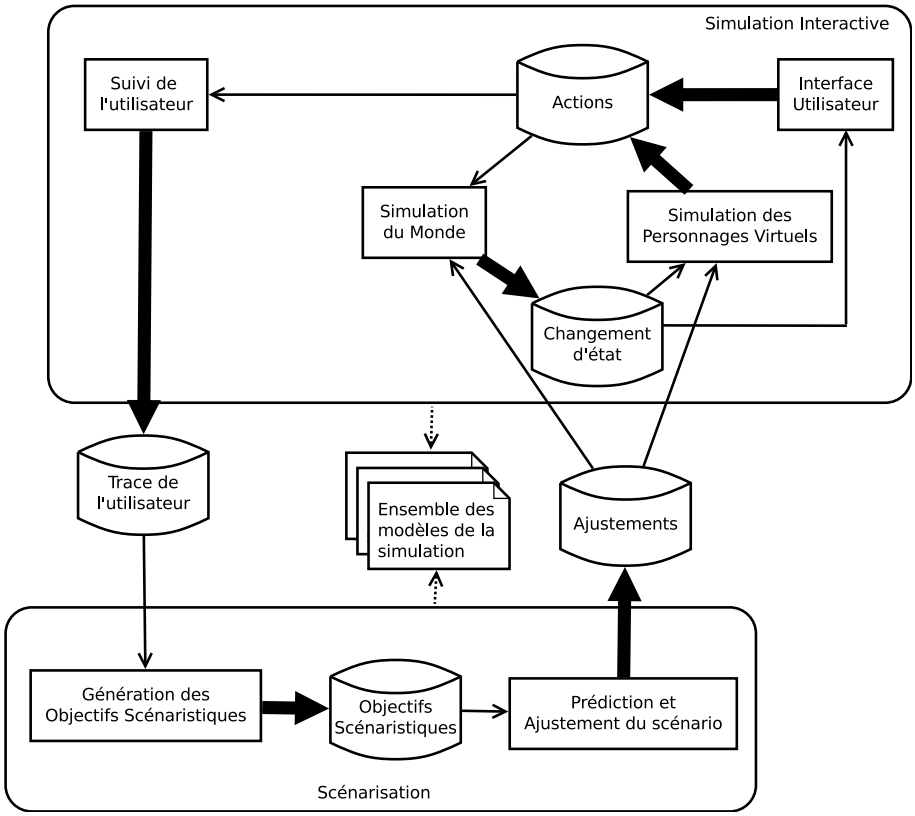


FIGURE 1.1 – Modèle SELDON (voir légende figure 1.2)

Le modèle SELDON tient son nom du personnage d'Hari Seldon, héros de la série de romans Fondation écrite par Isaac Asimov. Seldon est le fondateur de la psychohistoire, une science visant à prédire l'histoire à partir de connaissances sur la psychologie humaine. Pour mener l'humanité vers un meilleur futur, Hari Seldon utilise ses prédictions pour élaborer le Plan Seldon, qui décrit l'évolution souhaitée de la civilisation. Afin de s'assurer de la réalisation de ce plan, il met également en place des Fondations, organisations chargées d'agir localement pour guider l'évolution de l'humanité au moment des Crises Seldon, points clés du Plan Seldon.

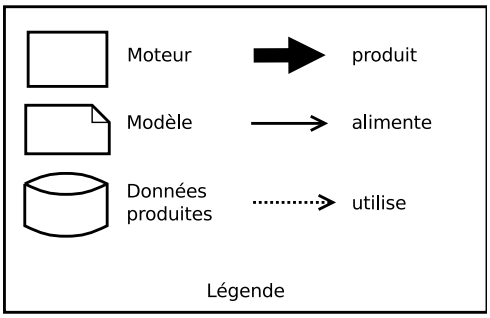


FIGURE 1.2 – Légende des schémas

Le principe derrière le modèle SELDON est similaire : à partir d'une simulation interactive peuplée d'entités autonomes, un module de scénarisation permet d'orienter indirectement le déroulement des événements en réalisant des ajustements ponctuels sur l'état du monde ou sur les personnages virtuels. Dans un premier temps, ce module utilise la trace d'activité de l'utilisateur pour générer un ensemble d'objectifs scénaristiques (désirabilité de situations particulières ou contraintes globales sur les propriétés du scénario). Puis il utilise les modèles qui sous-tendent la simulation pour prédire son évolution, et calculer un scénario répondant à ces objectifs à partir de ces prédictions et d'un ensemble d'ajustements possibles. Ces ajustements sont transmis aux moteurs de simulation du monde et de génération des comportements des personnages virtuels afin d'influencer leur évolution, de sorte à ce que le scénario prédit par le module de scénarisation se réalise effectivement dans l'environnement virtuel.

1.6.2 Moteur de planification et réalisation de scénarios prédictifs – DIRECTOR

Mes travaux de thèse se sont focalisés sur la génération et la réalisation du scénario dans l'environnement virtuel, ce qui correspond à la seconde phase de la scénarisation dans le modèle SELDON. Pour cela, j'ai proposé DIRECTOR, un moteur pour l'adaptation dynamique du scénario d'un environnement virtuel, utilisant de la planification pour prédire l'évolution de la simulation, et des ajustements sur l'état du monde pour guider cette évolution vers le scénario désiré.

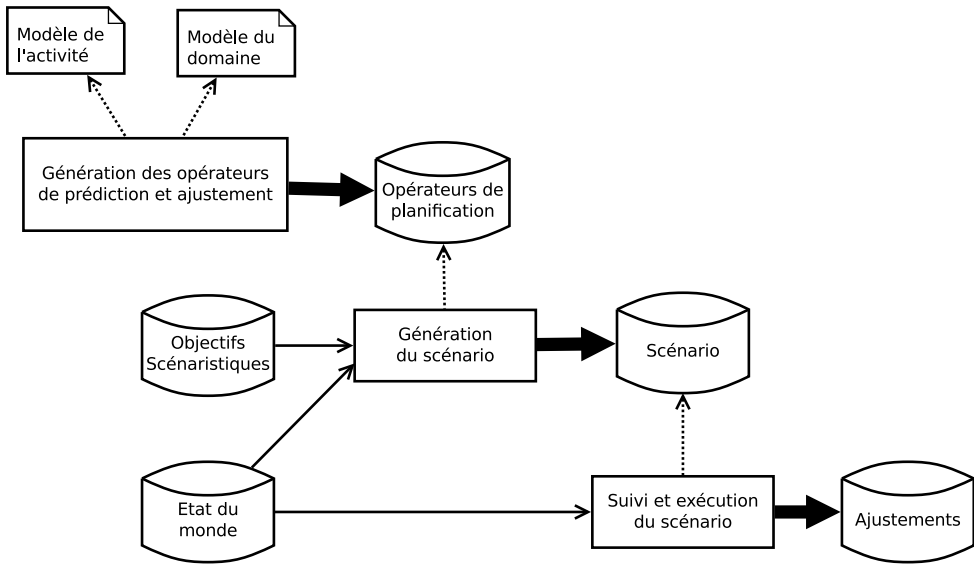


FIGURE 1.3 – Moteur DIRECTOR

En amont du lancement de l'environnement virtuel, DIRECTOR utilise les modèles du domaine et de l'activité pour générer un ensemble d'opérateurs de prédiction : des opérateurs de planification qui sont cadrés par des préconditions, de manière à permettre de prédire l'évolution de la simulation. Lorsque DIRECTOR, durant l'exécution de la simulation, reçoit un ensemble d'objectifs scénaristiques à atteindre, il utilise un planificateur, prenant en entrée à la fois les objectifs scénaristiques, les opérateurs de prédiction, et les opérateurs d'ajustements qui décrivent les interventions possibles sur l'environnement virtuel, pour planifier un scénario répondant aux objectifs. Le scénario planifié correspond au scénario souhaité dans la simulation, et contient à la fois des opérateurs de prédiction — qui décrivent les actions attendues de la part des personnages virtuels et de l'utilisateur, et les comportements attendus de la part des systèmes techniques — et des opérateurs d'ajustement — qui décrivent les ajustements nécessaires à la réalisation de ce scénario.
Puis, à la réception des messages indiquant les changements d'état dans l'environnement virtuel, DIRECTOR va comparer l'évolution de l'état du monde avec le scénario planifié, et déclencher au besoin les ajustements. Dans le cas où le scénario effectif dévie du scénario planifié, alors DIRECTOR planifiera un nouveau scénario tenant compte des changements dans l'environnement virtuel.

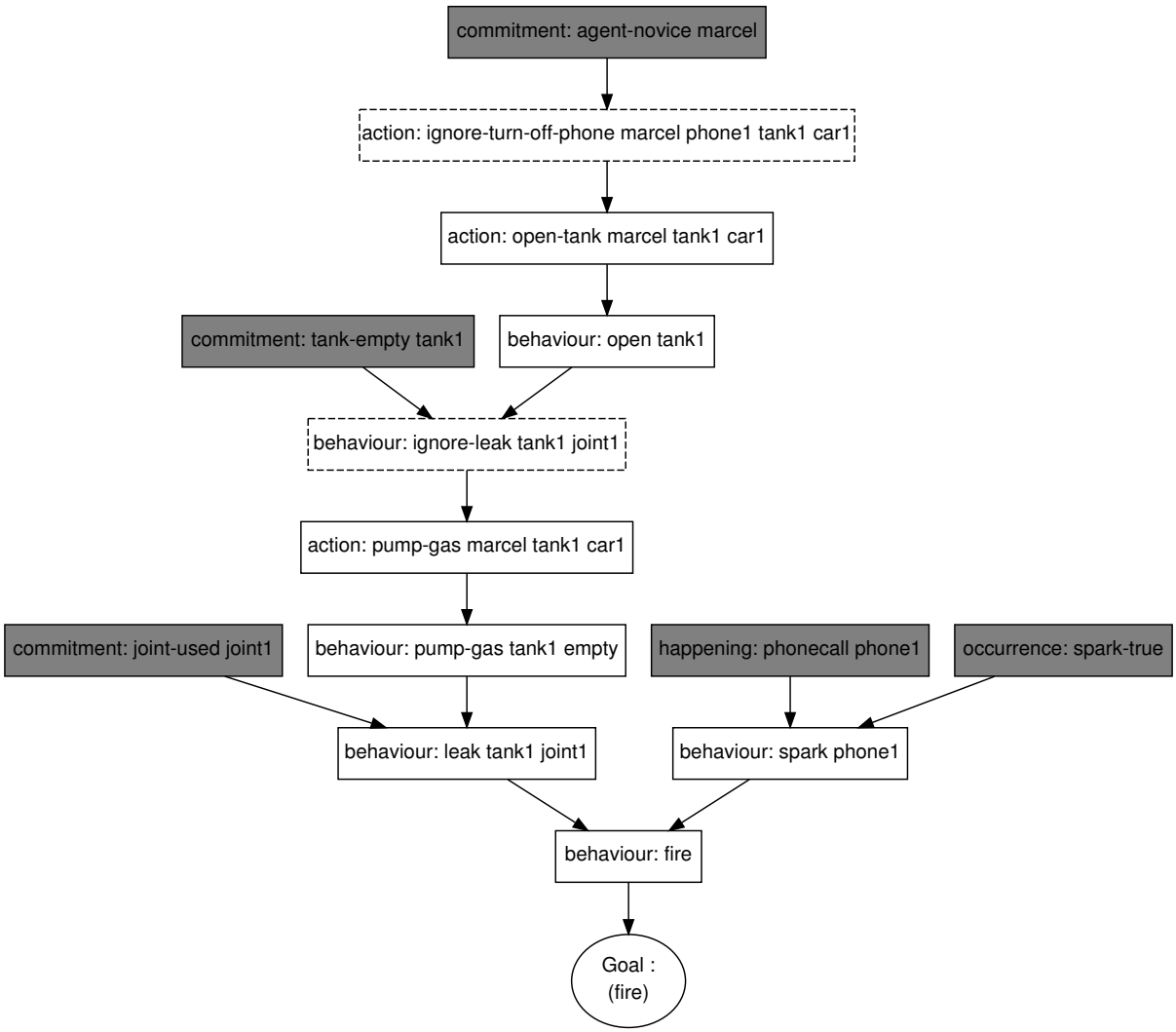


FIGURE 1.4 – Exemple de scénario prédictif construit par le modèle DIRECTOR

1.6.3 Méta-modèle de la causalité – CAUSALITY-DL

Afin de pouvoir contrôler de manière plus précise les scénarios générés par le système de scénarisation, j'ai proposé un langage nommé CAUSALITY-DL, qui repose sur un méta-modèle de la causalité.
CAUSALITY-DL permet de spécifier les relations de causalité entre les événements d'intérêt qui peuvent survenir dans l'environnement virtuel, ainsi que les barrières qui peuvent empêcher la propagation de la causalité entre ces événements.
Ce langage utilise des éléments de l'ontologie du modèle du domaine représenté dans le langage DOMAIN-DL, et est inspiré de formalismes d'analyses de risques : nœuds papillons et arbres de causes.
Dans le cadre de la scénarisation, le modèle de causalité permet de spécifier en amont un espace de trames scénaristiques d'intérêt (scénarios accidentels ressortis des analyses de risques, par exemple). Etant donné que le langage permet d'associer des valeurs aux événements (probabilité, complexité, etc.), il est également utilisé pour calculer les propriétés des différentes trames scénaristiques (crédibilité, complexité globale, etc.).

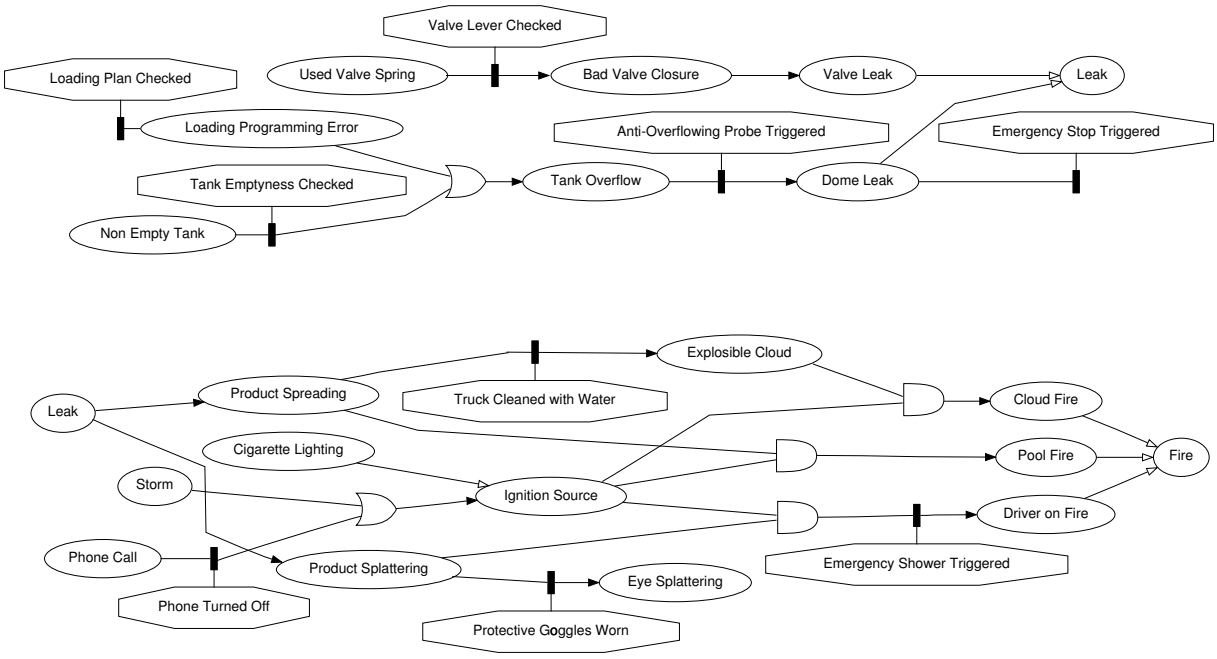


FIGURE 1.5 – Extrait d'un modèle de causalité

1.6.4 Autres contributions

Méta-modèle de l'activité – ACTIVITY-DL

J'ai participé avec l'ensemble de mon équipe aux spécifications du langage ACTIVITY-DL, qui repose sur un méta-modèle de l'activité issu des travaux précédents de l'équipe sur le formalisme HAWAI-DL [Edward et al., 2008]. ACTIVITY-DL permet la représentation de l'activité située par des ergonomes, sous un format interprétable par des modules informatiques. J'ai notamment proposé la formalisation des constructeurs et celle des types de préconditions.

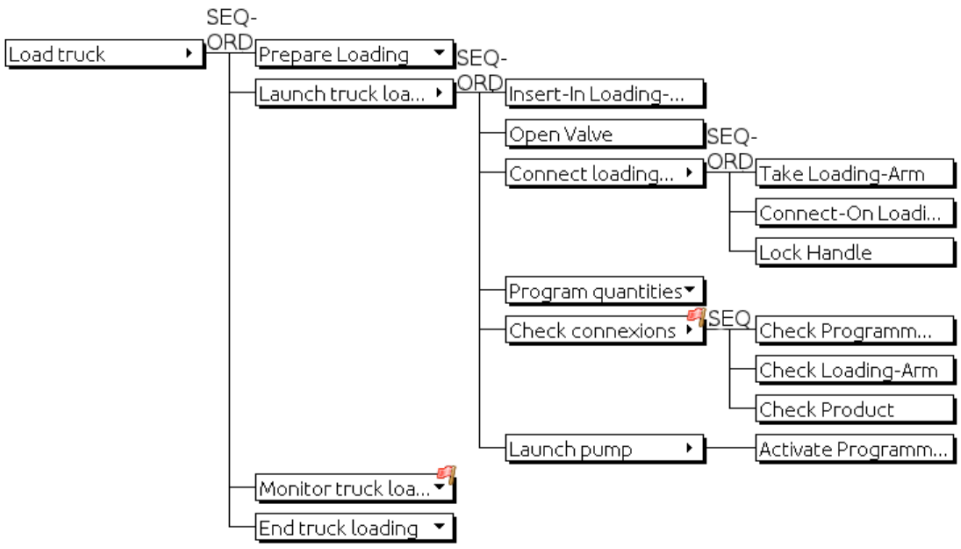


FIGURE 1.6 – Exemple de modèle représenté en ACTIVITY-DL

Méta-modèle du domaine – DOMAIN-DL

J'ai participé avec Kevin Carpentier aux spécifications du langage DOMAIN-DL, un formalisme ontologique qui permet de représenter les objets et actions possibles dans l'environnement virtuel, ainsi que les règles d'évolution des objets suite aux actions, au travers de règles de comportement. J'ai notamment travaillé sur le méta-modèle du domaine, en particulier sur la notion d'événements, commune avec le méta-modèle de causalité de CAUSALITY-DL.

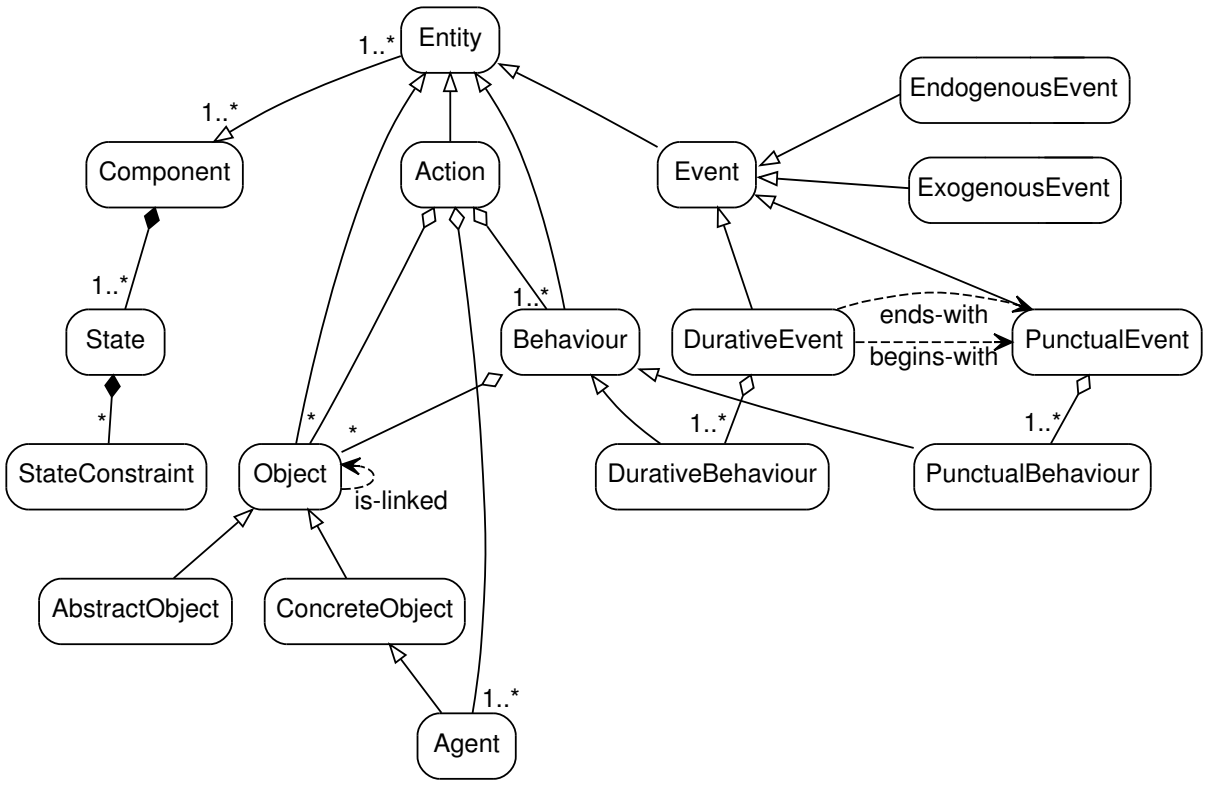


FIGURE 1.7 – Méta-modèle ontologique de DOMAIN-DL

1.7 Contexte

Le modèle SELDON a été implémenté au sein de la plateforme de création d'environnements virtuels HUMANS [Lanquepin et al., 2013], sous la forme de deux modules : TAILOR, qui fait l'objet de la thèse de doctorat de Kevin Carpentier [Carpentier et al., 2013], et DIRECTOR, qui fait l'objet de cette thèse. Cette plateforme a été utilisée pour créer plusieurs environnements virtuels dans le domaine de la formation.

1.7.1 Plateforme HUMANS

HUMANS, pour HUman Models based Artificial eNvironments Software platform, est une plateforme de création d'environnements virtuels complexes prenant en compte les facteurs humains. Elle est composée d'un ensemble de moteurs logiciels, qui raisonnent sur un ensemble de modèles communs. Grâce à cette approche modulaire, elle peut fonctionner avec différentes configurations, en ajoutant ou en enlevant des modules selon les besoins.

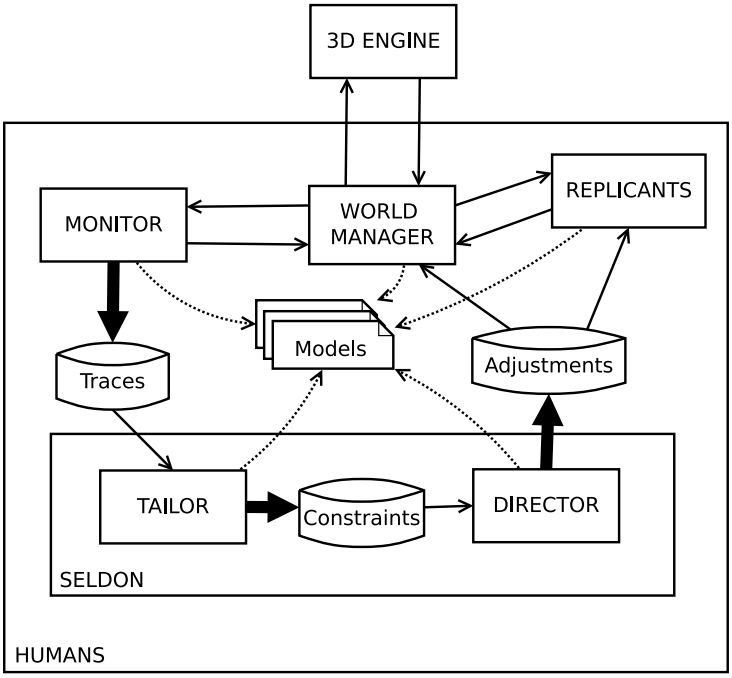


FIGURE 1.8 – La plateforme HUMANS.

Cette plateforme se place dans le cadre théorique de la didactique professionnelle et vise à encourager un apprentissage par la pratique, à partir de situations écologiquement valides qui reproduisent des situations de terrain. Afin de rendre compte de comportements observés, les modèles qui servent de base à la simulation sont créés à partir d'analyses terrain (analyses ergonomiques et analyses de risque). De plus, ces modèles ne contiennent pas seulement le déroulement idéal des procédures, mais décrivent également l'activité observée telle qu'elle apparait dans les conditions réelles, tendant vers une description de l'activité finalisée. Ainsi les erreurs et les violations conscientes sont prises en compte dans l'activité, que ce soit du côté de l'utilisateur ou de celui des personnages virtuels qui peuplent l'environnement. En effet, afin de rendre compte de la complexité des situations en termes de facteurs humains, la plateforme HUMANS permet de simuler des personnages virtuels riches, capables de comportements variés et adaptables.
Les scénarios émergent des comportements individuels des personnages virtuels, des systèmes techniques, et du ou des utilisateurs, afin de permettre de rendre compte d'une variété de situations sans

pour autant devoir les expliciter *a priori*. Ces scénarios sont représentés *a posteriori* sous la forme de traces de la simulation, qui peuvent être consultées par les utilisateurs ou utilisées comme entrée pour la scénarisation. Les scénarios peuvent en effet être adaptés dynamiquement grâce au modèle SELDON, afin d'offrir des situations d'apprentissage pertinentes en regard du profil et de l'activité de l'apprenant.
La plateforme HUMANS est composée de quatre modules :

- WORLD MANAGER, pour la gestion de l'état du monde ;
- REPLICANTS, pour la génération du comportement des personnages virtuels ;
- MONITOR, pour le suivi de l'apprenant et la génération de traces ;
- SELDON, pour la scénarisation adaptative.

Le module SELDON est lui-même divisé en deux sous-modules :

- TAILOR, pour la génération de situations d'apprentissage ;
- DIRECTOR, pour la génération et la mise en place de scénarios correspondant à ces situations d'apprentissage.

Ces modules utilisent un ensemble de modèles communs, représentés à l'aide de trois langages de description réunis dans le formalisme unifié HUMANS-DL. Les différents modules de la plateforme sont présentés plus en détail dans l'annexe B.

1.7.2 Cas d'application

La plateforme HUMANS a été utilisée pour créer plusieurs environnements virtuels pour la formation, dans des domaines tels que la maîtrise des risques, la maintenance aéronautique et la gestion de crise.
Trois cas d'application particuliers ont servi de support de réflexion pour les propositions présentées dans ce mémoire :

- le projet **ARAKIS**, sur la formation des opérateurs de dépôts pétroliers ;
- le projet **SimADVF**, sur la formation des assistantes de vie de famille ;
- le projet **NIKITA**, sur la formation des assembleurs-monteurs en aéronautique.

ARAKIS – Formation des opérateurs de dépôts pétroliers

Le projet ARAKIS, dans le cadre duquel s'est effectuée cette thèse, était financé par la Région Picardie et le FEDER. Il a été réalisé en partenariat avec l'INERIS et le LATI (Université Paris Descartes). Ce projet portait sur la scénarisation d'environnements virtuels pour la formation à la maîtrise des risques.
Le cas d'application abordé dans ce projet porte sur la formation des opérateurs de dépôts pétroliers. L'opérateur de dépôt pétrolier supervise le fonctionnement du dépôt, où des conducteurs de camions-citernes viennent charger leurs camions avec des hydrocarbures qu'ils iront ensuite distribuer dans des stations-essence. Du fait de la sous-traitance, ces situations sont sujettes à des contraintes particulières en termes de maîtrise des risques : le manque de contrôle direct de l'opérateur sur l'activité des conducteurs sous-traitants entraîne de leur part une application très formelle des procédures, sans réelle compréhension des risques qui sont en jeu.
L'opérateur, qui est en charge de la surveillance du dépôt, doit donc non seulement repérer les défauts matériels qui peuvent survenir, mais également faire l'interface avec les conducteurs et s'assurer qu'ils respectent les consignes de sécurité (port du casque et des lunettes de protection, coupure de la batterie du camion, etc.). Les fuites d'hydrocarbure pouvant avoir des conséquences catastrophiques, ces consignes sont très strictes. Pourtant, l'opérateur doit parfois faire face à des conducteurs expérimentés, qui se permettent de faire des compromis sur la sécurité (non-port des équipements de protection individuels par exemple), voire de passer outre certaines interdictions (utilisation du téléphone portable, cigarette, etc.). L'opérateur doit savoir gérer les problèmes relationnels avec ces conducteurs, en particulier lorsque, pressés, il le mettent lui-même sous pression. A l'inverse, il est parfois

nécessaire pour l'opérateur ou pour les conducteurs de dévier de la procédure prescrite, lorsque celle-ci ne permet pas de répondre aux problèmes rencontrés. C'est de ce type de comportements que l'on va chercher à rendre compte au travers des scénarios générés dans l'environnement virtuel. L'opérateur a également une mission de gestion des crises : c'est son rôle de sécuriser le dépôt, d'alerter et de réagir en cas d'accident (fuite d'hydrocarbures, feu, etc.). Ici, le système de scénarisation permettra d'atteindre la situation d'apprentissage permettant à l'apprenant de s'entraîner aux bonnes façons de faire et à réagir rapidement en cas d'accident.

FIGURE 1.9 – Capture d'écran du démonstrateur ARAKIS

La plupart des dépôts forment eux-mêmes leurs opérateurs, notamment avec du compagnonnage et des références régulières aux procédures. Dans ce cadre, l'objectif pour l'environnement virtuel est de maintenir les connaissances via un usage régulier (une fois par an ou tous les deux ans) sur des thèmes particuliers (par exemple : comment réagir à un épandage d'essence). Son utilisation doit également permettre de renforcer la vigilance sur les chargements : par la force de l'habitude un opérateur qui voit un conducteur sans ses lunettes de protection lui fera remarquer une fois, deux fois, puis abandonnera. Il est donc nécessaire qu'il prennne conscience des risques encourus.
Ce cas d'application fait suite à celui abordé dans le cadre du projet V3S [Barot et al., 2013], qui portait sur la formation des conducteurs de camions-citernes sous-traitants, sur la phase de chargement de matières dangereuses sur le dépôt. Les modèles réalisés pour le projet V3S ont été complétés par de nouvelles analyses terrain pour le projet ARAKIS.

SimADVF – Formation des assistantes de vie de famille

Le projet SimADVF, financé par la DGCIS sur l'appel à projets Serious Games 2009 et réalisé en partenariat avec l'AFPA et VirtuoFacto, portait sur la conception d'un environnement virtuel pour la formation des assistantes de vie de famille, et plus particulièrement pour l'apprentissage des activités touchant à la sécurité dans la garde à domicile des jeunes enfants.
Il ne s'agit donc plus ici de maîtrise des risques dans le domaine industriel, mais de prévention des accidents domestiques : chutes, brûlures, intoxications, etc. Cet environnement virtuel répond en réalité à deux ambitions : la formation des personnels des services à la personne d'un côté, et l'information de l'autre, avec la sensibilisation des parents et des enfants sur les risques d'accidents de la vie courante.
L'utilisation d'un environnement virtuel permet de confronter les apprenants à des situations impossibles à mettre en pratique réellement, en plaçant des enfants dans des situations dangereuses.

L'approche HUMANS est ici mise à profit en laissant aux utilisateurs la liberté d'expérimenter avec l'environnement, pour les mettre face aux conséquences de leurs erreurs. Les compétences visées portent en effet à la fois sur la prévention des risques en amont, et sur la réaction en situation de crise. La scénarisation va donc être utilisée pour mener à ces crises. Un intérêt particulier a été porté à l'augmentation progressive de la gravité des accidents au fur et à mesure que l'apprenant répète ses erreurs. Ainsi, si l'apprenant oublie de fermer la barrière sécurisant l'escalier au premier étage, la petite fille qu'il doit surveiller va profiter d'un instant de distraction pour tomber dans l'escalier, et se fera un bleu. Mais si l'apprenant reproduit plusieurs fois cet oubli, la chute de la petite fille aura des conséquences plus marquantes, allant jusqu'à lui faire se briser la nuque.

FIGURE 1.10 – Capture d'écran du démonstrateur SimADVF.

Pour la formation des assistantes de vie de famille, un troisième domaine de compétences est visé : la posture professionnelle. Il s'agit pour le professionnel de conserver une certaine distance affective entre lui et l'enfant, en trouvant le bon équilibre entre la compassion et la négation de l'autre. L'environnement virtuel doit dès lors faire l'objet d'une scénarisation pour créer des scénarios intéressants et mettant en jeu des personnages suffisamment riches pour susciter des émotions chez l'utilisateur.

NIKITA – Formation des assembleurs-monteurs en aéronautique

Le projet NIKITA, financé par l'ANR, réunit un ensemble de partenaires — Heudiasyc, le LATI (Université Paris Descartes), le CEA-LIST, Emissive, EADS, AEROLIA et le lycée technique Henry Potez — autour de la question des interactions naturelles dans les environnements virtuels pour la formation.
Il s'agit ici de former les assembleurs-monteurs, travaillant sur les lignes d'assemblage des fuselages d'avions, aux tâches de rivetage sur différents matériaux. S'il est toujours question dans une certaine mesure de maîtrise des risques, les risques ne sont plus de l'ordre des scénarios accidentels mais sont en termes de qualité.
En marge de la question centrale du projet, sur l'apport du geste et de l'interaction naturelle pour l'apprentissage, se pose la problématique de la représentation des connaissances pour simuler une situation de travail complexe : la procédure visée implique un grand nombre d'outils, de ressources et de matériaux différents, qui donnent lieu à de nombreuses combinaisons, et peut être réalisée de différentes manières, selon que les suites d'opérations devant être répétées sur des cibles similaires sont effectuées cible par cible ou bien en parallèle.
L'approche HUMANS est donc particulièrement intéressante sur ce cas d'application, puisqu'elle per-

met de générer les scénarios de manière émergente à partir de modélisations du domaine, de l'activité et de la causalité, là où une modélisation exhaustive des scénarios serait impossible.

FIGURE 1.11 – Capture d'écran du démonstrateur NIKITA.

1.8 Organisation du mémoire

L'état de l'art est présenté dans les chapitres 2 et 3. Le chapitre 2 présente les différentes approches utilisées pour la scénarisation d'environnements virtuels et les problèmes que ces approches rencontrent. Le chapitre 3 porte sur la représentation des connaissances nécessaires pour cette scénarisation, et se divise en trois sections : la représentation du contenu scénaristique, la représentation d'objectifs scénaristiques, et la représentation des scénarios eux-mêmes.
Nos contributions sont présentées dans les chapitres 4, 5, 6 et 7. Le chapitre 4 présente SELDON, notre modèle pour la scénarisation d'environnements virtuels basé sur le contrôle indirect d'entités autonomes à partir d'objectifs scénaristiques générés dynamiquement. Le chapitre 5 décrit les langages utilisés pour représenter le contenu scénaristique : DOMAIN-DL, qui permet d'écrire des modèles de domaine, et ACTIVITY-DL, utilisé pour la représentation de modèles d'activité. Le chapitre 6 présente notre contribution principale : DIRECTOR, un moteur de planification et de réalisation de scénarios prédictifs qui utilise les modèles de domaine et d'activité pour générer des tâches de planification permettant de calculer les ajustements nécessaires pour mener le scénario à une situation souhaitée, compte tenu des prédictions des comportements de l'utilisateur, des personnages virtuels et des systèmes techniques simulés. Le chapitre 7 décrit une extension de la phase de génération des scénarios de DIRECTOR permettant de prendre en compte des objectifs scénaristiques statiques et dynamiques, et présente notamment CAUSALITY-DL, un langage de représentation d'espaces de scénarios d'intérêt.
Les chapitres 8 et 9 concernent l'implémentation et les résultats obtenus. Le chapitre 8 décrit l'implémentation qui a été réalisée à partir de ces propositions. Cette implémentation a donné lieu à une première étape de validation, présentée dans le chapitre 9, au travers d'une évaluation informatique et de retours d'usage. Nous discutons également dans ce chapitre d'un ensemble de pistes d'évaluation pour les systèmes de scénarisation.
Enfin, nous proposons dans le chapitre 10 une discussion sur nos contributions, leurs limites, et les perspectives que nous envisageons pour étendre ce travail, avant de conclure dans le chapitre 11.

Première partie

Etat de l’art

Chapitre 2

Scénarisation dans les environnements virtuels

Sommaire

La scénarisation des environnements virtuels est une problématique complexe, qui nécessite de concilier des objectifs souvent considérés comme incompatibles : contrôle dynamique des événements, liberté d'action de l'utilisateur, cohérence du scénario global et des comportements individuels et adaptabilité du système.
Depuis une vingtaine d'années, de nombreux systèmes de scénarisation ont été proposés pour répondre à cette problématique, en particulier avec l'essor du domaine de la narration interactive. Reprenons notre définition des systèmes de scénarisation :

Système de scénarisation

Un système de scénarisation est composé d'un ou plusieurs langages de scénarisation, permettant de modéliser le contenu scénaristique et/ou les objectifs scénaristiques, et d'un moteur de scénarisation permettant de gérer de manière dynamique la réalisation du scénario.

Nous nous intéresserons dans ce chapitre aux moteurs de scénarisation, en présentant les différentes approches ayant été adoptées jusqu'ici pour le contrôle dynamique du scénario dans les environnements virtuels. Nous exposons tout d'abord différentes classifications utilisées pour ces approches, puis nous détaillons un certain nombre d'entre elles : les simulations pures, les systèmes basés sur un scénario prédéfini, ceux qui s'appuient sur un graphe multilinéaire, les systèmes optant pour une génération dynamique du scénario et ceux qui utilisent des personnages partiellement autonomes. Enfin, nous exposons un certain nombre de travaux portant sur le contrôle de personnages autonomes, avant de conclure sur notre positionnement. Les différents formalismes de représentation des connaissances utilisées pour la scénarisation seront présentés dans le chapitre suivant.

2.1 Classifications des approches pour la scénarisation

Le domaine de la narration interactive oppose classiquement les approches centrées sur le scénario et celles centrées sur les personnages, mais la réalité des systèmes est plus complexe. Nous faisons ici deux distinctions supplémentaires, portant sur d'autres aspects : d'une part, nous séparons les systèmes dont la scénarisation est intrinsèque et ceux où elle est extrinsèque ; d'autre part, nous identifions un continuum allant des systèmes basés sur des scénarios prédéfinis aux systèmes qui génèrent dynamiquement les scénarios.

2.1.1 Approches centrées sur le scénario ou sur les personnages

La dichotomie la plus répandue dans les systèmes de narration interactive oppose ainsi les approches centrées sur le scénario et celles centrées sur les personnages.
Les approches centrées sur le scénario (ou *strong story approaches* [Mateas and Stern, 2000]) mettent l'accent sur la qualité globale du scénario. Le contrôle des événements est centralisé dans un *drama manager* qui décide des événements à suivre et dirige l'ensemble des personnages de l'environnement. Les systèmes suivants suivent ce principe : IDA [Magerko, 2005], IDTension [Szilas, 2003], Crystal Island [Mott and Lester, 2006], Façade [Mateas, 2002].
Dans les approches centrées sur les personnages (ou *strong autonomy approaches* [Mateas and Stern, 2000]), au contraire, le scénario émerge des interactions entre l'utilisateur et les personnages virtuels qui peuplent l'environnement. Le contrôle est distribué, chaque personnage étant responsable de ses propres prises de décision. Ces approches se focalisent sur la création de personnages virtuels complexes et crédibles. C'est le cas des systèmes suivants : I-Storytelling [Cavazza et al., 2002], EmoEmma

[Pizzi and Cavazza, 2007], FearNot! [Aylett et al., 2006], MRE [Hill et al., 2001].

2.1.2 Scénarisation intrinsèque ou extrinsèque

On peut également faire la distinction au niveau de la conception des environnements virtuels entre ceux où la scénarisation est intrinsèque, et ceux où elle est extrinsèque.
La scénarisation est intrinsèque lorsqu'elle est partie intégrante du processus de conception de l'environnement virtuel. Dans ce cas, le moteur de scénarisation est indissociable du moteur de simulation de l'environnement, et l'ensemble des scénarios défini par les concepteurs est à la fois celui des scénarios possibles et celui des scénarios souhaitables. C'est le cas de nombreux environnements virtuels du domaine de la narration interactive, comme I-Storytelling [Cavazza et al., 2002] ou Façade [Mateas and Stern, 2005].
La scénarisation est extrinsèque si elle est vue comme une surcouche, une étape supplémentaire de cadrage ou de personnalisation d'un environnement virtuel existant. Elle consiste alors à définir un sous-espace de scénarios souhaitables à partir de l'espace de scénarios possibles dans la simulation, en sélectionnant, ordonnant ou priorisant des sous-ensembles d'événements en vue d'une utilisation particulière. C'est l'approche utilisée pour la paillasse virtuelle décrite dans [Marion, 2010], qui utilise le modèle de scénario pédagogique POSEIDON pour diriger une simulation précédemment créée avec le méta-modèle MASCARET [Baudouin, 2007].

2.1.3 Scénario prédéfini ou généré dynamiquement

Il existe également une différence entre les environnements virtuels qui s'appuient sur un scénario prédéfini, et ceux dont le scénario est généré dynamiquement. On parle alors dans ce dernier cas d'"approche générative", ou d'"approche émergente" [Aylett, 1999], et deux cas de figure se présentent : ceux où un scénario est généré en amont de la simulation, et ceux où le scénario est généré au fur et à mesure.
Le scénario peut être prédéfini, soit sous la forme d'une séquence linéaire d'événements, soit sous la forme d'un graphe multi-linéaire (scénario à embranchements). Le moteur de scénarisation sert à suivre et maintenir le déroulement de ce scénario, en limitant les choix de l'utilisateur ou en inhibant leur impact sur le déroulement des événements. L'accent est mis sur le respect de la volonté de l'auteur humain qui va créer le scénario et s'assurer de la qualité de l'expérience de l'utilisateur en termes de narration. Le système PAPOUS, par exemple, représente ainsi le scénario par une structure arborescente d'événements [Silva et al., 2003].
Les systèmes où le scénario est généré en amont fonctionnent de manière similaire, à ceci près qu'ils possèdent une phase préalable de génération du scénario. Ils se rapprochent en cela des systèmes de génération automatique d'histoires (*narrative generation*), comme TALE-SPIN [Meehan, 1977], Minstrel [Turner, 1993] ou, plus récemment, Fabulist [Riedl, 2004], qui visent à générer une séquence d'assertions et d'événements correspondant à un ensemble d'objectifs scénaristiques. La création du discours est alors remplacée par l'exécution du scénario dans l'environnement virtuel. Cette approche permet de réduire l'effort de création de scénarios et d'augmenter leur variabilité. On peut citer notamment le système Mimesis [Riedl et al., 2003], où le scénario est planifié en amont de la simulation et exécuté en donnant des ordres à des personnages virtuels. Si l'activité de l'utilisateur met en péril ce scénario alors soit les effets de son action seront annulés, soit le moteur de planification sera utilisé à nouveau pour réparer le plan en prenant en compte l'action de l'utilisateur.
Enfin, le scénario est parfois généré au fur et à mesure du déroulement de la simulation, soit de manière explicite via la sélection de l'état suivant par le moteur de scénarisation, soit en émergeant des interactions de l'utilisateur avec des personnages virtuels réagissant de manière autonome. L'accent est ici mis sur la liberté et la capacité d'action de l'utilisateur, et sur l'adaptabilité de l'environnement virtuel en cours de session. De nombreux environnements se trouvent dans ce cas de figure : Moe

[Weyhrauch, 1997], IDTension [Szilas, 2003], Crystal Island [Mott and Lester, 2006], FearNot! [Aylett et al., 2006], etc.
Il peut cependant être difficile de faire la distinction entre ce qui relève d'un scénario prédéfini et ce qui relève d'une génération dynamique. Les objectifs scénaristiques utilisés pour générer le scénario en amont, ou pour sélectionner l'état suivant, peuvent dans certains cas être représentés sous la forme d'une trame scénaristique, composée de points clés par lesquels le scénario devra passer. Selon le niveau auquel cette trame est détaillée, on pourra alors se trouver dans un cas où dans l'autre. Dans IDA [Magerko, 2005], par exemple, l'auteur spécifie des points clés et les ordonne partiellement, et le moteur de scénarisation complète dynamiquement le scénario entre ces points clés. De même, il est possible de prédéfinir seulement certains aspects du scénario. Ainsi, les auteurs de Façade [Mateas, 2002] distinguent, entre les systèmes basés sur des graphes d'histoires et ceux basés sur de la génération pure, une troisième approche qu'ils nomment *drama management*. Dans cette approche, les événements sont prédéfinis, mais l'ordre de ces derniers est déterminé de manière dynamique. Les événements de Façade étant complexes et faisant intervenir les actions synchronisées de plusieurs personnages, un gros effort de création du contenu a dû être fourni, malgré l'apport de la génération dynamique.

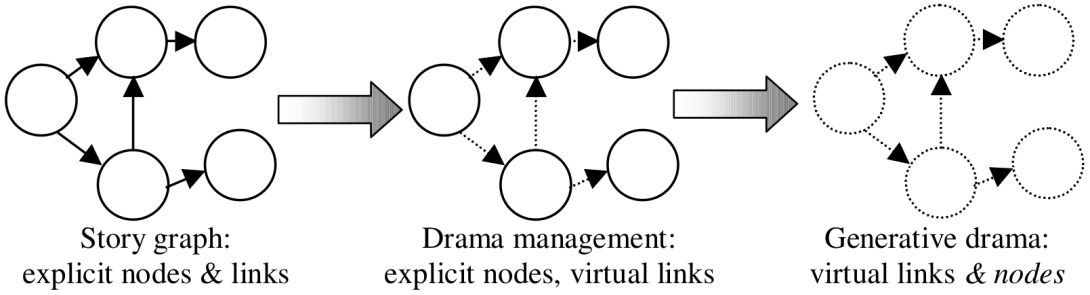


FIGURE 2.1 – Des graphes d'histoires à la génération dynamique [Mateas, 2002]

2.2 Simulations pures

Les environnements virtuels de type "simulation" ou "bac à sable" se trouvent à l'intersection des approches basées sur les personnages, de la scénarisation intrinsèque et des approches génératives. Ils sont centrés sur la simulation d'un monde virtuel, et laissent l'utilisateur expérimenter. Un nouvel état du monde est calculé à chaque étape en réaction aux actions de l'utilisateur et en fonction des règles de fonctionnement des systèmes techniques modélisés. Si l'environnement est peuplé de personnages virtuels, ces derniers sont autonomes, et leur comportement est généré par des systèmes allant de simples scripts à des architectures cognitives plus complexes. Nous présentons ici certains de ces environnements, allant de simulations de systèmes techniques comme CS WAVE [Da Dalto, 2004] et VTT [Crison et al., 2005] à des environnements contenant des personnages virtuels comme VRaptor [Shawver, 1997], I-Storytelling [Cavazza et al., 2002] et EmoEmma [Pizzi and Cavazza, 2007].

2.2.1 CS WAVE et VTT

La plupart des environnements de type "simulation" sont des simulateurs de systèmes techniques, utilisés pour la formation au geste ou l'entraînement à l'utilisation de machineries.
VTT (Virtual Technical Trainer) [Crison et al., 2005] est un environnement pour la formation à l'usinage. L'apprenant utilise une interface à retour d'effort pour saisir un outil de coupe virtuel, qu'il déplace pour réaliser des passes sur la pièce à usiner.

FIGURE 2.2 – Le dispositif CS WAVE [Da Dalto, 2004]

CS WAVE [Da Dalto, 2004] est une application de formation professionnelle aux gestes du soudage. L'utilisation de l'environnement est découpée en sessions d'utilisation de la simulation, avec différents exercices faisant intervenir plus ou moins de composantes du geste technique et plus ou moins d'assistances virtuelles.
Ces environnements utilisent des interfaces tangibles pour offrir une expérience au plus proche de la réalité. Ils utilisent des simulations physiques ou des règles d'évolution des systèmes techniques pour assurer la validité écologique des comportements des objets virtuels. Les scénarios abordés restent cependant relativement simples, et ne font pas intervenir de personnages virtuels.

2.2.2 VRaptor

VRaptor [Shawver, 1997] est un environnement virtuel visant à permettre un apprentissage situé, en confrontant un apprenant à de multiples situations. L'apprenant joue ici le rôle d'un membre d'équipe du FBI devant résoudre une situation de prise d'otages.

FIGURE 2.3 – Capture d'écran de VRaptor [Shawver, 1997]

L'environnement virtuel est peuplé de personnages virtuels au comportement réactif. Le formateur définit les conditions initiales en début de session, en choisissant pour chaque personnage sa position, son rôle (terroriste ou otage) et son comportement en cas de fusillade (panique, reddition, choc...). Les situations rencontrées dans la simulation par la suite dépendent fortement des actions de l'apprenant. Si le formateur souhaite confronter ce dernier à une situation particulière, il ne peut être certain que cette situation se produise. Par exemple, un apprenant pourrait tirer sur le terroriste avec qui le formateur souhaitait qu'il négocie.

2.2.3 I-Storytelling et EmoEmma

L'équipe Intelligent Virtual Environments de l'Université de Teeside a produit un certain nombre d'environnements virtuels basés sur l'approche centrée sur les personnages.
I-Storytelling [Cavazza et al., 2002] est une simulation de sitcom inspirée de la série *Friends*. Les personnages virtuels agissent de manière autonome, leur comportement étant représenté à l'aide de réseaux de tâches hiérarchiques (voir section 3.1.2). L'utilisateur peut influencer le déroulement de l'histoire en déplaçant des objets. Par exemple, Ross veut offrir des fleurs à Rachel pour la séduire mais échoue car l'utilisateur les a cachées.

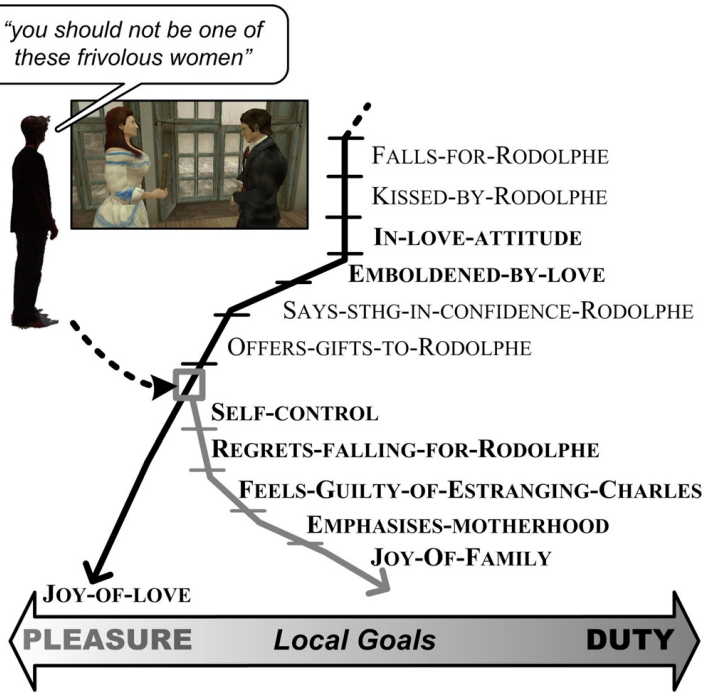


FIGURE 2.4 – Influence de l'utilisateur sur l'histoire dans EmoEmma [Pizzi and Cavazza, 2007]

EmoEmma [Pizzi and Cavazza, 2007] est basé sur le roman *Madame Bovary* et met en scène des personnages virtuels autonomes au comportement complexe, doté de désirs et d'émotions. L'utilisateur peut interagir avec Emma par le biais de langage naturel, soit en jouant le rôle d'un autre personnage, soit pour lui donner des conseils ou au contraire la sermonner, ce qui aura pour effet d'influencer son comportement.
Ces environnements visent à créer des histoires, et parviennent à donner à l'utilisateur le sentiment d'agir sur le cours des événements. Cependant, le scénario émergeant des actions choisies par ce dernier et par les personnages virtuels, il est impossible de s'assurer de son issue.

2.2.4 Bilan

Les systèmes de scénarisation centrés sur la simulation permettent d'assurer à la fois la **liberté d'action** de l'utilisateur, la **cohérence** des comportements — à la fois ceux des systèmes techniques et ceux des personnages virtuels — et l'**adaptabilité** des scénarios. Le scénario global émergeant des comportements individuels, il est possible de modéliser des comportements complexes de systèmes techniques et des situations de co-activité. La **variabilité** des scénarios ainsi générés permet d'utiliser ces environnements virtuels pour de l'apprentissage situé.
Cependant, ces systèmes laissent très peu de possibilités de **contrôle**, en dehors de celui des conditions initiales de la simulation. Une fois lancée, la simulation peut aller dans n'importe quelle direction, selon les actions de l'utilisateur, sans assurance que les situations rencontrées restent pertinentes par rapport au profil et à l'activité de ce dernier. Ainsi, dans un cadre d'apprentissage, il serait intéressant de limiter l'apparition de situations que l'apprenant n'a pas encore les capacités de résoudre, mais la simulation pourrait confronter un apprenant novice à une situation catastrophique qui n'aurait d'autre effet que de le traumatiser.
De plus, ces approches émergentes ne tendent pas vers une situation but particulière : si les règles de fonctionnement de la simulation ne permettent pas de s'assurer qu'on aboutira dans tous les cas à une situation terminale (ex : "les otages sont sauvés" ou "les otages sont morts"), alors dans certains cas le scénario généré n'a potentiellement pas de fin autre que celle de l'arrêt brutal de la simulation.

2.3 Scénarios prédéfinis

A l'inverse, les environnements virtuels contrôlés par un scénario linéaire pré-établi permettent de s'assurer que l'utilisateur rencontre des situations pertinentes. C'est notamment l'approche adoptée par de nombreux environnements de formation basés sur des scénarios pédagogiques.
Les personnages virtuels, de même que les systèmes techniques qui sont simulés, sont contrôlés de manière globale par le moteur de scénarisation, de sorte à exécuter le scénario prescrit. Afin que l'apprenant n'oriente pas le scénario dans une direction imprévue, ses possibilités d'actions sont souvent restreintes artificiellement. Dans le cas contraire, si ses actions mettent le scénario en péril, alors leurs effets sont annulés, soit de manière extra-diégétique, en arrêtant la simulation et repartant en arrière, soit de manière intra-diégétique, par exemple avec des interventions similaires à celles du système Mimesis [Riedl et al., 2003] présenté dans la partie 2.5. Nous présentons ici deux de ces systèmes : EMSAVE [Buttussi et al., 2013] et GVT [Gerbaud, 2008].

2.3.1 EMSAVE

EMSAVE [Vidani and Chittaro, 2009] [Buttussi et al., 2013] est un environnement virtuel pour la formation aux procédures de secourisme.
La simulation est basée sur un ensemble de scénarios prédéfinis, chacun mettant l'apprenant face à un patient ayant besoin de soins particuliers. Les apprenants doivent effectuer un ensemble d'examens et de traitements en suivant les procédures de premiers secours adaptées. A chaque étape, un ensemble d'actions est proposé à l'apprenant par le biais d'un menu. S'il choisit une action appropriée, l'environnement s'anime et évolue en appliquant les effets de l'action. Dans le cas contraire, le système fournit à l'apprenant un rapport résumant la situation et donnant des indices sur l'action correcte.

FIGURE 2.5 – Le menu de sélection d'action de EMSAVE [Vidani and Chittaro, 2009]

2.3.2 Generic Virtual Trainer

L'architecture GVT (Generic Virtual Trainer) [Gerbaud, 2008] [Gerbaud and Arnaldi, 2009] est destinée à la création d'environnements virtuels pour l'apprentissage de procédures industrielles, individuelles tout d'abord, puis collaboratives dans la seconde version de l'architecture.

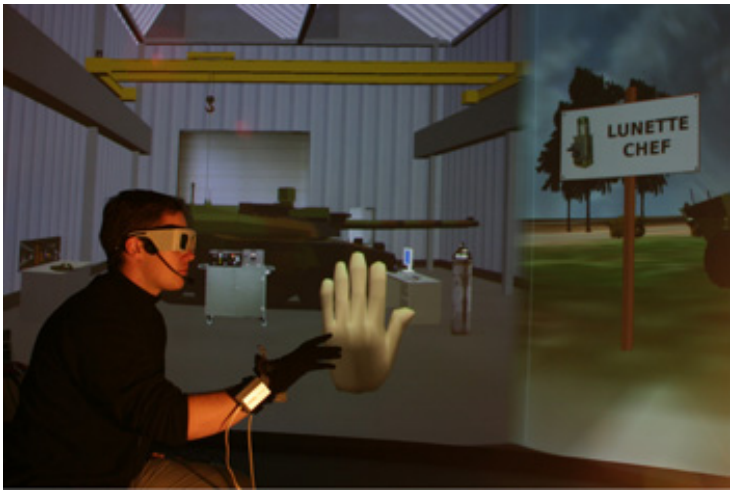


FIGURE 2.6 – L'environnement GVT [Gerbaud and Arnaldi, 2009]

La procédure revêt une importance centrale : les personnages virtuels, ainsi que le moteur de scénario, possèdent un modèle de cette procédure, représenté en langage LORA (décrit dans la section 3.3). Les personnages virtuels choisissent leurs actions dans la procédure en fonction de son état d'avancement — ils peuvent toutefois s'en écarter temporairement s'ils ont un profil de perturbateur.
Le moteur de scénario suit l'avancement de la procédure et contraint les possibilités d'interaction de l'apprenant aux actions autorisées selon cette dernière. Un moteur pédagogique peut également fournir une aide à l'apprenant en fonction de son niveau. Pour un novice, il s'agira de réaliser l'action à sa place, tandis que pour un apprenant plus expérimenté il s'agira d'afficher des documents pédagogiques et des rappels de la procédure.

2.3.3 Bilan

Les environnements qui suivent un scénario linéaire prédéfini permettent d'opérer un **contrôle** très poussé des situations rencontrées par l'utilisateur, ce qui est particulièrement pertinent lorsqu'il s'agit de situations pédagogiques.
La spécification explicite de ces scénarios permet d'assurer leur **cohérence**, ouvrant ainsi la possibilité de revenir avec l'apprenant sur le déroulement détaillé de la simulation après une session d'utilisation.
Cependant l'**adaptabilité** de ces systèmes est limitée, puisque chaque nouveau cas à traiter nécessite la conception d'un nouveau scénario, et que ces scénarios peuvent nécessiter d'importants efforts de conception, particulièrement lorsqu'il s'agit de former à des situations complexes et variées impliquant de la coactivité.
De plus, ces approches limitent drastiquement la **liberté d'action** de l'utilisateur, l'empêchant d'expérimenter avec la simulation et de pouvoir ainsi apprendre par essai-erreur.

2.4 Graphes multilinéaires

Pour assurer à la fois la liberté d'action de l'apprenant et la pertinence des scénarios déroulés, il est possible de définir à l'avance l'ensemble des scénarios de manière explicite, sous forme de graphes multilinéaires. Cette approche est similaire aux "livres dont vous êtes le héros", où le lecteur est confronté à un ensemble de choix possibles à la fin de chaque paragraphe, et où sa décision détermine le prochain paragraphe qu'il doit lire. L'ensemble des combinaisons de paragraphes possibles est déterminé par l'auteur, qui s'assure ainsi de leur qualité, cependant le lecteur a la sensation de prendre part activement à la création de sa propre histoire. Nous présentons ici trois environnements virtuels qui utilisent cette approche : PAPOUS [Silva et al., 2003], ICT Leaders [Gordon et al., 2004] et Façade [Mateas, 2002]. L'architecture U-Director [Mott and Lester, 2006], basée sur des réseaux bayésiens dynamiques, peut elle aussi être classée dans cette catégorie.

2.4.1 PAPOUS

Dans PAPOUS, un personnage virtuel raconte une histoire, qu'il fait évoluer en fonction de l'intérêt perçu dans l'assistance [Silva et al., 2003]. A chaque étape, le directeur choisit l'étape suivante à partir d'un arbre de "StoryBits" décrivant des morceaux de l'histoire et reprenant les fonctions narratives de Vladimir Propp [Propp, 1928].
D'après les auteurs, la définition d'histoires non-linéaires dans l'interface de création de cet arbre est une tâche ennuyeuse et difficile, car il est difficile pour un être humain de garder en mémoire une idée claire de ce qui se passe dans chaque version possible de l'histoire. Ils suggèrent donc d'automatiser cette tâche en générant dynamiquement le scénario.

2.4.2 ICT Leaders

Le projet ICT Leaders [Gordon et al., 2004] est un environnement virtuel pour la formation des dirigeants dans le domaine militaire qui a été développé en partenariat avec l'industrie du cinéma.
L'environnement virtuel présente à l'utilisateur des séquences cinématiques scriptées exécutées dans l'environnement virtuel, organisées dans un scénario à embranchements. A chaque séquence, l'apprenant doit prendre une décision lors d'un dialogue réalisé en langage naturel. Chaque décision binaire déclenche l'avancée dans la branche correspondante du scénario. Ce projet a été le fruit d'un énorme travail de conception des scénarios.

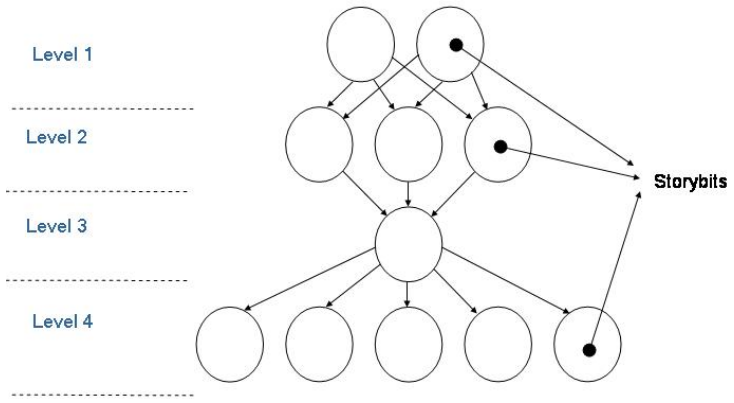


FIGURE 2.7 – Arbre de StoryBits [Silva et al., 2003]

FIGURE 2.8 – Capture d'écran de ICT Leaders [Gordon et al., 2004]

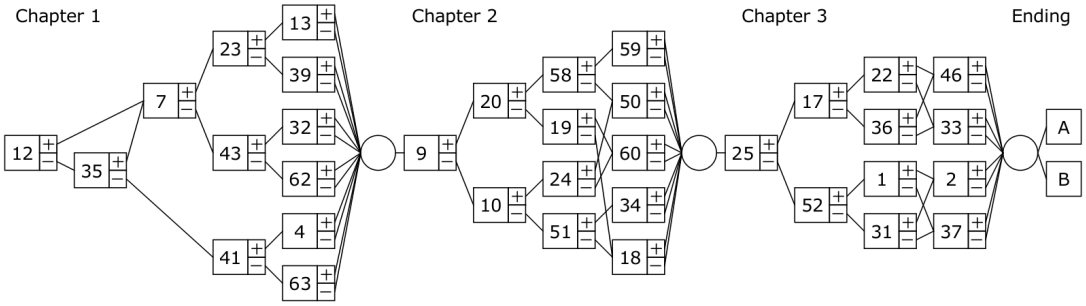


FIGURE 2.9 – Graphe de scénario de ICT Leaders [Iuppa et al., 2004]

Ainsi, parmi 63 situations d'apprentissages identifiées, les 35 plus importantes ont été sélectionnées et arrangées dans un graphe (voir figure 2.9). Pour chacune de ces situations, une séquence d'introduction et deux séquences correspondant aux résultats possibles ont été scriptées, définissant les mouvements des personnages, les dialogues, les mouvements de caméra, etc. De plus, pour chaque situation, la réponse du personnage virtuel pour chaque énoncé de l'apprenant reconnaissable par le système a également été scriptée.

2.4.3 Façade

Façade [Mateas, 2002] est un drame interactif permettant à l'utilisateur, via une interaction en langage naturel, d'intervenir dans une histoire mettant en scène un couple de personnages. Ces derniers, Trip et Grace, traversent une crise maritale, et le comportement de l'utilisateur déterminera la survie de leur couple et de son amitié avec eux.

FIGURE 2.10 – Capture d'écran de Façade [Mateas, 2002]

Façade utilise un compromis entre une approche basée sur un graphe multilinéaire et une approche de génération procédurale de contenu : les nœuds du graphe sont prédéfinis, mais les transitions entre ces noeuds sont dynamiques. Ces noeuds sont associés à des préconditions et des post-conditions qui permettent de renconstruire le graphe des transitions possibles entre les événements. Même sans définir explicitement le graphe, l'effort qui a été fourni pour concevoir l'histoire de Façade est gigantesque, mobilisant deux programmeurs expérimentés sur une période de 5 ans pour définir les 27 *beats* — les événements utilisés comme unité de narration dans Façade. La représentation de ces *beats* est détaillée dans la section 3.1.3.

2.4.4 Bilan

Les systèmes de scénarisation basés sur la sélection dans un graphe multilinéaire permettent de s'assurer de la **cohérence** des scénarios, et d'opérer un **contrôle** précis des événements, s'assurant de leur pertinence par rapport aux actions de l'utilisateur.
Ces systèmes permettent également de laisser une certaine **liberté d'action** à l'utilisateur, de par la création d'embranchements qui permettent de modifier le déroulement du scénario en fonction de ses choix.
Les environnements virtuels conçus à partir de ces systèmes peuvent proposer une très grande **variabilité** des scénarios. Façade est d'ailleurs reconnu comme le système le mieux finalisé et le plus riche en termes d'expérience narrative.
Cependant cette variabilité et cette liberté d'action se font au prix d'un énorme effort de conception des modèles. En effet, la nécessité de définir explicitement toutes les transitions entre événements, ou dans une moindre mesure, les conditions de passage d'un état à un autre, limitent fortement la **maintenabilité** et le **passage à l'échelle**.

2.5 Scénarios dynamiques

Pour alléger la charge de travail liée à la définition exhaustive de l'ensemble des chemins possibles, de nombreux travaux adoptant une approche centrée sur le scénario s'orientent vers la génération dynamique de ce dernier. Riedl et Young [Riedl and Young, 2006] proposent ainsi une méthode pour générer automatiquement un graphe multilinéaire prenant en compte les actions de l'utilisateur à partir d'un scénario linéaire. D'autres travaux se basent non pas sur un scénario prédéfini, mais sur des objectifs scénaristiques de plus haut niveau, comme des critères narratifs ou des situations clés à rencontrer. On trouve ici deux approches :

- la génération pas à pas, où le système de scénarisation considère à chaque étape, en réaction à l'action de l'utilisateur, l'événement ou l'action des personnages virtuels qui maximiserait un ensemble de critères scénaristiques, et l'exécute dans l'environnement virtuel. C'est le cas du système IDTension [Szilas, 2003].
- la génération, généralement par de la planification automatique, d'un scénario entier répondant à ces critères, qui est ensuite exécuté pas à pas et peut éventuellement être ajusté en fonction des actions de l'utilisateur. Nous décrivons ici le système Mimesis [Riedl et al., 2003], mais de nombreux autres systèmes utilisent cette approche. On peut notamment citer IDA [Magerko, 2005] et les travaux de Julie Porteous [Porteous et al., 2010] qui utilisent des points clés partiellement ordonnés pour guider la génération du scénario.

2.5.1 IDTension

IDTension [Szilas, 2003] est un système de génération d'histoires interactif, basé sur la sélection dynamique des actions des personnages virtuels de sorte à maximiser l'intérêt narratif du récit.
La sélection se fait en deux temps. Dans un premier temps le moteur de logique narrative (*narrative logic*) calcule les actions possibles pour chaque personnage en fonction de ses croyances et des différents états du monde. Puis, le moteur de séquençage narratif (*narrative sequencer*) classe ces actions de la plus intéressante à la moins intéressante pour le récit, et exécute la meilleure action dans l'environnement.
Ce classement se fait en fonction d'un ensemble de critères narratifs, parmi lesquels certains ont trait aux personnages virtuels : la cohérence éthique (l'action est-elle cohérente avec les actions précédentes du personnage et son système de valeurs), la cohérence motivationnelle (l'action est-elle cohérente avec les buts du personnage) ou encore la caractérisation (l'action aide-t-elle l'utilisateur à comprendre les propriétés du personnage). D'autres ont trait à l'action elle-même, comme sa pertinence par rapport aux actions qui l'ont directement précédée. Une action peut donc être choisie pour un personnage, même si elle ne faisait pas partie de ses priorités, dans le cas où elle est considérée comme plus intéressante pour la progression du récit.
L'approche proposée par IDTension est intéressante car elle vise à la fois la crédibilité des personnages virtuels (par l'utilisation d'un ensemble de critères liés à leur cohérence) et la qualité narrative du récit. Cependant la sélection d'une action ne se fait qu'en considérant sa pertinence locale, sans prendre en compte ses conséquences sur le récit à long terme. Il n'est donc pas possible de s'assurer de l'occurrence d'un événement particulier dans le scénario.

2.5.2 Mimesis

Mimesis [Riedl et al., 2003] [Young et al., 2004] est une architecture pour la création de jeux vidéo interactifs qui repose sur la génération de plans et le maintien de ces plans face aux interactions de l'utilisateur. Mimesis est une architecture à deux niveaux, composée du *Mimesis Controller*, chargé de générer, suivre et gérer l'exécution du scénario, et d'un moteur de jeu chargé de l'exécution bas-niveau du scénario. Le *Mimesis Controller* est composé à son tour de deux modules : le *Story World Planner*, chargé de générer le plan correspondant au scénario, et l'*Execution Manager*, chargé d'exé-

cuter ce plan. Le plan est généré à partir de requêtes précisant l'état courant du monde, un ensemble de buts à atteindre (les conditions devant être vraies au moment où le plan se termine), et la librairie d'actions disponibles pour construire le plan. Une fois ce plan généré il est transformé en une représentation sous forme de graphe orienté acyclique, utilisée pour déclencher les actions correspondantes dans le moteur de jeu (actions des agents et/ou objets comportementaux contrôlés par le moteur de jeu). Pour les actions du joueur, un nœud fictif est créé, dont l'état d'exécution est mis à jour lorsque l'utilisateur a atteint les conditions de succès de l'action en question.

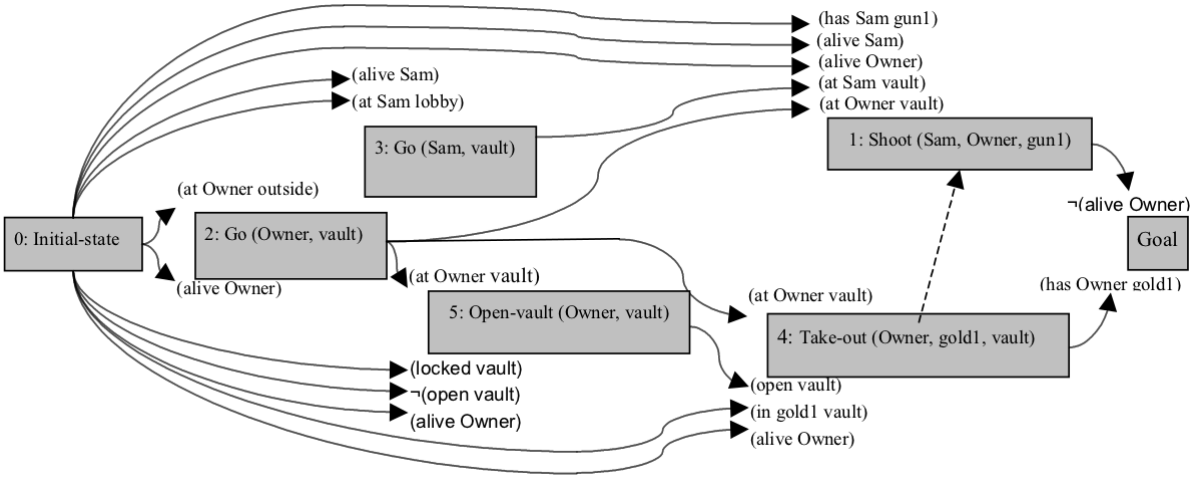


FIGURE 2.11 – Un plan généré par le système Mimesis [Riedl et al., 2003]

Afin d'assurer la résilience du scénario face aux actions de l'utilisateur qui peuvent aller à son encontre, Mimesis intègre un mécanisme intéressant de médiation narrative. A chaque fois que l'utilisateur choisit d'effectuer une action, le moteur de jeu vérifie que les effets de l'action ne sont pas en conflit avec les contraintes de causalité du scénario. Chaque action de l'utilisateur peut être considérée par rapport au plan comme en étant :

1. constituante (*constituent*) : l'action correspond à une action comprise dans le plan et considérée comme devant être exécutée par l'utilisateur ;
2. consistente (*consistent*) : l'action n'est pas constituante et aucun de ses effets n'interagit avec la portion de plan restante ;
3. exceptionnelle (*exceptional*) : l'action n'est pas constituante et un ou plus de ses effets menacent les conditions du monde requises pour les actions à venir.

Lorsqu'une telle exception est levée par le moteur de jeu, Mimesis possède deux moyens pour réaliser une médiation :

1. *intervention* : substituer les effets de l'action en la remplaçant par une autre instance d'action, appelée *failure mode* ; chaque action peut posséder plusieurs *failure modes* (par exemple, le coup de feu tiré par l'utilisateur peut passer à côté de sa cible),
2. *accomodation* : ajuster la structure du plan, en replanifiant en prenant en compte l'action de l'utilisateur.

Le *Story World Planner* de Mimesis utilise l'algorithme DPOCL (Decomposition and causality in partial-order planning), un algorithme de planification hiérarchique permettant d'obtenir des plans partiellement ordonnés.
Une limite rencontrée par la génération centralisée du scénario dans un contexte multi-agent est la crédibilité du scénario au niveau des personnages individuels : il faut que chaque action apparaisse cohérente avec les motivations du personnage qui l'effectue. Reprenons l'exemple utilisé pour Mimesis, où l'on souhaite générer un scénario mettant en scène un voleur et un directeur de banque qui

aboutisse à une situation où le voleur possède l'argent et le directeur de banque est mort. Le plan suivant correspondrait à un tel scénario : le voleur se rend à la banque puis le directeur de banque donne l'argent au voleur et se suicide. Un tel scénario ne serait cependant pas satisfaisant du point de vue des intentions des personnages. Pour pallier ce problème, Riedl a donc par la suite développé l'algorithme IPOCL (Intent-Driven Partial Order Causal Link) [Riedl, 2004], qui introduit des *frames of commitment* liées à des ensembles d'actions. Ces *frames* correspondent aux buts que le personnage doit adopter dans un intervalle donné pour que les actions qui s'y rapportent puissent figurer dans le plan. Ces travaux ont récemment été transposés à des moteurs de planification classiques par [Haslum, 2012].
La génération du plan dans Mimesis se fait en amont de son exécution dans l'environnement virtuel, et ne permet donc pas de modifier dynamiquement la situation finale en fonction des actions de l'utilisateur — le système ne fait que répondre à ces actions selon les stratégies de médiation narrative pré-calculées au moment de la génération du plan. Une extension de ce système qui générerait le plan dynamiquement permettrait d'offrir un contrôle plus poussé sur le déroulement des événements, mais a été écartée pour des raisons de coût computationnel.

2.5.3 Bilan

La sélection dynamique des événements du scénario permet d'assurer à la fois la **liberté d'action** de l'apprenant et l'**adaptabilité** du système. La **cohérence** des scénarios tient aux particularités de chaque système : dans IDTension, elle est le fruit de la représentation des motivations des personnages dans les règles de sélection des actions, et de la prise en compte de cette cohérence dans les critères narratifs guidant le choix des actions. Cependant, si cette approche permet d'avoir un **contrôle** dynamique sur le scénario, la portée de ce contrôle est limitée. Elle ne permet notamment pas de s'assurer de l'occurrence de situations particulières, ce qui est limitant lorsqu'il s'agit de mener un apprenant à des situations pédagogiques données.
La génération dynamique du scénario permet au contraire un **contrôle** plus poussé, puisque cette approche considère le scénario dans sa globalité. Des mécanismes comme celui présent dans Mimesis assurent une très grande **résilience** face aux actions de l'utilisateur. La **cohérence** du scénario, notamment au niveau des motivations des personnages individuels, nécessite elle-aussi l'adjonction de mécanismes particuliers.
La limite principale de cette approche se fait cependant ressentir lorsqu'il s'agit de passer de la génération d'une histoire purement textuelle à son exécution dans un environnement virtuel. Le plan généré ne contient en effet que les actions et événements signifiants par rapport à l'histoire, et ce qui ne pose pas de problème dans une représentation textuelle où les ellipses sont fréquentes devient plus gênant lorsqu'il s'agit d'un environnement où les personnages n'agissent qu'en fonction des ordres donnés par le moteur de scénarisation : un personnage qui ne serait impliqué que dans certains passages du scénario resterait alors immobile en attendant la prochaine instruction, faisant perdre toute crédibilité à l'environnement virtuel.

2.6 Personnages partiellement autonomes

Afin de contourner la complexité liée à la définition exhaustive de l'ensemble des scénarios possibles ou à la génération du comportement d'un ensemble de personnages individuels par un moteur de scénarisation centralisé, certains systèmes centrés sur le scénario laissent une autonomie partielle aux personnages virtuels. On peut ici distinguer les personnages virtuels autonomes — capables de choisir leurs propres buts et les actions à réaliser pour les atteindre — des personnages semi-autonomes — capables d'agir de manière autonome par moments, et d'exécuter des ordres à d'autres moments [Riedl et al., 2008b].

La distinction n'est cependant pas si triviale. Blumberg et Galyean [Blumberg and Galyean, 1997] donnent deux dimensions permettant de caractériser le degré de contrôle que l'on peut avoir sur des agents :

- le degré de contrôle, qui indique si un ordre a une valeur prescriptive ("il faut faire ceci") ou proscriptive ("il ne faut pas faire cela"), et l'importance qui lui est accordée,
- le niveau de contrôle, allant du plus direct au plus indirect.

Ils définissent ainsi quatre niveaux de contrôle :

- le niveau **moteur**, qui correspond aux actions unitaires (ex : "marche") ;
- le niveau **comportemental**, où une certaine planification est requise du personnage (ex : "va chercher un sandwich") ;
- le niveau **motivationnel**, qui prédispose le personnage à certaines actions (ex : "tu as envie de manger") ;
- le niveau **environnemental**, qui induit une réponse potentielle du personnage via la manipulation de son environnement (ex : "il y a un sandwich sur la table").

Nous considèrerons donc qu'un personnage virtuel est semi-autonome à partir du moment où il est possible d'opérer sur lui un contrôle moteur, comportemental ou motivationnel au cours de la simulation.

Nous commenterons ici deux systèmes faisant appel à des personnages autonomes — MRE [Hill et al., 2001] et FearNot! [Aylett et al., 2006] — et deux systèmes faisant appel à des personnages semi-autonomes — ISAT [Magerko et al., 2005] et IN-TALE [Riedl et al., 2008b]. L'architecture de pilotage proposée par [Delmas, 2009], bien que peu focalisée sur les personnages virtuels, pourrait également être assimilée à un système utilisant des personnages semi-autonomes, puisqu'en l'absence d'instructions chaque entité suit un comportement par défaut modélisé par un réseau de Petri.

2.6.1 MRE et FearNot!

Mission Rehearsal Exercise (MRE) [Hill et al., 2001] est un environnement virtuel pour la formation de dirigeants militaires. MRE adopte une approche mixte entre un graphe multilinéaire et l'utilisation de personnages autonomes. La simulation se base sur un *StoryNet*, qui contient un ensemble de nœuds — correspondant à des passages de simulation où l'utilisateur peut interagir librement avec des personnages autonomes — et des liens entre ces noeuds — qui sont des séquences d'événements scriptés qui font passer d'un noeud à l'autre.

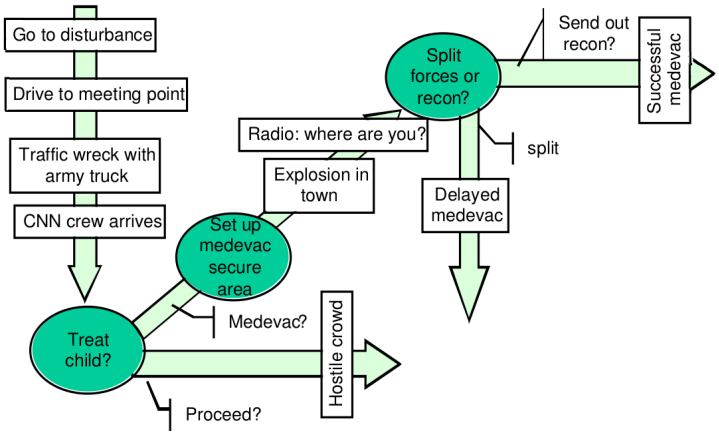


FIGURE 2.12 – Extrait du StoryNet de MRE [Hill et al., 2001]

Le système FearNot! [Aylett et al., 2006] utilise également des personnages autonomes, avec qui l'utilisateur interagit lors de différents épisodes. Ces épisodes sont définis par le lieu où ils se déroulent,

les personnages, et les buts adoptés par ces personnages. Ils sont sélectionnés dynamiquement par un *Story Facilitator*, en fonction de préconditions sur des événements. La génération du scénario global est ainsi similaire à celle que l'on peut trouver dans Façade.
Ces deux systèmes permettent à l'utilisateur d'interagir avec des personnages très complexes et dotés d'émotions, basés sur des architectures cognitives — EMA pour MRE, et FAtiMA pour FearNot!. L'utilisation de personnages autonomes lors de scènes atomiques permet une grande liberté d'action et une grande variabilité au niveau local, cependant la création du scénario global se heurte aux mêmes limites que les approches basées sur des graphes multilinéaires.

2.6.2 ISAT

L'architecture ISAT (Interactive Storytelling Architecture for Training) [Magerko et al., 2005] vise à introduire de l'adaptabilité dans les environnements virtuels pour la formation. ISAT possède un agent *director*, qui adapte l'environnement de manière dynamique pour répondre aux besoins de l'apprenant. Cet agent suit un scénario défini comme une collection de scènes partiellement ordonnées, possédant un certain nombre de paramètres sur lesquels l'agent *director* peut jouer pour construire une situation mettant en jeu les compétences qu'il souhaite tester chez l'apprenant.
Pour manipuler l'environnement, l'agent *director* peut ordonner aux personnages virtuels l'exécution de certaines actions, ou faire apparaître des personnages et des objets. Les actions ainsi déclenchées ne se rapportant pas aux motivations des personnages semi-autonomes, elles peuvent apparaître, aux yeux de l'utilisateur, comme incohérentes par rapport à leur comportement antérieur.

2.6.3 IN-TALE

IN-TALE (Interactive Narrative - Tacit Adaptive Leader Experience) [Riedl et al., 2008b] est un environnement virtuel destiné à la formation des militaires à des compétences de prise de décision et d'ouverture aux différences culturelles. IN-TALE est doté d'un *experience manager*, l'Automated Story Director, qui vise à réaliser dans l'environnement un scénario prédéterminé. Ce système reprend la suite des travaux réalisés sur Mimesis et Fabulist.
L'environnement est peuplé de personnages semi-autonomes, qui peuvent exécuter deux types de comportements :

- des comportements locaux autonomes (LAB, pour *Local Autonomous Behaviors*), qu'ils déclenchent par eux-mêmes et qui n'ont pas nécessairement d'effet sur le scénario (ex : faire des courses),
- des comportements spécifiques à l'histoire (NDB, pour *Narrative Directive Behaviors*), qui sont partie intégrante du scénario (ex : poser une bombe).

FIGURE 2.13 – Capture d'écran de IN-TALE [Riedl et al., 2008b]

Les NDB sont utilisés lorsqu'un personnage reçoit un ordre du *director* lui indiquant d'adopter un but particulier. Le *director* peut en effet donner deux types d'ordres aux personnages :

- des ordres prescriptifs, qui leur indiquent un but à adopter,
- des ordres proscriptifs, qui leur indiquent des actions interdites.

Contrairement à ISAT, le contrôle des personnages semi-autonomes ne se fait donc pas au niveau comportemental, mais au niveau motivationnel. Les LAB sont utilisés pour "meubler" la simulation, permettant de rendre les personnages plus crédibles en leur assignant un comportement, sans toutefois avoir à définir celui-ci au niveau du scénario. Cela ne permet pas toutefois d'éviter les incohérences, car rien ne permet d'assurer que les actions précédemment réalisées par le personnage ne soient pas en contradiction avec le nouveau but qui leur est assigné, et ces changements de motivations ne peuvent être expliqués par aucun événement apparent.
La même approche pour le contrôle de personnages semi-autonomes est adoptée dans les travaux ultérieurs de l'équipe, avec l'ajout d'un *Scenario Adaptor* [Niehaus and Riedl, 2009] qui permet de modifier dynamiquement le scénario de référence pour l'adapter à l'activité de l'apprenant.

2.6.4 Bilan

L'utilisation de **personnages autonomes** lors de scènes prédéterminées permet d'avoir à la fois de la **cohérence**, de la **variabilité** et de la **liberté d'action** pour l'utilisateur au niveau local. Cependant, le scénario global de la simulation étant déterminé de la même manière que pour les approches basées sur des graphes multilinéaires, leur utilisation ne fournit pas de solution satisfaisante au compromis entre la **variabilité** des scénarios au niveau global d'un côté, et la **maintenabilité** et le **passage à l'échelle** de l'autre.
Certains travaux se tournent alors vers l'utilisation de **personnages semi-autonomes**, dont les possibilités de contrôle permettent d'influencer dynamiquement le déroulement du scénario au niveau global. L'utilisation de personnages semi-autonomes permet alors une grande **variabilité**, tout en limitant l'effort à fournir pour la **maintenabilité** et le **passage à l'échelle**. Ces personnages pouvant être influencés de manière dynamique à différents niveaux, le système permet d'opérer un **contrôle** important sur le déroulement du scénario.
Cependant, les manipulations du système de scénarisation sur la simulation (apparition d'objets, changements dans les motivations des personnages...) mettent en péril sa **cohérence**, à la fois au niveau de l'**explicabilité** et de la **validité écologique** des scénarios. Que les personnages reçoivent des ordres au niveau de leurs actions ou bien de leurs motivations, rien ne garantit que les actions qui en résulteront seront perçues comme cohérentes avec leur comportement précédent. Or il a été montré dans [Si et al., 2010] que la cohérence percue au niveau des motivations des personnages virtuels est essentielle pour la compréhension du scénario.

2.7 Contrôle de personnages autonomes

Le contrôle de personnages semi-autonomes par le biais d'ordres au niveau comportemental ou motivationnel met ainsi en péril la cohérence de leurs comportements. Comment est-il alors possible pour un moteur de scénarisation centralisé d'influencer le comportement de personnages autonomes ou semi-autonomes ?
Il existe dans la littérature de nombreuses façons d'influencer le comportement de personnages autonomes, en jouant sur des facteurs tels que leurs buts, leur personnalité, leurs émotions ou encore leurs relations sociales [Rizzo et al., 1999], [Damiano and Pizzo, 2008], [Porteous et al., 2013]. Cependant, la nécessité de conserver l'explicabilité des comportements limite ces manipulations à la définition des conditions initiales de la simulation ; elles ne peuvent être utilisées de manière dynamique durant son déroulement.

Nous présentons ici un certain nombre de travaux visant à dépasser ces limites : l'*Initial State Revision* proposée par [Riedl, 2004], le *late commitment* de [Swartjes, 2010], l'architecture Thespian [Si, 2010] et la planification sociale de [Chang and Soo, 2008].

2.7.1 Initial State Revision

L'*Initial State Revision* (ISR) a été introduite par [Riedl, 2004] comme un moyen de réduire à la fois l'effort nécessaire à la création de contenu scénaristique et les limitations de l'utilisation de la planification pour la génération de scénarios, limitations qui résultent de l'hypothèse du monde clos. Son principe consiste à laisser un ensemble de faits indéfinis dans la description de l'état initial du monde, et à donner au moteur de planification la possibilité de valider ou de réfuter ces faits. Le moteur peut ainsi déterminer l'existence d'un objet, les attributs d'un objet existant, ou les relations entre plusieurs objets, de sorte à construire les conditions initiales permettant d'obtenir le meilleur scénario possible. Pour éviter les incohérences dans la base de faits, les faits indéfinis sont organisés dans des ensembles appelés *mutex sets*. Un *mutex set* est un ensemble de faits dont la valeur de vérité est indéterminée et dont un seul peut devenir vrai. Dès que l'un de ces faits devient vrai, les autres faits de l'ensemble sont considérés comme faux. Par exemple, `(at gun lobby)` et `(at gun office)` font partie du même ensemble, car un objet ne peut avoir qu'une position à la fois. Ce principe a par la suite été généralisé par [Ware and Young, 2010].

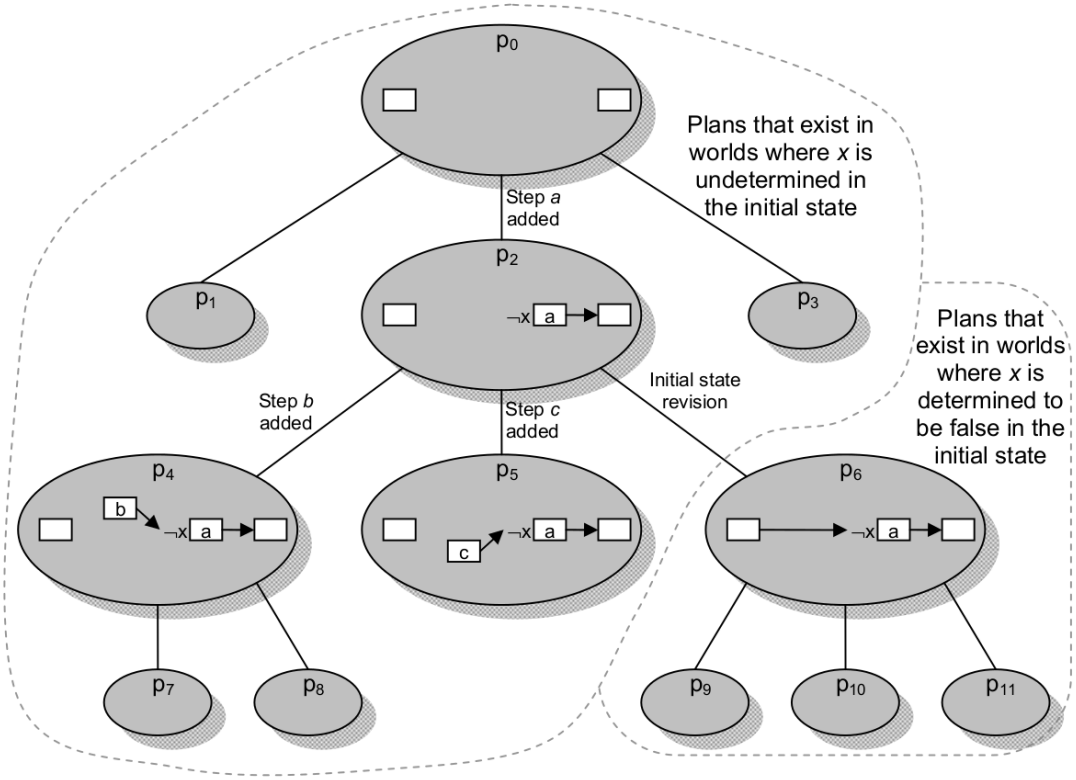


FIGURE 2.14 – Espace de recherche de plans résultant de l'ISR [Riedl, 2004]

L'ISR permet de manipuler l'état initial du monde pendant la génération du plan, et donc d'influencer le comportement des personnages virtuels de manière explicable, notamment lorsqu'il s'agit de valider ou réfuter des éléments de leur personnalité ou de leurs motivations. Cependant, cette manipulation a lieu lors de la génération du plan hors-ligne, et ne permet donc pas une influence dynamique sur les personnages durant la simulation.

2.7.2 Virtual Storyteller

[Swartjes, 2010] transpose cette idée à une approche centrée sur les personnages au sein du système Virtual Storyteller, avec le concept de *late commitment*.
Le Virtual Storyteller est créé autour du principe du *double appraisal*, c'est à dire que les personnages sont capables de raisonner à la fois *in-character* — "quelles sont les meilleures actions pour atteindre mes buts ?" — et *out-of-character* — "quelles sont les meilleures actions pour le développement de l'histoire ?".
Le *late commitment* — à ne pas confondre avec le *least commitment planning* — s'inspire du théâtre d'improvisation, où les acteurs décident des différents éléments qui caractérisent leur personnage (personnalité, histoire, relations avec les autres personnages...) au fil de leurs dialogues et de leurs actions, de sorte à justifier ces derniers.
Les personnages du Virtual Storyteller sont donc capables d'utiliser des *framing operators*, leur permettant, à la manière de l'ISR, d'ajouter des faits à l'état du monde. A l'instar des opérateurs utilisés pour représenter les actions, les *framing operators* ont des préconditions et des effets. Les effets correspondent non pas à un changement de l'état du monde, comme c'est le cas pour les actions, mais à un engagement (*commitment*). Ils sont exécutés *out-of-character*, comme si l'acteur disait aux autres acteurs "Disons que mon personnage est le père de ton personnage". Les préconditions sont utilisées pour éviter les incohérences, à la manière des *mutex sets*, ou pour définir un contexte à l'utilisateur de l'opérateur. Un exemple est présenté dans la figure 2.15.

CarryRapier	
Preconditions:	*Effects:*
`pirate(?char)`	`at(?rapier, ?char)`
`rapier(?rapier)`	
`owns(?char, ?rapier)`	
¬ `at(?rapier, ?loc)`	

FIGURE 2.15 – Un *framing operator* illustrant le principe de *late commitment* [Swartjes, 2010]

Les *framing operators* peuvent être utilisés par les personnages soit pour justifier la sélection d'une action, soit pour justifier l'adoption d'un but qui aurait été prédéfini par l'auteur : par exemple, si l'auteur stipule que le scénario doit contenir un personnage de pirate qui a pour but d'aborder un navire, alors un des personnages pourra utiliser un *framing operator* pour faire apparaître un navire en vue. Un *framing operator* peut également servir à justifier un autre *framing operator* : par exemple, un operateur `A a insulté B` pourra rendre vraies les préconditions d'un opérateur `B déteste A`.
Alors que l'ISR était utilisé pour la génération de plans hors-ligne, les manipulations du scénario permises par le *late commitment* sont ici dynamiques, puisque les personnages virtuels planifient leurs actions et utilisent les *framing operators* pendant qu'ils interagissent entre eux et avec l'utilisateur. Le fait que ces décisions soient prises par les personnages individuels est cependant problématique, puisqu'un opérateur qui bénéficierait à un personnage donné pourrait mettre en péril les plans d'un autre. Il est alors nécessaire de mettre en place un mécanisme de négociation entre les personnages.

2.7.3 Thespian

Thespian [Si, 2010] est une architecture pour la création et la simulation d'histoires interactives qui utilise une approche hybride entre les approches centrées sur le scénario et celles centrées sur les personnages. Thespian est construit autour de PsychSim [Pynadath and Marsella, 2004], un système multi-agent basé sur des Processus de décision markoviens partiellement observables (POMDP).
Le contrôle du scénario prend place en deux temps. Dans un premier temps, le moteur se base sur un ensemble de scénarios linéaires prédéfinis par un auteur pour calculer les pondérations des motiva-

tions des personnages virtuels qui permettraient de reproduire ces scénarios le mieux possible. Pour cela, il utilise une procédure d'ajustement (*fitting procedure*) qui reproduit les itérations réalisées par les auteurs humains lors de la création d'un système centré sur les personnages. Ce processus a lieu hors-ligne, en amont de la simulation, à la manière de l'ISR.
Puis dans un deuxième temps, il se base sur un ensemble de contraintes scénaristiques associées à ces scénarios pour corriger le déroulement des événements de manière dynamique. Pour cela, le moteur simule les actions des personnages jusqu'à un horizon donné, afin de détecter les éventuelles violations de ces contraintes (par exemple, si un événement A doit arriver avant l'événement B et que le moteur prédit que l'événement B va être réalisé alors que l'événement A ne l'est pas encore, il lèvera une exception). Dans ce cas, le système utilise un mécanisme proche du *late commitment* pour corriger le comportement du personnage fautif : Mei Si parle de "least commitment approach", c'est à dire que le système maintient un espace de configurations possibles pour les personnages, qui permettent de justifier le comportement ayant été exprimé jusqu'ici, et restreint peu à peu cet espace.
L'approche utilisée par Thespian est très puissante, en cela qu'elle permet de contrôler le scénario de manière dynamique, tout en respectant la cohérence des comportements des personnages virtuels. En particulier, l'utilisation de PsychSim pour simuler les agents permet des comportements très complexes, puisque les agents peuvent posséder des modèles récursifs des autres agents et de leurs croyances, et prendre en considération dans leur décision les actions qu'ils anticipent de la part des autres agents.
Cependant, cette richesse de comportement a un coût, et il n'est possible de réaliser cette prédiction qu'à un faible horizon, c'est à dire que sur quelques pas de simulation. Cela est particulièrement gênant lorsqu'il s'agit de s'assurer de l'occurrence d'une situation donnée, d'autant plus si cette situation est déterminée de manière dynamique lors de la simulation et non prédéfinie en amont comme c'est le cas ici.

2.7.4 Planification sociale

[Chang and Soo, 2008] proposent un système utilisant de la planification sociale pour permettre à un des personnages virtuels de manipuler les autres personnages pour leur faire réaliser les actions qu'il souhaite.
Leur exemple est basé sur une version simplifiée d'Othello de Shakespeare : Iago souhaite faire souffrir Othello en l'amenant à tuer sa propre femme, Desdémone. Pour cela, il manipule différents personnages afin que, suite à un quiproquo, Othello croie que sa femme le trompe avec un de ses lieutenants et les tue tous les deux. Le plan de Iago est présenté en figure 2.16.
Le plan du personnage qui réalise la planification sociale, en l'occurrence Iago, comprend donc des actions qui ne sont pas réalisées par lui mais par d'autres personnages. Pour certaines de ces actions, il a la possibilité de donner un ordre direct au personnage qui doit la réaliser (par exemple, Iago demande à sa femme Emilia de lui apporter le mouchoir de Desdémone). Il opère alors un contrôle motivationnel sur l'autre personnage. Pour d'autres, il doit amener indirectement l'autre personnage à leur réalisation, en agissant sur l'état du monde (par exemple, Iago dépose le mouchoir dans la résidence de Cassio afin que celui ci le ramasse). Il opère alors un contrôle environnemental.
Pour pouvoir générer de tels plans, le moteur de planification doit être capable de considérer à la fois les actions du personnage à l'origine du plan, qui pourront être exécutées, mais également les réactions des autres personnages à ces actions en termes de croyances, et les plans générés par ces personnages eux-mêmes. Pour cela, il intègre deux éléments : une description des mécanismes mentaux des personnages, et un ensemble de préconditions intentionnelles. La description des mécanismes mentaux utilise des "prédicats dérivés" pour représenter les mécanismes de révision des croyances et des buts des agents. Les prédicats dérivés permettent d'exprimer des faits qui peuvent être dérivés d'autres faits, mais ne peuvent pas être considérés comme les effets directs d'une action. Les préconditions intentionnelles sont ajoutées aux actions pour préciser qu'un opérateur `(agent, action)` ne peut être appliqué que si l'agent est le personnage à l'origine du plan, ou si l'agent a

comme but les effets de l'action.

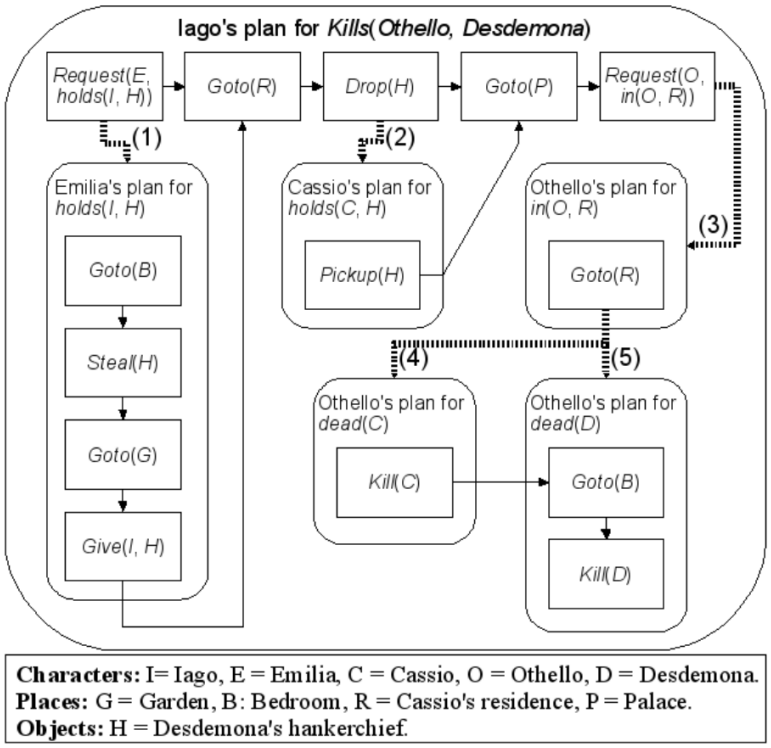


FIGURE 2.16 – Un plan généré par le moteur de planification sociale de [Chang and Soo, 2008]

Il serait possible de transposer ce système de planification sociale en considérant que l'agent qui génère le plan n'est plus un personnage virtuel cherchant à manipuler les autres, mais un système de scénarisation centralisé cherchant à manipuler les personnages virtuels d'un environnement pour réaliser un scénario. Les actions exécutables par ce système seraient cependant à reconsidérer, celui-ci ne pouvant agir directement sur le monde sans perturber la cohérence du scénario.
L'algorithme proposé possède de plus certaines limites. La définition des préconditions intentionnelles borne notamment les actions pouvant être considérées de la part des autres personnages à celles ayant directement pour effet le but du personnage, et ne permet donc pas de considérer des plans à l'intérieur du plan. Pour cela, les auteurs définissent des macro-actions, qui sont des suites d'actions permettant de réaliser un but donné, mais la nécessité de définir ces macro-actions explicitement est très contraignante.

2.7.5 Bilan

Ces travaux proposent un ensemble de solutions intéressantes pour faire face à notre problématique, qui est de concilier contrôle, liberté d'action, cohérence et adaptabilité, mais aucune de ces solutions n'est entièrement satisfaisante. L'approche centrée sur les personnages utilisée par le Virtual Storyteller, tout comme la simulation à un horizon limité effectuée par Thespian, ne permettent pas de s'assurer de l'occurrence d'une situation donnée comme c'est le cas avec les approches qui utilisent de la planification au niveau du scénario global. Ceci est particulièrement contraignant lorsqu'il s'agit de confronter un apprenant à des situations pédagogiques données. L'ISR ne permet pas de contrôler la simulation de manière dynamique, quant à la planification sociale, elle est adaptée au niveau d'un personnage donné mais peu au niveau d'un moteur de scénarisation centralisé.
Deux éléments sont toutefois à retenir :

– le **late commitment** réalisé sur les états et motivations des personnages virtuels avec l'ISR tout d'abord, puis dans le Virtual Storyteller et dans Thespian, qui permettent de conserver la cohérence des comportements de personnages autonomes, tout en exerçant dessus un certain contrôle ;
– la **prédiction dynamique des actions** des personnages virtuels réalisée dans Thespian et dans le moteur de planification sociale, qui permet de concilier les approches centrées sur le scénario et celles centrées sur les personnages, en permettant là encore un contrôle centralisé et dynamique sur des personnages au comportement complexe.

2.8 Bilan global et positionnement

La littérature sur les systèmes de scénarisation, en particulier dans le domaine de la narration interactive, oppose classiquement les approches centrées sur le scénario d'un côté et les approches centrées sur les personnages de l'autre, mais la distinction entre ces deux approches n'est en réalité pas si nette, notamment car de plus en plus de systèmes optent pour une approche mixte en incluant des personnages autonomes ou semi-autonomes dans des systèmes centrés sur le scénario.
Nous avons ainsi détaillé et proposé une critique d'un ensemble d'approches :

– Les systèmes basés sur des **simulations pures** laissent une grande liberté d'action à l'utilisateur tout en assurant la cohérence des comportements des systèmes techniques et des personnages virtuels. Ces systèmes permettent une représentation distribuée des règles d'évolution de la simulation, ce qui assure son adaptabilité. Cependant, la simulation fonctionnant en complète autonomie, les possibilités de contrôle durant son déroulement sont très limitées.
– Les systèmes qui suivent un **scénario linéaire prédéfini** offrent un contrôle très fort sur les situations rencontrées par l'utilisateur, et permettent à la personne en charge de leur définition de s'assurer de leur cohérence. La liberté d'action est cependant quasiment absente, ce qui empêche l'apprentissage par essai-erreur, et l'adaptabilité est elle aussi très limitée.
– Les systèmes basés sur des **graphes multilinéaires** visent à concilier liberté d'action, contrôle et cohérence, à la manière des "livres dont vous êtes le héros". Ils pêchent cependant au niveau de l'adaptabilité, la création et modification de ces graphes étant très coûteuse, même lorsque ces graphes ne sont que partiellement définis comme c'est le cas dans Façade [Mateas, 2002].
– La génération de **scénarios dynamiques** apporte une solution quant à l'adaptabilité, mais les systèmes basés sur la **sélection dynamique** des événements ne permettent pas un contrôle à long terme des événements, tandis qu'il est difficile d'assurer la cohérence des comportements des personnages inviduels dans les systèmes basés sur la **planification du scénario**.
– L'utilisation de **personnages semi-autonomes** offre également davantage d'adaptabilité, mais les conflits entre les motivations propres des personnages et les ordres qu'il reçoivent du système de scénarisation posent là encore un problème au niveau de la cohérence. Quant à l'utilisation de **personnages autonomes** lors de scènes au sein de scénarios multilinéaires, si elle permet adaptabilité et liberté d'action au niveau des scènes, elle ne résoud pas le problème posé par la définition exhaustive du scénario au niveau global.
– Enfin, certains travaux récents se sont intéressés à la problématique du **contrôle de personnages autonomes**. Si aucun ne permet de répondre directement à notre problématique, en conciliant liberté d'action de l'utilisateur, contrôle dynamique du scénario, cohérence des comportements individuels et adaptabilité du système, nous retenons toutefois deux éléments intéressants : le **late commitment** qui permet de raffiner progressivement l'état initial du monde, et la **prédiction des actions des personnages virtuels** lors de la génération ou du suivi du scénario global.

Le tableau 2.1 reprend les points principaux de ces différentes approches.
Nous proposons un modèle pour la scénarisation des environnements virtuels, nommé SELDON, qui repose sur la génération dynamique et le contrôle indirect du scénario dans un environnement virtuel peuplé de personnages autonomes. Cette proposition intègre un ensemble d'éléments permettant de

pallier les différentes lacunes de la littérature :

- la **scénarisation extrinsèque** de l'environnement virtuel, qui augmente la **réutilisabilité** du système en permettant son utilisation avec différentes simulations ;
- la possibilité pour l'apprenant de choisir ses actions à partir de l'**ensemble des actions possibles**, plutôt qu'à partir d'une présélection dépendante d'un scénario prédéfini, pour assurer sa **liberté d'action** ;
- la **génération dynamique** du scénario à partir de contenu scénaristique, qui permet de laisser à l'utilisateur cette **liberté d'action** et d'alléger la charge de travail liée à la création des scénarios, augmentant ainsi l'**adaptabilité** du système ;
- la **planification du scénario global**, plutôt que la sélection d'événements au niveau local, qui permet d'assurer un **contrôle** sur le long terme du scénario, et notamment de l'orienter vers des situations précises ;
- l'intégration de **personnages virtuels autonomes**, qui augmente elle aussi l'**adaptabilité** du système en permettant l'émergence de situations non définies au préalable ;
- le **contrôle indirect** de ces personnages, que ce soit par un contrôle environnemental ou par le raffinement progressif de leurs états internes (*late commitment*), qui permet de conserver la **cohérence** de leurs comportements ;
- la **prédiction de ces comportements** et leur **suivi en temps-réel** qui permet de les prendre en compte au niveau de la génération du scénario et de son exécution.

Le modèle SELDON est présenté dans le chapitre 4. Cette scénarisation est réalisée par le biais d'un moteur de planification et de réalisation de scénarios prédictifs, nommé DIRECTOR. DIRECTOR est présenté dans le chapitre 6.

	Approches			Objectifs			
	centrée sur...	intri-/extrinsèque	scénario prédéfini/généré	contrôle	liberté d'action	cohérence	adaptabilité
Simulations pures							
CS-WAVE, VTT, VRaptor, I-Storytelling, EmoEmma	personnages	intrinsèque	généré	−	++	++	++
Scénarios prédéfinis							
EMSAVE	scénario	intrinsèque	prédéfini	++	−	++	−
GVT		extrinsèque					
Graphes multilinéaires							
PAPOUS, ICT Leaders	scénario	intrinsèque	prédéfini	++	+	++	−
Façade, U-Director			en partie prédéfini				
Scénarios dynamiques							
IDTension	scénario	intrinsèque	généré	++ (local) / −	++	+	++
Mimesis				++	++	−	++
Personnages partiellement autonomes							
MRE, FearNot!	personnages	intrinsèque	généré (local) / prédéfini	+	+	++	++ (local) / −
ISAT, IN-TALE	scénario		en partie prédéfini	++	++	−	++
Contrôle de personnages autonomes							
ISR	scénario	intrinsèque	généré	−	−	++	++
Virtual Storyteller	personnages		généré	++ (local) / −	++		++
Thespian	mixte		en partie prédéfini	++ (local) / −	++		+
Planification sociale	personnages		généré	+	pas d'utilisateur		−

TABLE 2.1 – Tableau récapitulatif des différentes approches présentées pour la scénarisation en environnement virtuel

Chapitre 3

Représentation des connaissances pour la scénarisation

Sommaire

La seconde composante de ces systèmes est l'ensemble des **langages** et **formalismes** [1] permettant de représenter les connaissances qui sous-tendent l'environnement virtuel, et qui vont être utilisées par le moteur de scénarisation. Ces langages sont utilisés par les auteurs, formateurs et concepteurs de l'environnement virtuel pour décrire l'ensemble des scénarios possibles et souhaitables dans la simulation — soit directement, soit en faisant appel à un informaticien chargé de l'écriture des modèles.

Ces deux aspects — moteurs et langages — ne sont cependant pas indépendants : la représentation des connaissances utilisée va déterminer les processus qui vont affecter ces connaissances, et vice et versa [Magerko, 2007].

La nature des connaissances représentées dépend de l'approche choisie pour le système de scénarisation. Ainsi, un système adoptant une scénarisation intrinsèque, en intégrant directement la scénarisation au sein du processus de l'environnement virtuel pourra par exemple se baser sur une représentation explicite d'un **ensemble de scénarios**. Au contraire, un système optant pour une scénarisation extrinsèque, en apposant à un environnement virtuel existant une surcouche de scénarisation, nécessitera d'une part une représentation du **contenu scénaristique** présent dans la simulation, et d'autre part des connaissances lui permettant de déterminer le sous-espace de scénarios d'intérêt — des **objectifs scénaristiques**.

De même, la représentation utilisée est intrinsèquement liée aux techniques d'intelligence artificielle choisies pour le moteur. Par exemple, si le moteur de scénarisation repose sur des algorithmes de planification, alors le contenu scénaristique sera représenté sous la forme d'un ensemble d'opérateurs de planification, et le scénario généré sera représenté par un plan.

On distinguera ainsi trois types de langages :

– les langages de représentation de **contenu scénaristique**, qui représentent un espace de scénarios de manière implicite et distribuée,

1. Un **formalisme** est une symbolisation matérielle (textuelle, graphique, etc.) arbitraire permettant de rendre manipulables des concepts. Un **langage** est un système de signes doté d'une sémantique et d'une syntaxe. Un **langage formel** est un langage défini de sorte qu'il soit dénué d'ambiguïté, par opposition aux langages naturels. Un langage formel peut être spécifié par une grammaire formelle, une expression régulière, un automate, etc. Un langage formel repose sur un ou plusieurs formalismes. Ces deux notions sont souvent utilisées de manière interchangeable dans la littérature sur la représentation de connaissances pour les environnements virtuels. En règle générale, parler de formalisme renvoie à la notion de **manipulation**, tandis que parler de langage renvoie davantage à l'**expressivité**.

– les langages de représentation d'**objectifs scénaristiques**, qui représentent des connaissances permettant de déterminer un sous-espace de scénarios d'intérêt,
– les langages de représentation de **scénarios ou d'ensembles de scénarios**, qui représentent un espace de scénarios de manière explicite et centralisée.

3.1 Représentation de contenu scénaristique

La définition de l'ensemble des scénarios possibles nécessite tout d'abord de définir l'ensemble des éléments unitaires pouvant intervenir dans ces scénarios. Pour cela, il convient d'opter pour un formalisme de représentation du contenu scénaristique.
Selon les cas, la représentation de ce contenu pourra soit être divisée en représentation du domaine — objets, actions possibles sur ces objets, règles de fonctionnement des systèmes techniques, etc. — d'un côté et représentation de l'activité — comportement des personnages virtuels, procédures à suivre pour l'utilisateur, etc. — de l'autre, soit réunir ces deux aspects dans une représentation des événements possibles du scénario. Nous adoptons ici la définition suivante d'un événement :

Evénement

Un événement est un fait marquant, qui survient à un moment donné. Un événement peut correspondre à une agrégation de changements d'états liés de manière causale ou au maintien des valeurs d'un ensemble d'états sur une certaine durée, dès lors qu'ils sont signifiants du point de vue de l'observateur.

Ces événements peuvent ainsi regrouper des actions réalisées par plusieurs personnages ou groupes de personnages, et les réactions associées des systèmes techniques.
Selon l'approche adoptée par le moteur de scénarisation, ce contenu scénaristique jouera un rôle différent dans le processus de génération du scénario. Pour les systèmes faisant appel à une scénarisation extrinsèque, ce contenu sera intégré à l'environnement virtuel de départ, qui devra ensuite être cadré par le système de scénarisation (ex : [Gerbaud and Arnaldi, 2009]). Au contraire, pour les systèmes adoptant une approche émergente, ce contenu scénaristique constituera l'intégralité des connaissances utilisées par le système et les scénarios émergeront des différentes combinaisons possibles des éléments de ce contenu, en fonction des interactions entre personnages virtuels et utilisateurs (ex : [Charles et al., 2002]). Enfin, certains systèmes utiliseront ces éléments comme les briques de base des scénarios, qui seront générés en sélectionnant et ré-ordonnant dynamiquement les différents événements possibles, puis exécutés dans l'environnement virtuel (ex : [Mateas, 2002]).

3.1.1 Représentation du domaine

La représentation du domaine contient la définition des objets du domaine, des actions qui peuvent avoir lieu sur ces objets, et leurs effets sur le monde, qu'il s'agisse des actions effectuées par les personnages virtuels ou de celles proposées à l'utilisateur. Si les premiers environnements virtuels intégraient directement ces informations dans le code de la simulation, au même titre que les graphismes, de nombreux systèmes utilisent à présent des représentations formelles des systèmes techniques et de leurs règles de fonctionnement. On parle alors d'environnements virtuels informés (EVI). D'après [Donikian, 2004] :

> “ Un EVI est un environnement virtuel dont les modèles 3D contiennent non seulement la géométrie de la scène mais aussi toutes les informations pertinentes pour les entités comportementales à simuler, comme les éléments symboliques ou sémantiques qui peuvent permettre a ces entités de percevoir, décider et agir. ”

[Thouvenin, 2009] élargit cette définition afin de considérer les environnements virtuels dans lesquels sont intégrés des modèles de représentation de connaissances :

> “ Un EVI est un environnement virtuel doté de modèles à base de connaissance dans lequel il est possible à la fois d'interagir et de permettre des comportements par interprétation de représentations dynamiques ou statiques. ”

Les modèles associés aux environnements virtuels peuvent ainsi décrire les objets présents dans l'environnement, les actions possibles sur ces objets, les conditions nécessaires à la réalisation de ces actions, les effets de ces actions sur le monde, etc. Nous présentons ici les modèles et formalismes suivants : Smart Objects [Kallmann and Thalmann, 1999a], STARFISH [Badawi and Donikian, 2004], STORM [Mollet, 2005], VEHA [Chevaillier et al., 2009], COLOMBO [Edward et al., 2010], et la représentation Cause-and-Effect proposée par [Lugrin and Cavazza, 2006].

Smart Objects

Les Smart Objects [Kallmann and Thalmann, 1999a] [Kallmann and Thalmann, 1999b] visent à intégrer dans l'environnement les connaissances nécessaires à l'interaction avec ce dernier.
Les Smart Objects peuvent contenir quatre types d'information :

- les propriétés intrinsèques de l'objet, physiques ou sémantiques,
- les informations liées à l'interaction, généralement les positions où l'interaction est possible,
- les comportements de l'objet, exprimant les changements d'états des objets provoqués par les différentes actions,
- les comportements attendus de l'utilisateur, qui indiquent où et quand l'utilisateur peut manipuler l'objet.

La représentation des connaissances sur le domaine sous forme de Smart Objects est une représentation plutôt bas niveau, centrée autour de l'interaction 3D. Cette représentation ne permet pas d'exprimer des comportements complexes, et l'édition de comportements est difficile à maîtriser pour des non-informaticiens. Il s'agit cependant du modèle le plus utilisé, la plupart des environnements virtuels n'ayant pas besoin d'une telle complexité au niveau des comportements simulés.

STARFISH

Dans la lignée des Smart Objects, le modèle STARFISH (Synoptic Objects for Tracking Actions Received From Interactive Surfaces and Humanoids) [Badawi and Donikian, 2004] permet de décrire finement les surfaces d'interaction des objets géométriques, et d'associer à chaque sous-partie active d'un objet un automate comportemental décrivant l'interaction par le biais d'un ensemble d'actions élémentaires.
La description sous forme d'automates est plus intuitive que celle proposé par le modèle des Smart Objects. Comme pour les Smart Objects, le niveau de description de l'interaction est très fin, mais l'utilisation d'automates limite la complexité des comportements qu'il est possible de décrire de manière lisible.

```
COMMANDS
# name          action          part     ini    end    inc
cmd_open_door   translation1    part1    0.00   1.00   0.05
cmd_open_door   translation2    part2    0.00   1.00   0.05
cmd_close_door  translation1    part1    1.00   0.00   0.05
cmd_close_door  translation2    part2    1.00   0.00   0.05
END # of commands

VARIABLES
  open          0.0
  passing       0.0
END # of variables
BEHAVIOR go_out1
  Subroutine
  addvar      passing  1.0
  gotopos     pos_out1
  addvar      passing -1.0
END # of behavior

BEHAVIOR open_door
  Subroutine
  checkvar    open     0.0
  changevar   open     1.0
  docmd       cmd_open_door
END

BEHAVIOR enter1
  Subroutine
  gotopos    pos_enter1
  dobh       open_door
  dobh       go_out1
  dobh       close_door
END # of behavior

BEHAVIOR enter
  Dobhifnear enter1 pos_enter1
  Dobhifnear enter2 pos_enter2
END # of behavior
```

```
BEHAVIOR go_out2
  Subroutine
  addvar      passing  1.0
  gotopos     pos_out2
  addvar      passing -1.0
END # of behavior

BEHAVIOR close_door
  Subroutine
  checkvar    open     1.0
  checkvar    passing  0.0
  changevar   open     0.0
  docmd       cmd_close_door
END

BEHAVIOR enter2
  Subroutine
  gotopos    pos_enter2
  dobh       open_door
  dobh       go_out2
  dobh       close_door
END # of behavior
```

FIGURE 3.1 – Comportements liés à un Smart Object de type porte [Kallmann and Thalmann, 1999b]

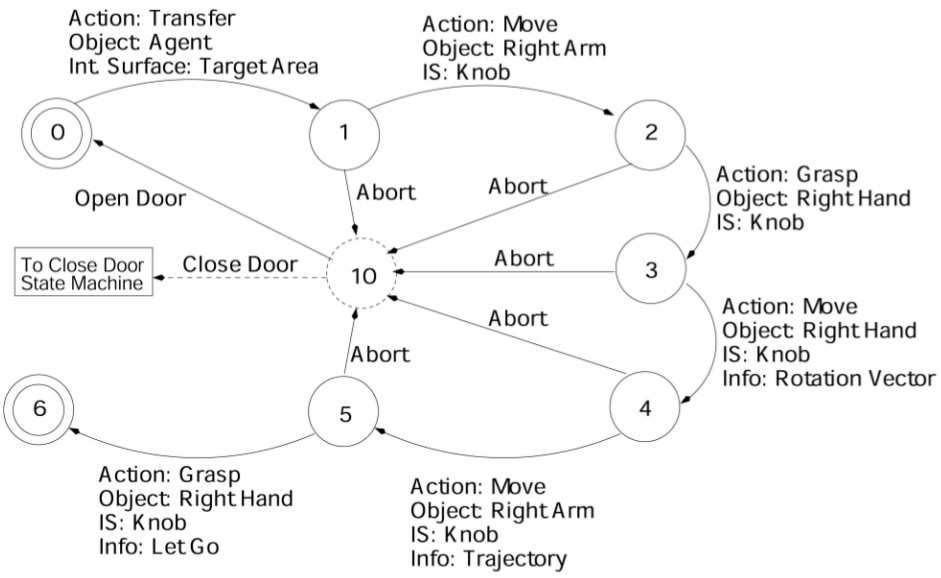


FIGURE 3.2 – Machine à état représentant le comportement d'une porte [Badawi and Donikian, 2004]

STORM

Le modèle STORM (Simulation and Training Object-Relation Model) [Mollet, 2005] [Mollet and Arnaldi, 2006] est lui aussi inspiré des Smart Objects. Il étend ces derniers en permettant de définir des interactions non plus seulement entre un agent (utilisateur ou personnage virtuel) et un objet, mais également entre plusieurs objets. Les relations entre deux objets étant elles-aussi considérées comme des objets, une interaction peut concerner plus de deux objets physiques, introduisant ainsi la notion d'outils.
Le modèle STORM associe un modèle d'objets comportementaux et un modèle d'interaction. Les objets définis dans le modèle d'objets comportementaux sont dotés de capacités (ex : emboîtable), qui permettent de déterminer les interactions dont ils peuvent être la cible (voir figure 3.3). Les relations définies dans le modèle d'interaction se réfèrent à ces capacités pour désigner les objets qu'elles permettent de lier (voir figure 3.4).

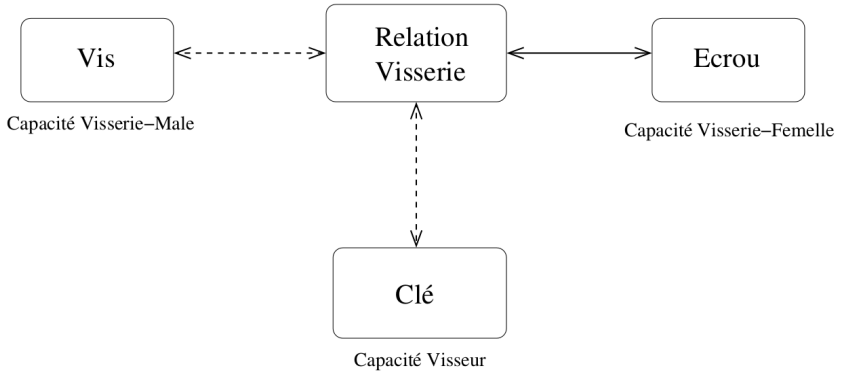


FIGURE 3.3 – Relation STORM relative à la visserie [Mollet, 2005]

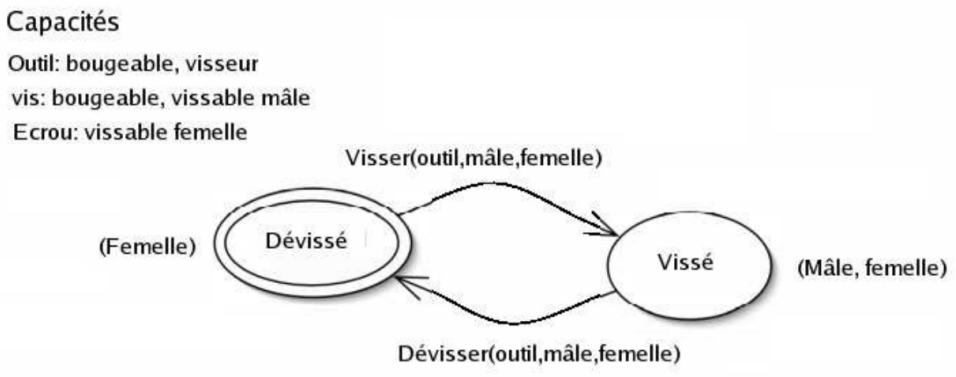


FIGURE 3.4 – Comportement associé à la relation de visserie [Mollet, 2005]

Ainsi, par rapport aux modèles présentés précédemment, le modèle STORM introduit de l'abstraction dans la représentation des objets. Cependant, les comportements eux-mêmes ne sont pas définis en suivant le même méta-modèle, mais sont représentés de manière externe, généralement grâce à des automates. Il n'est donc là encore pas possible de raisonner sur les actions pour tenter d'aboutir à un état particulier de la simulation.

VEHA

VEHA (Virtual Environment supporting Human Activities) [Chevaillier et al., 2009] est un méta-modèle d'environnement informé et structuré, défini comme une extension d'UML 2.1. Les auteurs

ont pour volonté affichée de permettre la création de modèles du domaine indépendants des moteurs utilisant ces modèles, en modélisant à la fois les propriétés sémantiques, structurelles, géométriques et topologiques des entités de l'environnement virtuel, ainsi que leurs comportements réactifs. L'objectif de VEHA est de permettre d'interroger ces différentes propriétés en ligne, et de les rendre directement utilisables pour simuler l'évolution de l'environnement virtuel.

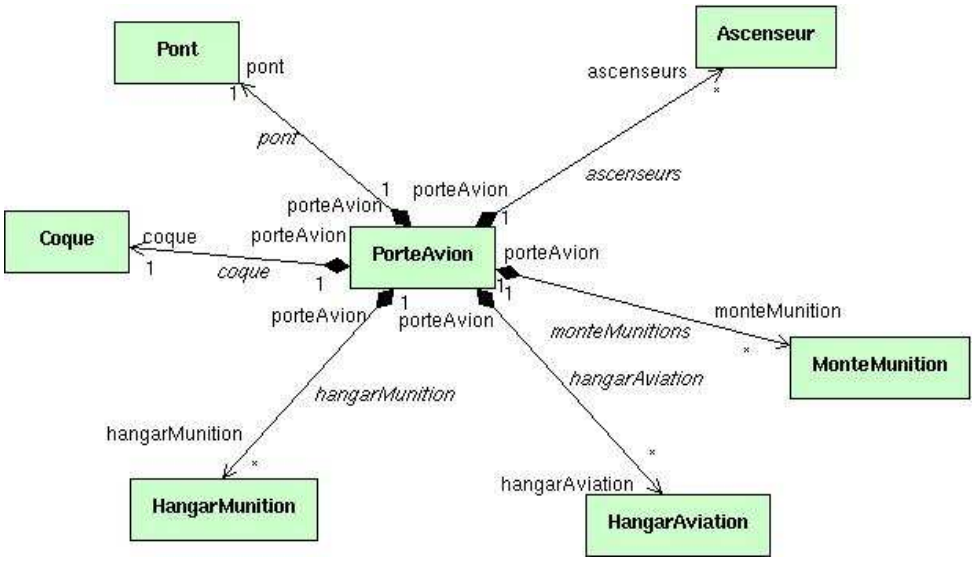


FIGURE 3.5 – Modèle d'un porte avions [Chevaillier et al., 2009]

La modélisation du comportement des entités est réalisée par le biais d'automates finis, représentés comme des machines à états comportementales UML. Un exemple de machine à état est présenté dans la figure 3.6. D'après les auteurs, cette explicitation est non seulement exécutable mais permet aussi aux agents d'analyser le comportement des objets et d'anticiper leur évolution. Cependant, ici encore l'utilisation d'automates limite la complexité des comportements modélisables, ainsi que les possibilités de raisonnement sur ces comportements. Les comportements spécifiques, trop complexes pour être représentés par des machines à états, sont d'ailleurs encodés sous la forme d'*opaque behaviors*, qu'il est possible d'exécuter mais pas d'interroger sur leurs transitions, préconditions, etc.

COLOMBO

Le modèle COLOMBO (Création Ontologique Liée à la MOdélisation des OBjets) [Edward et al., 2010] [Edward, 2011] a un positionnement proche de celui de VEHA.

COLOMBO s'appuie sur une représentation ontologique de l'environnement pour décrire les objets, leurs propriétés, et les actions pouvant s'appliquer en fonction de ces propriétés. L'ontologie est décrite en langage MOSS, au sein d'un environnement de développement lié à la plateforme multi-agent OMAS [Barthès and Ramos, 2002]. Un extrait de cette ontologie est présenté dans la figure 3.7.

Pour modéliser le comportement des objets lié aux actions, COLOMBO utilise un système de règles, elles-aussi représentés en MOSS, donc inspectables — c'est à dire qu'il est possible de les exécuter mais également de raisonner sur leur code. Ces règles sont soit des règles d'exécution de l'action (voir figure 3.8), qui définissent les conditions dans lesquelles l'action peut être déclenchée, soit des règles de transition (voir figure 3.9), qui déterminent si les effets de l'action peuvent être appliqués au niveau de l'objet ou si celle-ci est en échec. Ces règles permettent non seulement d'exécuter les com-

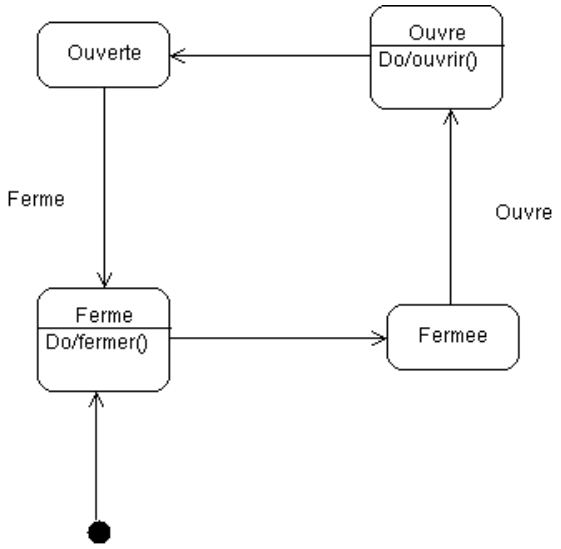


FIGURE 3.6 – Comportement d'un déflecteur [Chevaillier et al., 2009]

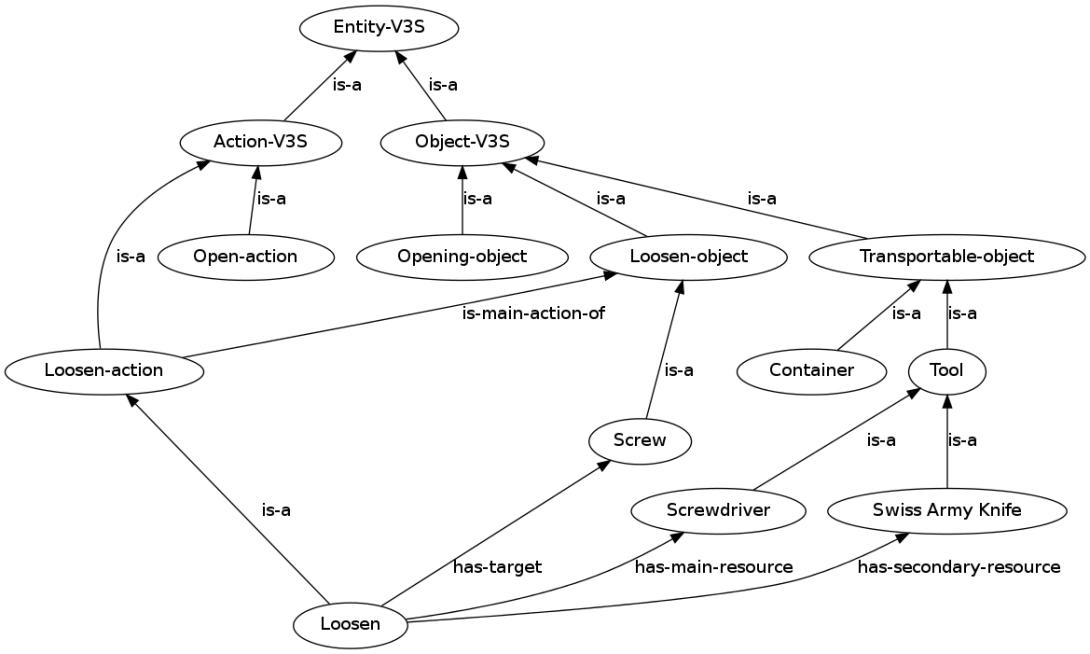


FIGURE 3.7 – Extrait de l'ontologie COLOMBO [Edward et al., 2010]

portements mais aussi de raisonner dessus : les personnages virtuels peuvent ainsi planifier leurs séquences d'actions en fonction des conditions nécessaires au déroulement correct des différents comportements des objets.
Si le méta-modèle ontologique et l'utilisation des démons dans COLOMBO sont intéressants, son implémentation manque toutefois de généricité, la modélisation se plaçant parfois au niveau des instances d'objets plutôt que des concepts. Par ailleurs, l'utilisation exclusive de prédicats au détriments de valeurs numériques est pénalisante lorsqu'il s'agit de représenter des valeurs continues pour la modélisation de systèmes techniques complexes.

```
(defconcept ( :en "connect-loading-arm" :fr "connecter-bras-chargement")
          ( :is-a "action-V3S")
      ( :rel "objet-cible" ( :default "f-bras-chargement"))
      ( :rel ( :en "connect-to" :fr "connecter-a") ( :to "objet-V3S"))
      ( :att "rules" ( :default
                       ( :if ( ?* "f-bras-chargement" ("statut" :is "normal"))
                       ( ?* "f-bras-chargement" ("a-etat" :is-not "connecte"))
                        :then
                       ( :return :success))
      ))
      ( :att ( :en "code" :fr "code") ( :default 143))
)
```

FIGURE 3.8 – Règle d'exécution d'une action en MOSS [Edward, 2011]

```
(defconcept ( :en "f-loading-arm" :fr "f-bras-chargement")
          ( :is-a "objet-transportable")
      ( :att ( :en "e-length" :fr "f-longueur"))
      ( :att ( :en "e-diameter" :fr "f-diametre"))
      ( :att ( :en "type" :fr "type") ( :default "cognitif"))
      ( :rel ( :en "connect" :fr "connecter") ( :to "vanne-pied"))
      ( :rel ( :en "hilt" :fr "poignee") ( :to "f-poignee"))
      ( :rel ( :en "action" :fr "mv-action")
            ( :default "connecter-bras-chargement" "deconnecter-bras-chargement"))
      ( :att ( :en "rules" :fr "regles")
            ( :default ( :if ( ?* "f-bras-chargement" ("statut" :is "normal"))
            ( ?* "f-poignee" ("a-etat" :is "tourne"))
            ( ?* "vanne-pied" ("a-etat" :is "verouille"))
             :then
            ( :return :success))))
      ( :doc :fr "Le bras de chargement est un dispositif permettant de faire
             le chargement des produits dans les cuves du camion.")
)
```

FIGURE 3.9 – Règle de transition d'un object en MOSS [Edward, 2011]

Cause-and-Effect

[Lugrin and Cavazza, 2006] proposent un moteur de causalité (*causal engine*) destiné à écraser les décisions du moteur physique en déclenchant des conséquences physiques alternatives pour les différentes actions de l'utilisateur ou des personnages virtuels.
Les actions sont représentées dans un formalisme dénommé CE, pour Cause-and-Effect, qui associe un ensemble de préconditions à un ensemble d'effets. Chaque action est associée à un événement de base, qui permet de la déclencher. Ces événements peuvent correspondre à une interaction de l'utilisateur ou d'un personnage virtuel (par exemple USE ou GRASP), ou à des événements physiques non-intentionnels (par exemple HIT ou TOUCH). A chaque événement peuvent ainsi correspondre plusieurs actions ; si les préconditions de plus d'une action sont vérifiées en même temps, cela signifie qu'il est possible pour le système de déclencher alternativement les effets de l'une ou de l'autre. Les conditions et effets des actions font référence à des propriétés des objets, qui sont définies dans une ontologie des objets. Un exemple d'action et la représentation de l'objet associé sont présentés dans la figure 3.10.

De par l'association d'une représentation ontologique des objets et d'une représentation inspectable des actions, ce formalisme est assez proche de COLOMBO et de VEHA. L'association de pré-

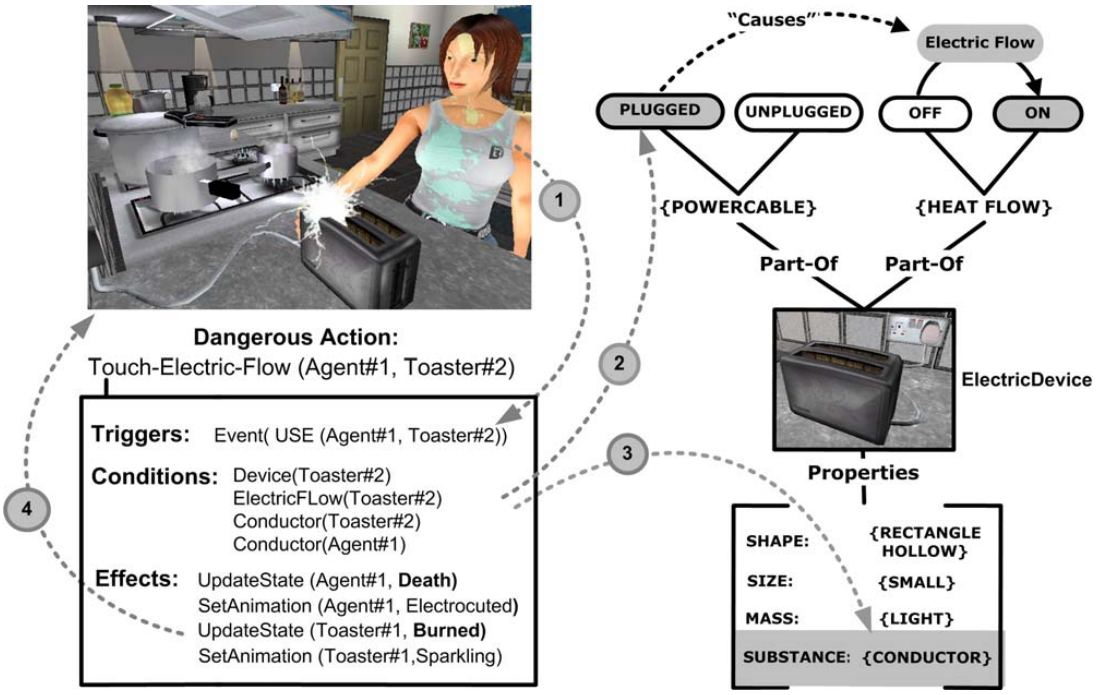


FIGURE 3.10 – Représentation d'une action et d'un objet dans [Lugrin and Cavazza, 2006]

conditions et d'effets pour chaque action, à la manière d'opérateurs de planification, permet non seulement d'utiliser cette représentation pour exécuter les effets des actions mais aussi à des fins de raisonnement, notamment en utilisant un moteur de planification pour calculer des suites d'actions aboutissant à une situation donnée. Cependant, l'absence d'alternatives au niveau des conditions, et la représentation des conditions et des effets sous formes de "blocs" indivisibles rendent nécessaire une duplication des actions pour représenter la moindre variation de comportement, et rendent fastidieuse la représentation de comportements complexes.
Dans les premiers formalismes présentés (Smart Objects, STARFISH, STORM), les comportements ne sont modélisés que par rapport aux actions, et sont déclenchés de manière réactive ; il n'y a pas de possibilité de modéliser l'évolution du système indépendamment des actions des personnages, par exemple le fait qu'un robinet laissé ouvert provoque le remplissage progressif d'un contenant, menant éventuellement à son débordement. De tels comportements ne peuvent être représentés en VEHA que grâce à des *opaque behaviors*, non inspectables, ou en COLOMBO par des méthodes *ad-hoc*. Ici, les comportements peuvent être déclenchés en réaction à des actions de l'utilisateur ou des personnages virtuels, mais aussi en réaction à des événements physiques. Il est donc possible de modéliser des chaînes de comportements ne faisant pas appel aux actions d'un agent. Cependant, cette modélisation nécessite l'ajout d'un moteur physique en parallèle. Les comportements gérés par la physique correspondent à des "boîtes noires" : il n'est pas possible, au niveau du raisonnement, de faire le lien entre les effets d'une action et les préconditions d'une autre si ce lien passe par des événements physiques. Il n'est pas non plus possible de modéliser des systèmes complexes ne faisant pas appel à la physique (systèmes électroniques par exemple).

3.1.2 Représentation de l'activité

S'il est nécessaire de représenter le domaine et ses règles de fonctionnement pour exécuter la simulation ou générer différents scénarios, cela ne suffit pas pour autant à assurer la validité écologique de ces scénarios. Il faut pour cela s'assurer également que le comportement des personnages virtuels qui peuplent l'environnement, et dont les choix ne sont pas toujours rationnels par rapport aux règles des systèmes techniques, soit conforme à la réalité.

Il va ainsi s'agir de représenter non seulement l'**activité prescrite** — les procédures que doivent suivre les personnages virtuels et les utilisateurs pour assurer le fonctionnement nominal du système simulé ou pour réagir en cas de défaillance — mais également l'**activité observée**, en intégrant au modèle les différents facteurs qui rentrent en jeu dans leurs mécanismes de prise de décision des agents, les erreurs qu'ils peuvent commettre, les déviations qui peuvent être relevées sur le terrain, etc. On cherche ainsi à tendre vers la modélisation de l'**activité finalisée**.
Ces langages permettent ainsi de représenter l'activité d'un personnage virtuel (ou d'un groupe de personnages ayant le même comportement, assimilable à un personnage unique), dans le but de générer son comportement. Ils peuvent aussi être utilisés pour analyser et prédire les actions de l'utilisateur de l'environnement virtuel.
Une fois encore, il est délicat de séparer représentation des connaissances et moteurs, puisque les connaissances représentées dans ces modèles sont liées à la manière dont elles vont être interprétées, selon que le système utilise une architecture cognitive donnée, que les personnages virtuels soient autonomes ou non, qu'ils sélectionnent leurs actions de manière réactive ou en utilisant de la planification, etc. De même, on trouve dans certains cas des modèles d'activité qui englobent directement le modèle du domaine associé, en intégrant dans la représentation des actions à la fois les facteurs motivationnels qui jouent sur la prise de décision du personnage, et les effets de l'action sur le monde [2].

Opérateurs de planification

L'approche la plus courante est de représenter l'activité sous la forme d'un ensemble d'opérateurs de planification, chaque opérateur correspondant à une action réalisable par un personnage virtuel [Charles et al., 2003], [Young et al., 2004], [Lugrin and Cavazza, 2006], [Pizzi and Cavazza, 2007], [Chang and Soo, 2008], [Riedl and Young, 2010], [Swartjes, 2010], [Porteous et al., 2010], [Porteous et al., 2013]. Ces opérateurs représentent les actions comme un ensemble de préconditions et d'effets, ou postconditions.

Action: **slay** (?slayer, ?monster, ?place)
actors: ?slayer
constraints: knight(?slayer), monster(?monster), place(?place)
precondition: at(?slayer, ?place), at(?monster, ?place), alive(?slayer), alive(?monster)
effect: ¬alive(?monster)

Action: **marry** (?groom, ?bride, ?place)
actors: ?groom, ?bride
constraints: male(?groom), female(?bride), place(?place)
precondition: at(?groom, ?place), at(?bride, ?place), loves(?groom, ?bride),
loves(?bride, ?groom), alive(?groom), alive(?bride)
effect: married(?groom), married(?bride), ¬single(?groom), ¬single(?bride),
married-to(?groom, ?bride), married-to(?bride, ?groom)

FIGURE 3.11 – Exemples d'opérateurs dans [Riedl and Young, 2010]

Il existe un certain nombre de langages permettant d'exprimer ces opérateurs. L'un des plus répandus est le langage STRIPS (dont l'acronyme est issu du Stanford Research Institute Problem Solver, le moteur de planification qui lui était associé) [Fikes and Nilsson, 1971]. Un exemple d'opérateur STRIPS est présenté dans la figure 3.12.

Le langage STRIPS étant considéré comme n'étant pas suffisamment expressif, de nombreux travaux utilisent à présent le langage PDDL (Planning Domain Definition Language) [McDermott et al., 1998],

2. Il ne s'agit cependant pas ici de chercher à représenter de scénarios dans leur globalité, puisque ces derniers seront issus de la combinaison de l'activité de plusieurs personnages. Les langages utilisés pour décrire les scénarios impliquant de la coactivité au niveau global sont décrits dans la partie 3.3.

```
(def-operator: use-rachels-pda
      (make-operator
            :pre-conditions '(:need-gift-idea)
            :exe-condition :pda-free
            :add-list '(:info-gift-flowers)
            :delete-list '(:need-gift-idea)))

(def-operator: purchase-flowers
      (make-operator
            :pre-conditions '(:info-gift-flowers :at-shop :has-money)
            :exe-condition :flowers-available
            :add-list '(:has-flowers)
            :delete-list '(:has-money)))
```

FIGURE 3.12 – Opérateurs STRIPS [Charles et al., 2003]

une syntaxe standard permettant l'échange et la comparaison des résultats lors des compétitions internationales de planification. Un exemple d'opérateur PDDL est présenté dans la figure 3.13.

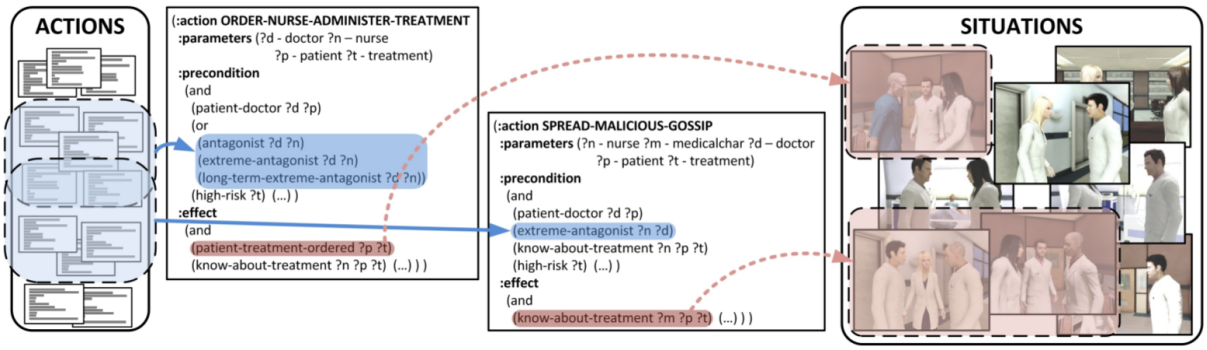


FIGURE 3.13 – Opérateurs PDDL [Porteous et al., 2013]

Les actions peuvent correspondre soit à des actions physiques, soit à des processus internes aux personnages qui vont provoquer des changements d'états mentaux. C'est le cas par exemple dans le système EmoEmma [Pizzi and Cavazza, 2007], qui vise à recréer la complexité émotionnelle des personnages du roman *Emma Bovary*. Un opérateur est présenté dans la figure 3.14.

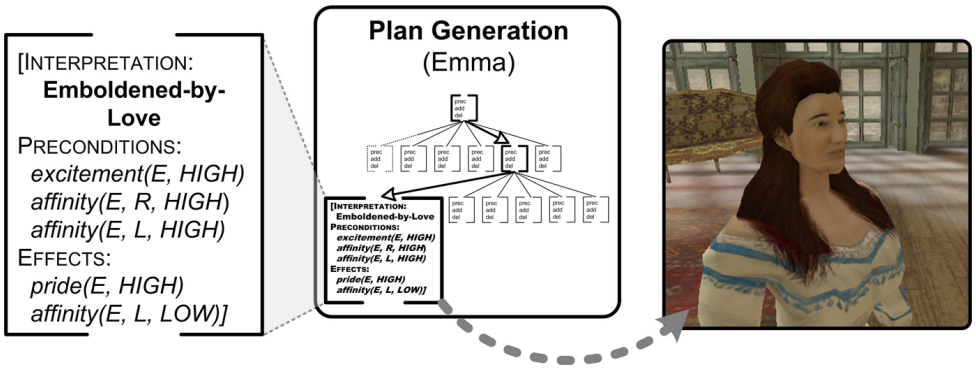


FIGURE 3.14 – Opérateur de planification émotionnel dans EmoEmma [Pizzi and Cavazza, 2007]

Les préconditions peuvent concerner les états du monde, mais aussi les états internes des personnages virtuels, comme leurs motivations, voire des états spécifiques au système de scénarisation. Dans [Porteous et al., 2010], le scénario généré dépend du point de vue adopté pour le discours, selon que l'on se focalise sur un personnage ou un autre, et les préconditions spécifient ainsi le point de

vue nécessaire pour que chaque opérateur soit valide.
Le principal avantage de cette représentation est qu'elle est directement interprétable par des moteurs de planification, permettant de générer des plans soit au niveau des personnages individuels (le plan correspondant aux actions choisies par le personnage pour réaliser son ou ses buts), soit au niveau du scénario global. Les préconditions permettent de cadrer l'activité en définissant des contraintes sur la séquentialité des actions ou sur les motivations et états internes des personnages nécessaires à la réalisation de l'action.
Cependant, il est très difficile d'avoir une vue globale des comportements représentés, du fait de l'absence de structure de l'ensemble des opérateurs. Plus le nombre de facteurs pris en compte augmente et plus les préconditions seront complexes, rendant difficile le passage à l'échelle et la maintenabilité de la représentation.

Règles

Similairement, certains systèmes utilisent des représentations basées sur des ensembles de règles logiques, composées de conditions et de conséquences sous la forme `SI` x `ALORS` y.
Le système IDtension [Szilas, 2003] [Szilas, 2007] utilise un ensemble de règles logiques afin de déterminer l'applicabilité des actions. Le système LOGTELL [Ciarlini et al., 2005] s'en sert pour représenter les effets des événements sur les motivations des personnages. Le modèle Brahms décrit dans [Sierhuis et al., 2003] utilise des scripts de la forme `when` x `do` y, assimilables à des règles.

IF
CAN (x , t , p)
KNOW(x , CAN (x , t , p))
KNOW(y , CAN (x , t , p))
~ KNOW (y , WANT(x , t , p))
~ KNOW (y , HAVE_BEGUN (x , t , p))
~ KNOW (y , HAVE_FINISHED(x , t , p))
x ≠ y
THEN
Incite (y, x , t , p)

FIGURE 3.15 – Règle de sélection de l'action d'incitation dans IDtension [Szilas, 2003]

Les représentations à base de règles sont confrontées au même écueil que les représentations basées sur des ensembles d'opérateurs de planification : elles deviennent difficiles à maintenir dès lors qu'il s'agit de prendre en compte de nombreux facteurs dans les processus décisionnels des personnages virtuels.

Réseaux de tâches hiérachiques

L'utilisation de réseaux de tâches hiérarchiques (HTN, pour Hierarchical Task Networks) permet d'introduire de la structure, en hiérarchisant les actions par rapport aux buts qu'elles permettent de satisfaire. Cette représentation a notamment été utilisée par [Cavazza et al., 2002] pour créer un environnement virtuel à la scénarisation émergente basée sur la série Friends.

Un réseau de tâches hiérarchique est défini pour chaque personnage, et décompose les buts de haut niveau du personnage (ex : "Demander un rendez-vous à Rachel") en buts de niveaux inférieurs (ex : "Apprendre ce que Rachel aime"), jusqu'au niveau des actions (ex : "Prendre le journal intime de Rachel"). Plusieurs méthodes peuvent être définies pour un même but, ces méthodes se décomposant elles-mêmes en sous-buts, ce qui permet de représenter des alternatives.

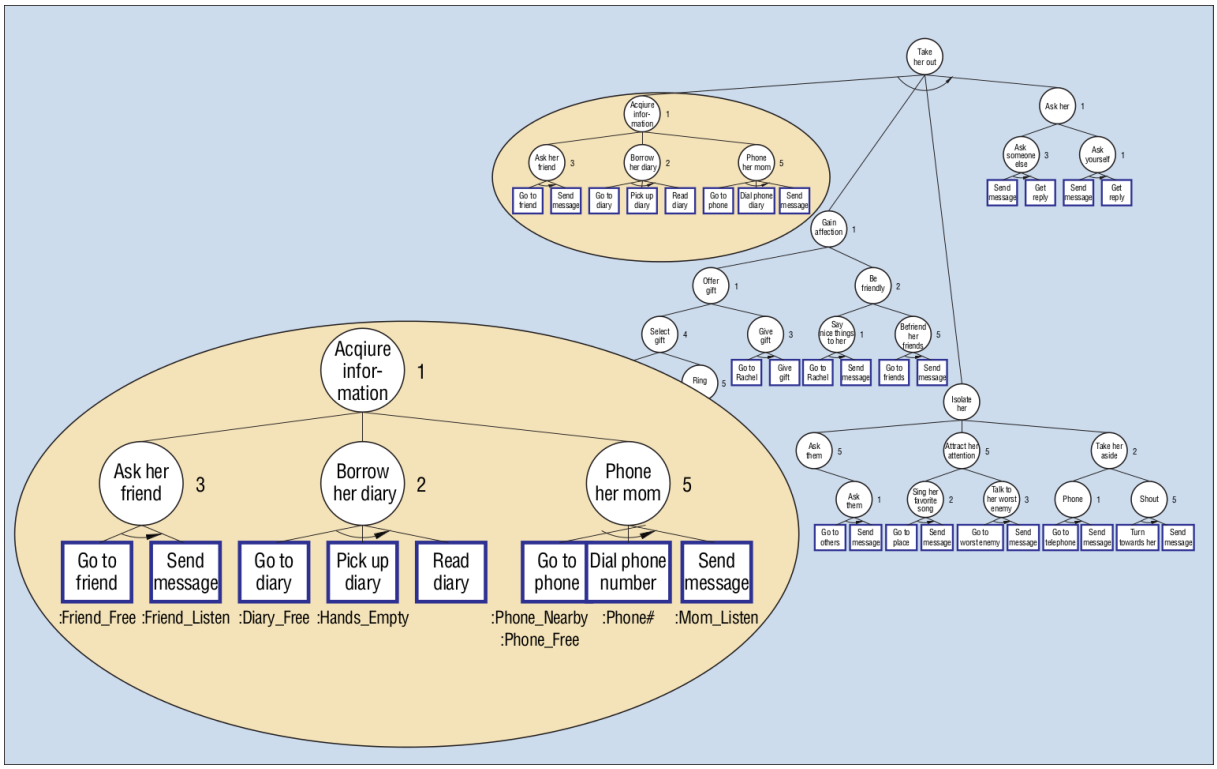


FIGURE 3.16 – Exemple de réseau de tâches hiérarchique [Cavazza et al., 2002]

La représentation de l'activité par un réseau de tâches hiérarchique nécessite qu'il n'y ait ni interactions négatives entre les tâches (une tâche annule les effets d'une tâche précédente) ni interactions positives (une tâche réalise les effets d'une tâche suivante, supprimant ainsi la nécessité d'effectuer cette dernière) [Charles et al., 2003].
Il est possible de rajouter des connaissances permettant la prise de décision au niveau d'une alternative entre plusieurs méthodes, cependant cela n'est pas prévu dans le formalisme à la base. [Cavazza et al., 2002] disent utiliser des modèles de préférences d'action annexes, qui prennent en compte le statut émotionnel des personnages, mais ne détaillent pas ces modèles.
Le modèle utilisé pour STEVE utilise lui aussi une décomposition hiérarchique de l'activité, représenté cette fois sous la forme de règles de production SOAR [Rickel and Johnson, 1998]. La décomposition hiérarchique est associée à un ensemble de contraintes temporelles et de liens de causalité entre actions.

Automates finis

Plutôt que de modéliser la logique de sélection des actions sous forme de préconditions et post-conditions, il est possible de représenter directement les transitions entre actions au niveau des comportements individuels. On peut utiliser pour cela des automates finis ou des réseaux de Petri [Balas et al., 2008] [Delmas, 2009].
Les automates finis définissent les différents enchaînements possibles entre les actions d'un personnage ou d'un groupe de personnages. Lorsque le nombre d'actions et le nombre de transitions augmentent, les automates classiques deviennent rapidement illisibles, et il est alors nécessaire d'utiliser un ensemble d'automates dont il faut définir l'exécution coordonnée. Cependant, là encore, il est difficile d'avoir une vue d'ensemble du comportement.
Les automates parallèles et hiérarchiques HPTS (Hierarchical Parallel Transition System) [Donikian, 2001] permettent d'intégrer la notion de but en modélisant l'activité à plusieurs niveaux de hiérarchie, et de simuler des activités simultanées via l'exécution parallèle de plusieurs automates. HPTS++

est une extension orientée objet de HPTS, qui permet de gérer la notion de priorités, de ressources et de degrés de préférence [Lamarche and Donikian, 2002].

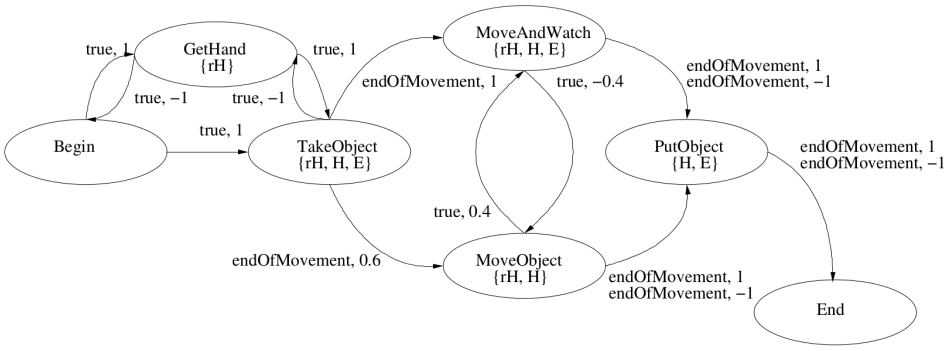


FIGURE 3.17 – Comportement de déplacement d'un objet modélisé en HPTS++ [Lamarche and Donikian, 2002]

Les automates finis et les réseaux de Petri permettent de contrôler très précisément le comportement des personnages. La décomposition hiérarchique et l'utilisation de plusieurs modèles parallèles simplifie la tâche de création des modèles, mais l'effort a fournir reste important car il est nécessaire d'expliciter toutes les transitions possibles. De plus, il s'agit de formalismes informatiques qui ne sont pas renseignables directement par des non-informaticiens.

Diagrammes d'activité UML

HAVE (Human Activities in a Virtual Environment) [Marion, 2010] est le méta-modèle d'activité utilisé dans la plateforme MASCARET, associé au méta-modèle d'environnement VEHA décrit dans la section 3.1.1. Les procédures métier y sont représentées sous la forme de diagrammes d'activité UML, qui décrivent les différents enchaînements d'actions devant être effectués par chaque agent, avec des points de synchronisation entre agents. A ce méta-modèle d'activité est associé un second méta-modèle, BEHAVE, permettant de décrire la structure organisationnelle et les rôles des différents agents. A chaque rôle est associé un ensemble d'activités du modèle HAVE.

HAVE est utilisé pour représenter la procédure prescrite, mais pas l'activité en mode dégradé (erreurs, compromis...), et est ainsi destiné plutôt au suivi de l'activité de l'apprenant qu'à la génération des comportements des personnages virtuels (*"libre à eux de faire autre chose"* [Marion, 2010]). L'absence de représentation hiérarchique et la nécessité de définir explicitement toutes les transitions entre actions et tous les points où une alternative se présente rend cette représentation très proche des automates finis classiques.

Langages ergonomiques

Il existe de nombreux langages dans le domaine de l'ergonomie qui permettent de représenter l'activité humaine [Couix and Burkhardt, 2011]. Contrairement aux formalismes informatiques, ces langages ne limitent pas la représentation en forçant à considérer l'humain comme un système de résolution de problèmes. Ils ont de fait l'avantage de pouvoir être utilisés par des non-informaticiens, cependant ils sont pour la plupart des formalismes textuels ou graphiques qui ne peuvent être directement interprétés par des modules informatiques. Nous détaillerons ici deux de ces langages qui ont été opérationnalisés : CTT et HAWAI-DL.

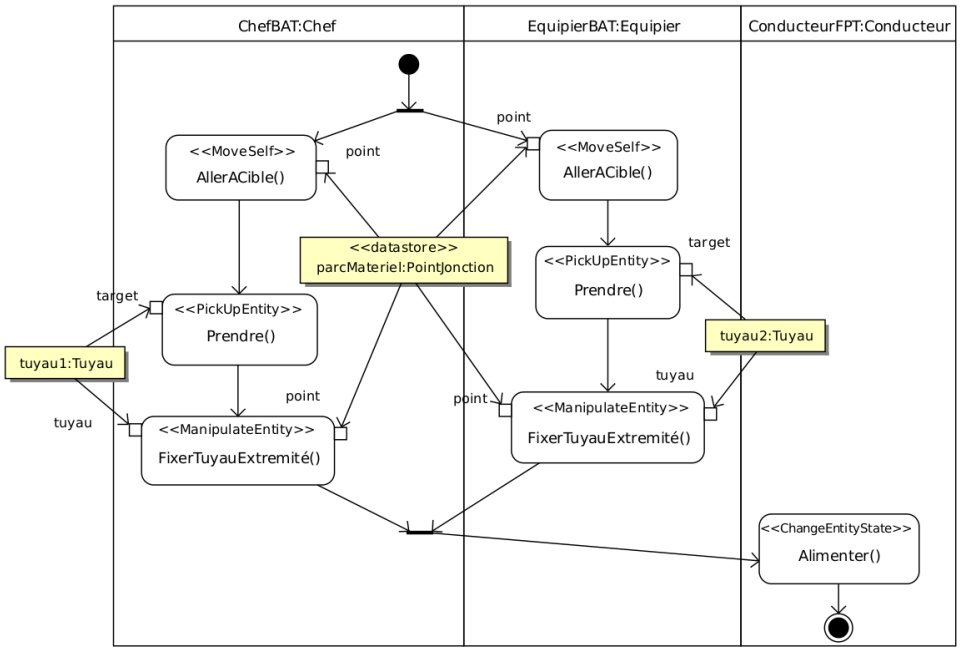


FIGURE 3.18 – Exemple de modèle d'activité HAVE [Marion, 2010]

CTT La notation CTT (ConcurTaskTrees) [Paternò, 2003] est un formalisme graphique qui s'interface avec des modèles de domaine représentés en UML. CTT représente les tâches de manière hiérarchique, les tâches d'un même niveau étant ordonnées grâce à des opérateurs exprimant la séquentialité ou encore l'alternative. CTT a pour but de modéliser la façon dont l'utilisateur d'un système se représente mentalement la tâche a réaliser, afin d'y faire correspondre le mieux possible la façon dont la tâche est implémentée dans le système.
Bien que CTT ait été conçu à l'origine pour servir de support à la conception d'interfaces hommes-machines, il a été utilisé dans [Vidani and Chittaro, 2009] pour représenter les procédures dans un environnement virtuel pour l'apprentissage humain, dans une optique de suivi de l'apprenant et de génération dynamique de menus. Cependant à notre connaissance, CTT n'a jamais été utilisé pour de la génération de comportements de personnages virtuels.

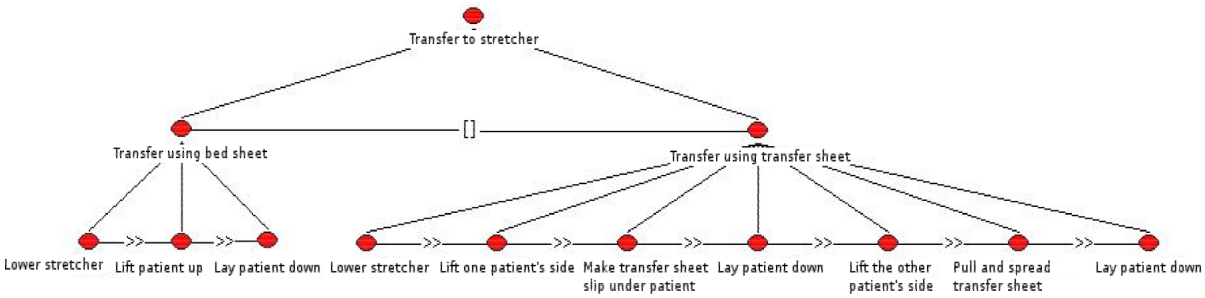


FIGURE 3.19 – Extrait de procédure en CTT contenant un choix [Vidani and Chittaro, 2009]

En tant que tel, CTT permet de suivre le choix d'un utilisateur au niveau d'une alternative, mais pas d'évaluer quelle serait la décision prise par un personnage virtuel. La prise en compte du contexte n'est en effet pas native dans CTT, et il serait nécessaire pour cela soit d'utiliser une extension comme les Contextual ConcurTaskTrees [Van den Bergh and Coninx, 2004], soit de le représenter via l'ajout de préconditions et postconditions spécifiques. De plus, un point soulevé par [Vidani and Chittaro, 2009] est que CTT ne permet pas tel quel la modélisation explicite des erreurs et l'expression de préférences quant aux différentes manières de réaliser une tâche.

HAWAI-DL HAWAI-DL (Human Activity and Work Analysis for sImulation-Description Language) [Amokrane et al., 2008] [Edward et al., 2008] est un langage de modélisation de l'activité inspiré de MAD* [Sebillotte and Scapin, 1994] et GTA [Van Der Veer et al., 1996], qui fait suite aux travaux sur le langage METISSE [El-Kechaï, 2007]. HAWAI-DL utilise une représentation hiérarchique des tâches, un ensemble de constructeurs permettant d'ordonner les sous-tâches entre elles, et différents types de préconditions spécifiques afin de déterminer les conditions dans lesquelles une tâche peut être exécutée ou bien sera favorisée par rapport à une autre. Les tâches référencent les objets et actions du modèle de domaine défini grâce à COLOMBO (voir section 3.1.1).

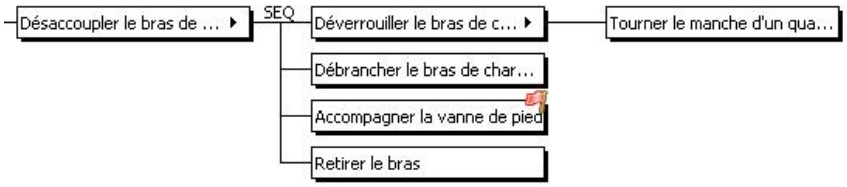


FIGURE 3.20 – Extrait de modèle d'activité HAWAI-DL.
Le drapeau rouge indique une tâche ALU, qu'il est toléré d'ignorer sous certaines conditions.

De même que CTT, HAWAI-DL vise a représenter l'activité telle que l'agent humain se la représente, mais tandis que CTT représente l'activité prescrite, HAWAI-DL est conçu pour représenter l'activité observée, telle qu'elle a lieu sur le terrain. HAWAI-DL intègre ainsi la notion d'Actions Limites tolérées par l'Usage (ALU) [Garza and Fadier, 2007]. Les ALU sont un concept issu de travaux en psychologie cognitive, traduisant le résultat d'un compromis local et souvent informel entre les acteurs du domaine. Par exemple, travailler avec un produit chimique sans équipements de protection individuelle, dans le but de gagner du temps, peut exister dans certains cas en tant qu'ALU, c'est-à-dire pratique à risque tolérée dans l'usage.
HAWAI-DL a été utilisé avec succès dans plusieurs environnements virtuels, à la fois pour générer le comportements de personnages virtuels [Edward, 2011] et pour suivre et analyser le comportement de l'utilisateur [Amokrane, 2010]. Cependant le langage souffre d'un manque de formalisation, à la fois au niveau des constructeurs temporels et des différents types de préconditions, introduisant des redondances et des imprécisions dans la modélisation.

3.1.3 Représentation des événements

Tous les systèmes ne font cependant pas de distinction entre ce qui fait partie de la représentation du domaine et ce qui fait partie de la représentation de l'activité. Certains, notamment ceux dont la scénarisation est intrinsèque, utilisent comme élément de base des représentations d'événements atomiques, qui mettent en jeu les actions de l'ensemble des personnages et leurs effets sur le monde.

ABL — Façade

Le récit de Façade [Mateas, 2002], [Mateas and Stern, 2005] repose sur une sélection dynamique parmi un ensemble de *story beats*.
Chaque *beat* correspond à une unité de récit et représente un rebondissement pouvant prendre place pendant la simulation. Ils sont dotés de préconditions et de postconditions permettant de déterminer

dans quelles situations ils peuvent s'appliquer, et comment ils vont impacter le récit en termes de relations entre les personnages ou d'intérêt narratif.
Ces éléments sont décrits en ABL (A Behavior Language) [Mateas, 2002], un langage de planification réactive à la structure proche du Java et qui permet de synchroniser les animations et dialogues des différents personnages. Chaque *beat* comprend entre 10 et 100 comportements (*behaviors*), séparés en plusieurs sous-ensembles qui seront déclenchés ou non selon les actions de l'utilisateur. La version finale de Façade contient 27 beats différents (ex : *ArgueOverItalyVacation, ExplainDatingAnniversary*).
ABL a également été utilisé dans IN-TALE [Riedl et al., 2008b] pour représenter des modèles de l'activité proches de réseaux de tâches hiérarchiques, et dans différents systèmes allant de la narration interactive au jeu vidéo [Weber et al., 2011]. Ce langage très expressif permet en effet de représenter le contenu scénaristique à plusieurs niveaux de granularité. Cependant, sa syntaxe n'est utilisable que par des programmeurs expérimentés.

```
// Trip's behavior
joint sequential behavior HaveAFight() {
    with (synchronize) joint subgoal YellForAwhile();
    subgoal StompOverToBarAndMakeDrink();
    with (synchronize) joint subgoal FightSomeMore();
}

// Grace's behavior
joint sequential behavior HaveAFight() {
    with (synchronize) joint subgoal YellForAwhile();
    subgoal StompOverToCabinetAndFiddleWithCollection();
    with (synchronize) joint subgoal FightSomeMore();
}
```

FIGURE 3.21 – Comportements synchronisés représentés en ABL [Mateas, 2002]

3.1.4 Bilan et positionnement

Afin d'assurer l'**adaptabilité** de l'environnement virtuel, le formalisme choisi pour la représentation du contenu scénaristique doit permettre la **maintenabilité**, la **réutilisabilité** et le **passage à l'échelle** de ce contenu, ainsi que la **variabilité** des scénarios construits à partir de ce contenu. De plus, afin de répondre à notre objectif de **cohérence**, le formalisme doit permettre d'exprimer du contenu qui ait une **validité écologique**.
A partir de ces objectifs, nous avons extrait quatre propriétés qu'il est nécessaire que le formalisme choisi possède :

- la **modularité**, afin de pouvoir ajouter, supprimer, modifier ou réutiliser des parties de ce contenu sans avoir à le modifier en intégralité,
- l'**intelligibilité**, afin que le contenu scénaristique puisse être renseigné directement par des experts du domaine concerné, et non uniquement par des informaticiens,
- l'**expressivité**, afin de pouvoir modéliser des comportements complexes, rares, non-idéaux, à la fois au niveau des systèmes techniques et des comportements humains,
- l'**interprétabilité**, afin de pouvoir être directement utilisé par des modules informatiques sans nécessiter de traduction manuelle préalable.

Parmi les formalismes présentés, aucun n'est satisfaisant au niveau de ces quatre propriétés. La séparation des modèles de domaine et d'activité, plutôt que l'utilisation d'un modèle d'événements, permet toutefois d'accroître la **modularité** du formalisme, en permettant notamment de réutiliser un même modèle de domaine pour différentes applications, avec des modèles d'activité différents.
L'ajout de sémantique aux formalismes de **représentation du domaine** permet davantage d'**expressivité**, de **modularité** et d'**intelligibilité** par rapport à des représentations qui soient purement sous

forme d'automates finis. Cependant dans la plupart des cas ces mêmes automates sont utilisés pour représenter les comportements des systèmes, qui ne sont donc pas inspectables. Des formalismes basés sur des règles (COLOMBO) ou des associations de préconditions et effets (Cause-and-Effect) apportent des solutions intéressantes, mais ceux ayant été étudiés manquent de **modularité**, ainsi que d'**expressivité** pour la représentation de comportements complexes des systèmes qui ne soient pas liés aux actions des agents.
L'utilisation d'automates finis ou de réseaux de Petri pour la **représentation de l'activité** n'est pas non plus satisfaisante au niveau de la **modularité**, puisque ceux ci nécessitent d'expliciter toutes les transitions possibles. La représentation par un ensemble d'opérateurs de planification est également limitée sur ce point, car elle ne permet pas d'avoir une vision d'ensemble de l'activité. L'utilisation d'un modèle hiérarchique permet une plus grande souplesse, notamment pour l'ajout d'alternatives pour la réalisation d'un même but. En particulier, les langages issus de l'ergonomie sont à la fois **expressifs** et **intelligibles**, cependant ils pèchent au niveau de l'**interprétabilité**.
Nous proposons deux langages pour la représentation du contenu scénaristique :

- **DOMAIN-DL**, un langage de modélisation du domaine, inspiré de COLOMBO et VEHA, et décrit dans la section 5.1 ;
- **ACTIVITY-DL**, un langage de modélisation de l'activité observée, inspiré de HAWAI-DL, CTT et HTN, et décrit dans la section 5.2.

Ces langages sont utilisés par les experts du domaine et ergonomes pour renseigner les connaissances qui serviront de base à l'environnement virtuel. Ces modèles sont ensuite traduits automatiquement sous la forme d'**opérateurs de planification PDDL** afin de pouvoir être manipulés par le moteur de planification inclus dans DIRECTOR. Ces opérateurs, et la manière dont ils sont générés, sont décrits dans la partie 6.4.

3.2 Représentation d'objectifs scénaristiques

En parallèle des connaissances sur le **contenu scénaristique**, qui définissent l'ensemble des **scénarios possibles**, la majorité des systèmes de scénarisation reposent également sur des connaissances concernant les **objectifs scénaristiques**, qui permettent de décrire le sous-ensemble de **scénarios souhaitables**. On parle également de *narrative control knowledge* [Porteous et al., 2010] ou d'*author goals* [Riedl, 2004].
Ces connaissances sont utilisées pour contrôler la génération ou la sélection dynamique de scénarios dans les approches où l'ensemble des scénarios n'est pas défini à l'avance par un auteur. Certains de ces objectifs sont définis de manière statique par rapport au système, permettant de spécifier un sous-espace de scénarios d'intérêt, par exemple dans le cas d'une scénarisation extrinsèque. D'autres sont définis de manière dynamique, soit au début d'une session, soit au sein-même d'une session d'utilisation de l'environnement virtuel. La définition dynamique de ces objectifs permet d'adapter les scénarios résultants à l'activité ou au profil de l'utilisateur. Les objectifs peuvent avoir une valeur prescriptive, sous la forme de contraintes dont le scénario ne peut s'écarter, ou indicative, reflétant le scénario idéal vers lequel tend la simulation.
On peut distinguer deux catégories d'objectifs scénaristiques : les objectifs de **bas niveau**, qui visent directement le **contenu** du scénario, et les objectif de **haut niveau** qui introduisent des **annotations** des situations et des **fonctions d'évaluation** de la valeur d'un scénario selon différents critères.

3.2.1 Objectifs scénaristiques de bas niveau

La majorité des systèmes de scénarisation utilisent des objectifs scénaristiques de bas niveau, portant directement sur le scénario et les événements qui le composent. Il peut s'agir de contraintes locales sur un moment particulier du scénario (conditions initiales ou situation finale) comme d'ob-

jectifs globaux portant sur l'ensemble du scénario (points clés à rencontrer, contraintes d'ordonnancement sur les événements, etc.).

Contraintes locales

De nombreux systèmes optant pour une approche centrée sur les personnages définissent les objectifs scénaristiques par un ensemble d'assertions sur l'**état initial** de la simulation, par exemple [Shawver, 1997]. [Hullett and Mateas, 2009] proposent une description plus ouverte, sous la forme d'un ensemble de contraintes qui permettent de calculer cet état initial.
Les systèmes basés sur la planification d'un scénario associent à la définition de l'état initial celle de l'état final, sous la forme d'une **situation but** contenant un ensemble d'assertions sur l'état de la simulation devant être vraies à la fin du scénario, par exemple [Young et al., 2004].

character(aladdin) male(aladdin) knight(aladdin) at(aladdin, castle) alive(aladdin) single(aladdin) loyal-to(aladdin, jafar) character(jafar) male(jafar) king(jafar) at(jafar, castle) alive(jafar) single(jafar)	character(jasmine) female(jasmine) at(jasmine, castle) alive(jasmine) single(jasmine) beautiful(jasmine) character(dragon) monster(dragon) dragon(dragon) at(dragon, mountain) alive(dragon) scary(dragon)	character(genie) monster(genie) genie(genie) in(genie, lamp) confined(genie) alive(genie) scary(genie) place(castle) place(mountain) thing(lamp) magic-lamp(lamp) has(dragon, lamp)

married-to(jafar, jasmine)	¬alive(genie)

FIGURE 3.22 – Situation initiale et situation finale définies sous forme de prédicats [Riedl and Young, 2010]

Ces contraintes permettent de contrôler précisément le début et la fin du scénario, mais n'offrent **aucun contrôle sur les situations intermédiaires**. Cette limitation est particulièrement visible dans les systèmes basés sur de la planification, où le moteur de planification va chercher à générer le plan le plus court entre la situation initiale et la situation finale, ce qui n'offre aucune garantie sur la qualité du plan. Pour offrir un contrôle plus important sur la qualité narrative ou pédagogique du scénario, certains systèmes prennent alors en compte des contraintes globales, devant être respectées sur l'ensemble du scénario.

Contraintes globales

Il est ainsi possible de définir des ensembles de **situations prescrites** (devant arriver au cours du scénario) ou **proscrites** (devant au contraire être évitées), ici encore sous la forme d'ensemble d'assertions sur l'état du monde. On parle des situations prescrites comme des "points clés", ou *plot points* [Magerko, 2005]. Ces points clés peuvent être associés à des contraintes temporelles, spécifiant des relations d'ordonnancement entre des couples de points clés [Riedl et al., 2008b] [Porteous and Cavazza, 2009], voire des contraintes absolues sur le nombre de pas de simulation avant lequel certains points clés doivent apparaître [Si, 2010]. [Porteous and Cavazza, 2009] définissent ces contraintes en utilisant les *state trajectory constraints* du langage PDDL 3.0 (voir figure 3.23).

operator	meaning
(sometime-before a b)	b must be made true for the first time before a
(sometime a)	predicate a must be true at some stage of the narrative
(at-end a)	predicate a must be true at the end of the narrative

(sometime-before (seduced bond jill) (won-cards bond goldfinger card-game))
(sometime-before (seduced bond jill) (got-mission bond))
(sometime (won-golf bond goldfinger golf-game))

FIGURE 3.23 – Opérateurs et exemples de contraintes utilisées dans [Porteous and Cavazza, 2009]

Cette représentation des objectifs scénaristiques comme des points clés partiellement ordonnés se rapproche, selon le niveau de détail utilisé, d'une représentation explicite d'un scénario. De même, certains systèmes utilisent une représentation d'un ou plusieurs scénarios linéaires servant de "fil-rouge", c'est à dire de séquences d'événements exemplaires que le système va chercher à approcher. C'est le cas notamment dans IN-TALE [Riedl et al., 2008b] et Thespian [Si, 2010]. Les formalismes utilisés pour la représentation explicite d'un scénario ou d'un ensemble de scénarios sont décrits plus en détails dans la section 3.3.
L'utilisation de points clés et de contraintes temporelles permet une grande souplesse dans le niveau de contrôle du scénario, allant jusqu'à un **contrôle très fin** des situations dans le cas des systèmes basés sur des représentations explicites de scénarios prédéfinis. Ici, la responsabilité de la qualité du scénario revient en grande partie à l'auteur ou au système qui définit ou génère ces points clés : c'est à eux de s'assurer que les situations prescrites présentent un intérêt et qu'il n'y ait pas d'incohérences entre les contraintes. Les points clés étant définis en termes de situations de l'environnement virtuel, **l'expressivité** de ces formalismes est limitée : ils ne permettent par exemple pas de poser des contraintes explicites sur des propriétés de plus haut niveau du scénario, comme son rythme, la charge cognitive de l'utilisateur, etc.

3.2.2 Objectifs scénaristiques de haut niveau

Dans certain cas, la pertinence d'un scénario n'est pas liée intrinsèquement aux situations qui le composent, mais à des propriétés de plus haut niveau, comme par exemple le rythme auquel ces situations s'enchaînent. En particulier lorsqu'il s'agit d'environnements virtuels destinés à la formation à des compétences non-techniques, comme la gestion du stress, il peut être intéressant pour un système de scénarisation de manipuler de tels objectifs scénaristiques de haut niveau.
Les connaissances liées à ces objectifs peuvent être directement encodées sous la forme de **fonctions d'évaluation** de différentes propriétés (ex : intérêt de l'utilisateur pour le scénario) que les moteurs chercheront à maximiser. Ces fonctions peuvent également être associées à des **valeurs souhaitées**, sous la forme de **trajectoires** permettant de faire varier ces propriétés au cours du scénario, ou de **structures de récit**. Dans certain cas, ces valeurs pourront être définies de manière **dynamique**, afin de pouvoir adapter ces propriétés en fonction de l'activité de l'utilisateur.

Fonctions d'évaluation

L'architecture Moe, proposée par [Weyhrauch, 1997] dans le cadre du projet Oz, a été l'un des premiers systèmes a utiliser une telle approche. Moe utilise une fonction d'évaluation basée sur un ensemble de propriétés du scénario, et cherche à maximiser cette fonction. Sept propriétés sont considérées :

- *thought flow* : l'existence de liens logiques entre les événements vécus par l'utilisateur,
- *activity flow* : la proximité spatiale entre les actions réalisées par l'utilisateur,

- *options* : la liberté d'action perçue par l'utilisateur,
- *motivation* : le lien entre les actions de l'utilisateur et ses buts,
- *momentum* : la proximité de certains événements définis par l'auteur comme devant être concentrés temporellement,
- *intensity* : l'intérêt de l'utilisateur pour l'histoire, assimilé à la progression de ses connaissances sur le monde,
- *manipulation* : le sentiment de manipulation éprouvé par l'utilisateur face au système.

Chacune de ces propriétés est associée à une fonction d'évaluation *ad hoc*. Pour le calcul du *thought flow*, par exemple, les actions possibles sont annotées en fonction des sujets concernés, et la valeur est incrémentée quand deux actions séquentielles possèdent des annotations communes.
Le système IDTension [Szilas, 2003] utilise une approche similaire basée sur la combinaison non-linéaire de *critères narratifs* reprenant en partie les propriétés de Moe. Ces critères sont au nombre de six : pertinence d'une action par rapport aux actions précédentes, cohérence des personnages au niveau de leurs motivations, cohérence des personnages au niveau de leur personnalité, charge cognitive, caractérisation (expression de la personnalité des personnages), conflit.
Les objectifs sont ici **directement encodés dans les fonctions d'évaluation**, ce qui ne permet pas de redéfinir et d'adapter ces objectifs selon les sessions d'utilisation de l'environnement virtuel. De plus, dans certains cas, il n'est pas souhaitable de maximiser une propriété donnée sur l'ensemble du scénario, mais plutôt de faire varier sa valeur : par exemple une augmentation graduelle de l'intensité, ou une diminution des conflits sur la fin du scénario pour ne pas laisser l'utilisateur trop en suspens.

Trajectoires et structures de récit

Pour cela, certains systèmes associent à la définition de fonctions d'évaluation une représentation des **trajectoires souhaitées** pour les valeurs associées, ou définissent des **structures de récit** à partir de théories narratives.
Delmas propose ainsi trois structures de récit interactif adaptées de structures narratives types [Delmas, 2009]. Ces structures décrivent les propriétés du scénario à générer pendant son exécution. La première s'inspire de la structure aristotélicienne, constituée de trois parties : prologue, épisode et dénouement. Elle est ici traduite sous la forme d'une grammaire, qui définit la décomposition du scénario en "situations" et "rebondissements" (deux catégories d'événements, annotés comme tels). A cette structure est associée une seconde structure basée sur l'évolution de la tension dramatique dans le récit. Elle est représentée par une **courbe canonique de tension dramatique** (voir figure 3.24). Afin de pouvoir évaluer la tension dramatique du scénario, chaque rebondissement possible est annoté avec une valeur de tension dramatique donnée. Enfin, une troisième structure, basée sur le Périple du Héros, utilise une catégorisation des événements plus détaillée.

[Porteous et al., 2011] utilisent une approche similaire basée sur la tension narrative pour la génération d'une séquence de points clés. Comme dans l'architecture de pilotage proposée par Delmas, chaque point clé est annoté avec une valeur de tension dramatique. La trajectoire voulue est représentée visuellement sous la forme d'un "arc narratif" (voir figure 3.25). Cependant, ici, l'auteur peut modifier la courbe en amont pour s'éloigner de la courbe aristotélicienne classique.

La spécification de trajectoires ou de structures narratives devant être respectées par le scénario pour certaines propriétés de haut niveau permet davantage de **contrôle** par rapport à l'utilisation de fonctions d'évaluation identiques sur l'ensemble du scénario. Les deux systèmes présentés utilisent des annotations sur les événements pour calculer ces propriétés, mais pourraient également utiliser des fonctions d'évaluation plus complexes.
Si les objectifs sont représentés ici sous la forme de trajectoires prédéterminées, les valeurs souhaitées pour les différentes propriétés peuvent également être **spécifiées de manière dynamique** durant le déroulement de la simulation. En effet, si certaines trajectoires — comme l'intensité suivant un arc

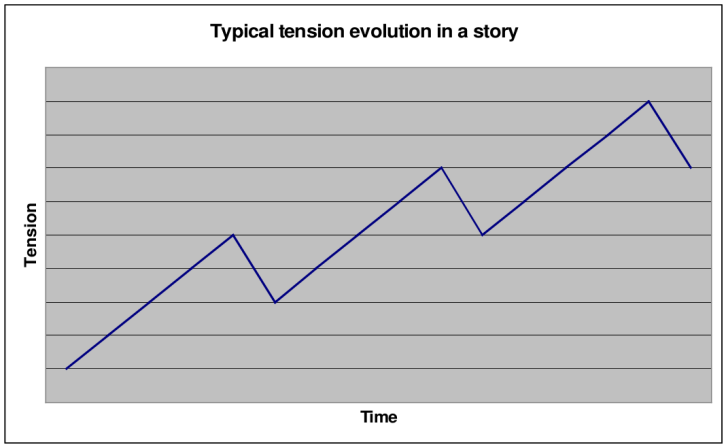


FIGURE 3.24 – Courbe canonique de tension dramatique [Delmas, 2009]

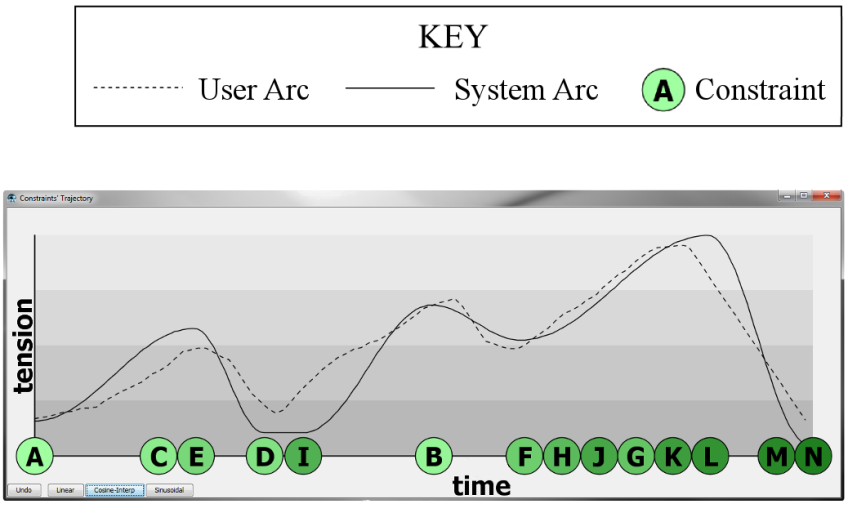


FIGURE 3.25 – Arcs de tension dramatique et points clés associés [Porteous et al., 2011]

aristotélicien — seront visées quelles que soient les conditions, d'autres peuvent être dépendantes du contexte. Par exemple, un formateur pourrait chercher à augmenter ou diminuer le niveau de complexité d'un scénario selon que l'apprenant qui utilise l'environnement virtuel rencontre ou non des difficultés. [Magerko et al., 2005] définit ainsi des propriétés de haut niveau sous la forme de **compétences**. Chaque point clé possible est associé avec les compétences qu'il met en jeu. Les objectifs scénaristiques sont déterminés à chaque pas en fonction des performances de l'utilisateur sur les différentes compétences, et envoyés au moteur de scénarisation sous la forme d'un vecteur associant un poids à chaque compétence.

3.2.3 Bilan et positionnement

La définition d'objectifs scénaristiques de **bas niveau** permet un **contrôle très fin** sur les situations rencontrées dans le scénario, tandis que la définition d'objectifs de **haut niveau** permet davantage d'**adaptabilité**, notamment lorsqu'il s'agit de modifier ces objectifs de manière dynamique. Plutôt que contradictoires, ces deux approches sont au contraire **complémentaires**.
Les motivations exprimées précédemment quant aux environnements virtuels pour la formation et l'aide à la décision concernent en effet à la fois le besoin de mener la simulation vers des situations d'apprentissage ponctuelles particulières, et celui de mettre en place des situations plus larges : situation de stress, co-activité difficile, etc. La scénarisation extrinsèque d'une simulation existante né-

cessite de plus de cadrer cette dernière, et donc de définir, en parallèle des modèles de domaine et d'activité, un modèle représentant l'espace des scénarios d'intérêt pour les besoins d'une application donnée.
Nous prenons ainsi en compte dans notre système **trois types de connaissances** pour les objectifs scénaristiques :

- **les situations prescrites et proscrites**,
- **les propriétés du scénario désirées**,
- **l'espace de scénarios d'intérêt**.

Les connaissances **indépendantes du domaine**, comme certaines propriétés du scénario, peuvent être directement encodées sous la forme de **fonctions d'évaluation**. Au contraire, les connaissances **dépendantes du domaine**, comme l'espace de scénarios d'intérêt, doivent faire l'objet d'une représentation explicite dans un **modèle**. L'espace de scénarios d'intérêt sera ainsi représenté grâce à un formalisme de représentation de scénarios, tels ceux décrits plus en détail dans la section 3.3.
A l'inverse de l'espace de scénarios d'intérêt, défini de manière statique, les situations prescrites et proscrites et les propriétés souhaitées du scénario sont générées de manière dynamique par un module pédagogique externe, afin de permettre l'adaptation à l'activité de l'utilisateur. Les formalismes utilisés, ainsi que les fonctions d'évaluation associées aux propriétés, sont décrites en détail dans la partie 7.

3.3 Représentation des scénarios

On distingue enfin un troisième type de formalismes : ceux qui permettent de représenter de manière explicite un scénario particulier, ou un espace de scénarios. Nous considérons ici un **scénario** comme étant un déroulement particulier d'événements et d'actions instanciés dans un environnement virtuel. Nous parlerons de **trame scénaristique** lorsqu'il s'agira de désigner un enchaînement d'événements clés non instanciés, sans forcément représenter les événements intermédiaires. Une trame scénaristique décrit alors un espace de scénarios. Un **espace de scénarios** peut également être décrit à l'aide de formalismes permettant d'exprimer des alternatives entre plusieurs événements ou séquences d'événements.
Selon les cas, ces formalismes sont utilisés pour représenter des scénarios ou des espaces de scénarios qui sont **prédéfinis**, ou bien qui sont **générés dynamiquement**. Les scénarios ainsi représentés peuvent soit être destinés à être **exécutés** dans l'environnement virtuel, soit constituer une **trace** des événements passés, soit correspondre à des **objectifs scénaristiques** pour la génération ou la sélection dynamique d'un scénario : espace de scénarios d'intérêt ou scénario "fil rouge" — ce dernier pouvant avoir une valeur indicative (scénario à approcher dans la mesure du possible) ou prescriptive (trame scénaristique à respecter).
Il existe de nombreuses approches pour la représentation des scénarios, selon le domaine de recherche considéré : modèles de compréhension d'histoires, de scénarios pédagogiques ou encore d'analyses de risques. Au niveau des formalismes purement informatiques, on trouve deux approches principales : les représentations basées sur des automates d'états finis ou des graphes orientés acycliques, et les représentations sous forme de plans.

3.3.1 Modèles de compréhension d'histoires

Les travaux menés sur la **compréhension d'histoires** ont donné naissance à un certain nombre de **modèles de structure narrative**, permettant de représenter des scénarios. Il s'agit non seulement de représenter les actions et changements d'états du monde et leur ordonnancement temporel, mais également d'exprimer les relations de causalité entre les buts des personnages, leurs actions et les effets de ces actions, ainsi que les relations hiérarchiques entre les buts, tels qu'ils peuvent être compris

par un lecteur ou spectateur.
Ces formalismes représentent les scénarios sous la forme de **graphes d'éléments narratifs** reliés par des **liens de causalité**, et définissent chacun une typologie d'éléments et de relations causales. Nous présentons ici trois d'entre eux : QUEST [Graesser et al., 1991], GTN [Trabasso et al., 1989] et Fabula [Swartjes, 2010].

QUEST

Le modèle QUEST [Graesser et al., 1991] vise à expliquer le processus de réponse à des questions au sujet d'un texte écrit. Il repose sur une représentation du texte sous forme de graphe contenant deux types de noeuds : les Buts et les Evénements. Ces nœuds sont liés par cinq types de relations : Conséquence (lien de causalité entre deux événements), Raison (lien entre un but et un événement), Initiation (lien entre un événement et un but), Résultat (lien entre un but et un événement) et Implication (lien de subsomption entre deux événements). Un exemple de graphe est présenté dans la figure 3.26.

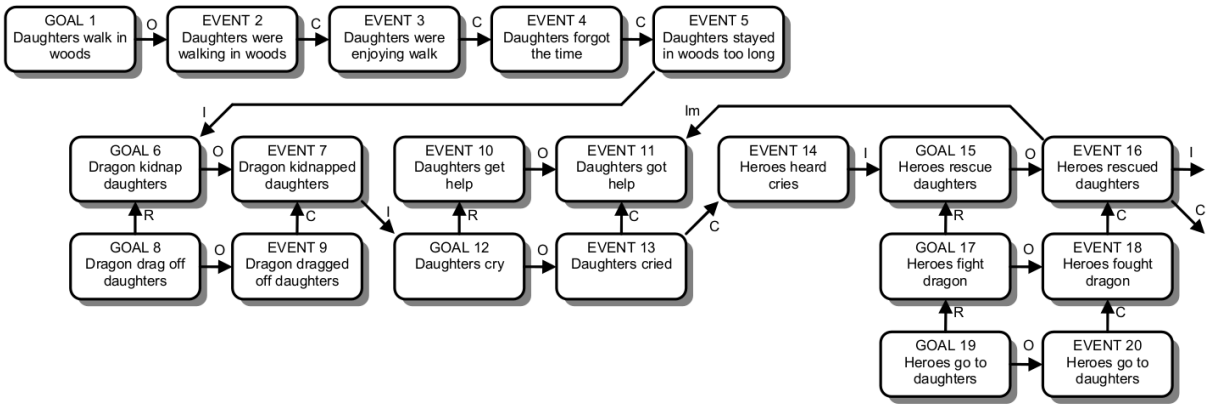


FIGURE 3.26 – Portion de graphe QUEST présentée dans [Riedl and Young, 2005]

Ce modèle permet de représenter les motivations des personnages de l'histoire, mais il est limité par l'absence de distinction formelle entre les actions des personnages, les changements d'états liés à des actions et les événements exogènes.

GTN

Le modèle GTN (General Transition Network), proposé par [Trabasso et al., 1989], est plus complet en cela qu'il propose de distinguer six éléments : Eléments de cadre (*Settings*), Evénements, Réponses internes, Buts, Tentatives et Résultats. Les relations entre ces éléments peuvent avoir trait à de la causalité physique, de la causalité psychologique, de la motivation ou de l'*enablement*, forme faible de causalité indiquant qu'un événement rend possible un autre événement.
Ce modèle permet une représentation des scénarios plus riche au niveau des personnages, cependant il s'attache à représenter l'histoire du point de vue de l'un d'eux. Pour obtenir une vision globale du scénario, il est alors nécessaire d'associer plusieurs graphes.

Fabula

Le modèle Fabula [Swartjes, 2010] étend le modèle GTN en regroupant les points de vue des différents personnages de l'histoire au sein d'un même graphe. Pour cela, il prend en compte un élément

narratif supplémentaire ayant trait à la *Perception*.
De plus, il inclut une hiérarchie d'éléments narratifs permettant de qualifier ces derniers de manière plus précise. Ainsi *Perception* englobe *See* (voir) et *Hear* (entendre), tandis que *Internal Element* inclut *Emotion* et *Belief*.

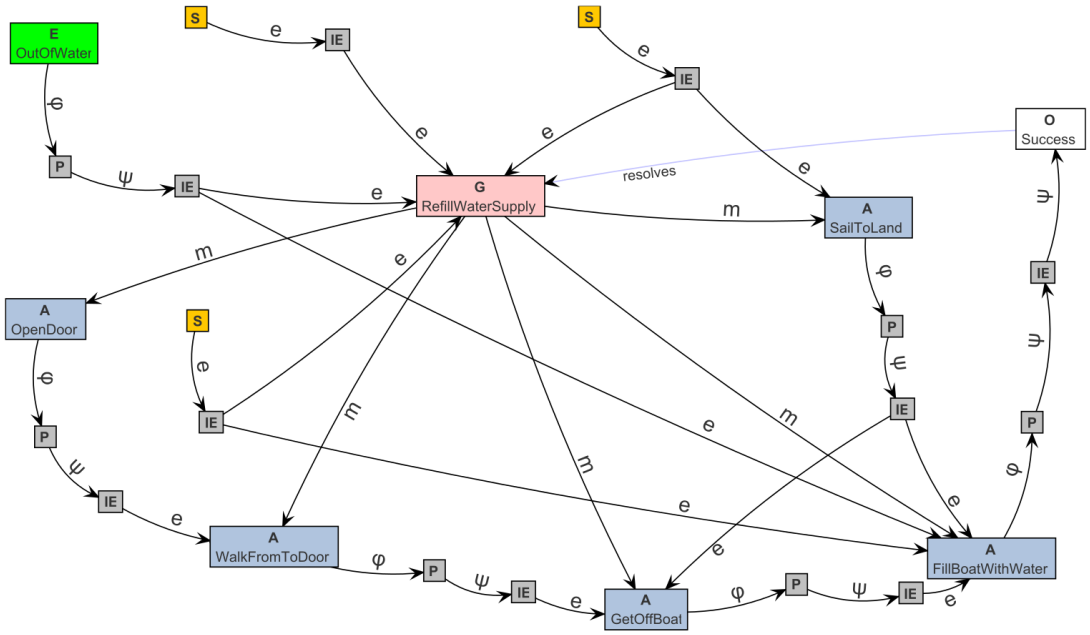


FIGURE 3.27 – Exemple de Fabula [Swartjes, 2010]

Tandis que les deux modèles précédents étaient destinés à de l'analyse et de la compréhension d'histoires textuelles, [Swartjes, 2010] s'intéresse à la génération automatique de cette représentation à partir d'une simulation. Il propose donc une opérationnalisation de ce modèle sous la forme d'une ontologie RDF, dont les éléments sont instanciés au fur et à mesure des actions et des processus internes des personnages. Un exemple de graphe ainsi généré est présenté dans la figure 3.27.

Bilan

Les modèles issus des recherches sur la compréhension d'histoires sont des **formalismes très expressifs**, qui permettent de représenter non pas simplement l'ordonnancement temporel entre les événements mais les relations de causalité entre eux. Les typologies d'éléments et de relations qu'ils utilisent permettent d'exprimer différents types de causalité, et de modéliser les processus décisionnels des personnages de manière fine.
Ces modèles sont destinés à de l'analyse et de la compréhension d'histoires. Seul Fabula a vocation à permettre la génération de récit à partir d'une histoire générée automatiquement. Cependant, dans tous les cas, ils visent à représenter un **scénario linéaire unique**, et non un ensemble de scénarios.
En cela, ces formalismes sont pertinents pour représenter la **trace d'une simulation** en vue de son explication, mais pas pour représenter un ensemble de scénarios d'intérêt.

3.3.2 Modèles de scénarios pédagogiques

Le domaine des EIAH (Environnements Informatiques pour l'Apprentissage Humain) s'intéresse, lui, à la représentation de **scénarios pédagogiques**. Un scénario pédagogique décrit non seulement l'**enchaînement des activités pédagogiques**, mais également les objectifs pédagogiques qui leur sont

associés — en termes de connaissances ou compétences —, les prérequis, les rôles des différents agents intervenant dans le scénario — qu'ils soient formateurs ou apprenants — et les outils et ressources nécessaires à la réalisation des activités pédagogiques [Marion, 2010].
Nous présentons ici trois langages de représentation de scénario pédagogiques : IMS-LD [Koper et al., 2003], FORMID [Guéraud and Lejeune, 2011] et POSEIDON [Marion, 2010].

IMS-LD

La norme IMS Learning Design, ou IMS-LD [Koper et al., 2003], est l'un des langages majeurs de la communauté des EIAH. IMS-LD décrit les scénarios comme un enchaînement d'activités pédagogiques ; chaque activité est décrite par un texte ou un ensemble de documents expliquant le but de l'activité, la tâche à réaliser et les consignes à respecter. Les activités peuvent être organisées de manière séquentielle ou parallèle, et être soumises à des conditions, en fonction des sorties des activités précédentes.
De par leur description textuelle, les activités sont considérées comme des "boîtes noires" du point de vue du système de scénarisation, qui ne connaît que les entrées (ressources) et les sorties (productions).

FORMID

D'autres travaux s'intéressent donc à décrire également les scénarios au niveau des activités elles-mêmes, afin de pouvoir contrôler à un niveau plus fin les situations d'apprentissage devant être rencontrées par les apprenants.
L'approche FORMID [Guéraud and Lejeune, 2011] propose ainsi de représenter les scénarios comme étant constitués de plusieurs étapes ; chaque étape comprend une consigne et un objectif à atteindre, qui correspond à une condition sur les états de la simulation. Les étapes comprennent également un ensemble de situations particulières, elles aussi modélisées par des conditions sur les états de la simulation, qui peuvent correspondre soit à des erreurs classiques, soit à des étapes attendues, et qui sont associées à des rétroactions à délivrer en réponse à ces situations. Le scénario d'une activité est donc vu ici comme un ensemble de situations, sans liens entre elles.

POSEIDON

POSEIDON (PedagOgical ScEnario for vIrtual and informeD envirONment) est un modèle générique de scénarios pédagogiques pour les environnements virtuels pour la formation [Marion, 2010]. Il s'appuie sur MASCARET, le méta-modèle basé sur UML dont font partie les formalismes VEHA et HAVE présentés respectivement dans les sections 3.1.1 et 3.1.2.

POSEIDON permet de décrire une activité pédagogique soit comme une boîte noire — comme c'est le cas avec IMS-LD — soit sous la forme d'un sous-diagramme d'activité. Ce sous-diagramme correspond en fait à un modèle BEHAVE, comme décrit dans la section 3.1.2. Les actions référencées dans ce modèle sont associées à un rôle, qui peut soit correspondre à un apprenant, soit à un encadrant. Ces rôles peuvent être endossés par un humain ou bien par un personnage virtuel. La description de ce scénario est de plus enrichie avec un ensemble de rétroactions associées à des situations particulières, de manière similaire à FORMID. Un exemple de scénario pédagogique modélisé avec POSEIDON est présenté dans la figure 3.28.

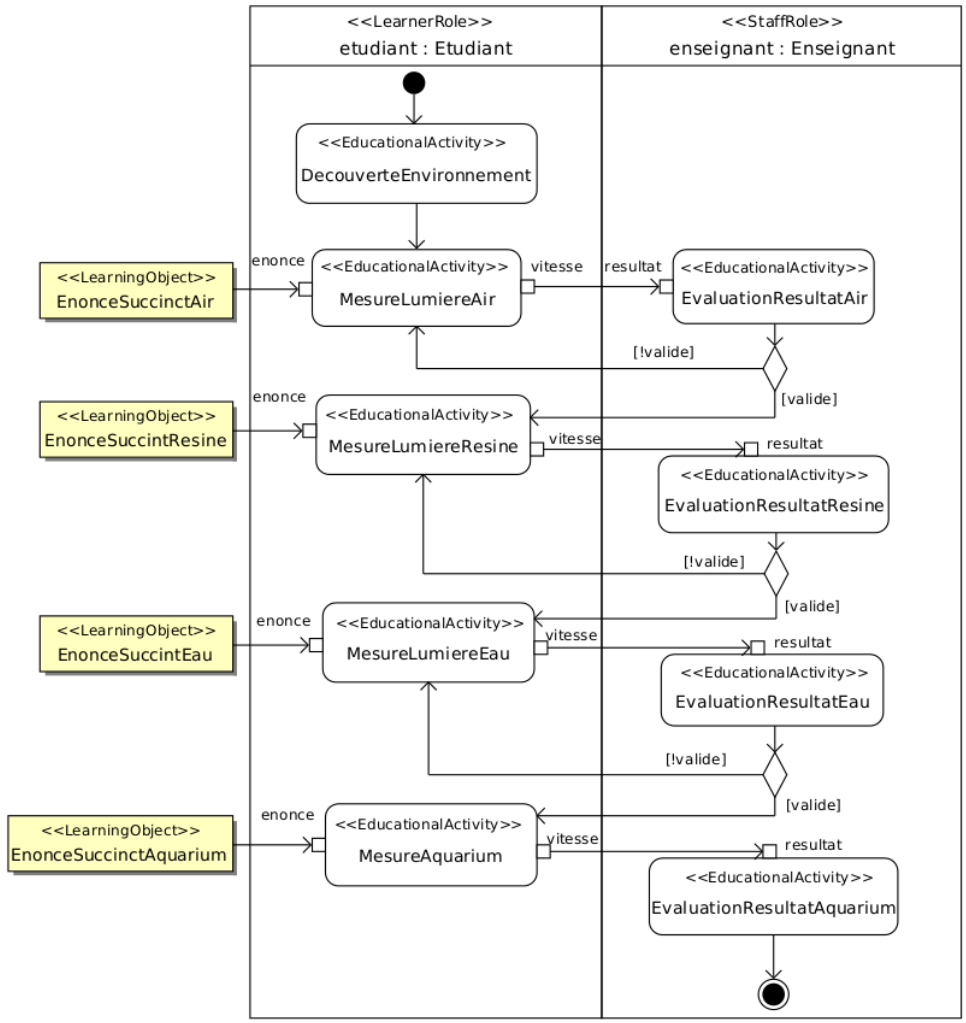


FIGURE 3.28 – Exemple de scénario pédagogique POSEIDON [Marion, 2010]

Bilan

Les modèles de scénarios pédagogiques permettent de décrire finement le **contexte d'utilisation de l'environnement virtuel**, en plaçant cette activité au sein d'un scénario d'apprentissage global et en l'associant à différents éléments liés à la pédagogie — objectifs d'apprentissages, compétences et connaissances requises, rétroactions multimédia associées aux erreurs classiques, etc.
Cependant, pour la plupart des langages, le niveau de description s'arrête aux **activités pédagogiques** et n'est donc pas adapté pour représenter le déroulement des événements au sein de l'environnement virtuel. Pour ces langages, comme IMS-LD par exemple, l'environnement virtuel est vu comme une **"boîte noire"**.
D'autres langages, comme FORMID et POSEIDON, se destinent plus spécifiquement à la représentation de scénarios pédagogiques faisant intervenir des environnements virtuels, et offrent un plus grand niveau de granularité en descendant au niveau des événements. Ils permettent d'exprimer les spécificités des tâches en fonction d'états du monde — modélisant ainsi les situations de danger ou d'erreur — ou encore de décrire les actions unitaires attendues de la part des apprenants et formateurs. Ces langages suffisent pour représenter des **ensembles de situations non ordonnées** ou des **séquences d'actions linéaires**, mais restent toutefois limités pour la **description des scénarios à haut niveau** et des **liens de causalité** entre événements.

3.3.3 Modèles d'analyses de risques

Les domaines de l'**analyse de risques** et de l'**analyse accidentelle** s'intéressent eux aussi à la représentation de scénarios. L'analyse accidentelle se focalise sur l'analyse *a posteriori* d'accidents, de sorte à reconstituer et expliquer le scénario accidentel y ayant mené. L'analyse de risque vise à caractériser la **gravité potentielle** et la **probabilité d'occurrence** d'hypothétiques futurs accidents, en prédisant les scénarios accidentels possibles. Pour cela, un certain nombre de formalismes graphiques ont été élaborés pour représenter les chaînes de causalité entre événements. Nous détaillerons ici les arbres de défaillance [Vesely et al., 1981], les arbres d'événements [Rausand and Høyland, 2004] et les nœuds papillons [Bernuchon et al., 2006], ainsi que des travaux portant sur la transposition de ces noeuds papillons en réseaux bayésiens [Amokrane, 2010].

Arbres de défaillances

La représentation en arbres de défaillances est une représentation arborescente des causes d'un événement, utilisée pour décrire les enchaînements et combinaisons d'événements pouvant mener à un événement redouté [Vesely et al., 1981]. On parle également d'arbre des causes lorsqu'il s'agit d'un contexte d'analyse accidentelle plutôt que d'analyse de risques [Bernuchon et al., 2006].
L'arbre est constitué d'un ensemble d'événements, liés entre eux à l'aide de portes ET et OU. Les événements de base, qui constituent les feuilles de l'arbre, sont indépendants et sont associés à des estimations de probabilité d'occurrence. Les liens passant par des portes OU indiquent des relations de subsomption entre les événements, tandis que les liens passant par des portes ET indiquent des relations de causalité (voir figure 3.29). Les probabilités d'occurrence de l'événement final et des événements intermédiaires peuvent être calculées à partir de cette décomposition et des probabilités d'occurrence des événements de base.

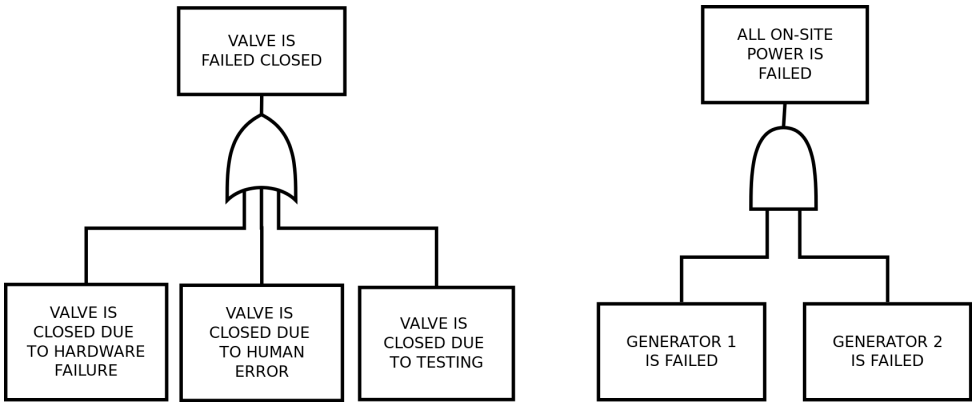


FIGURE 3.29 – Exemple de porte OU (à gauche) et de porte ET (à droite) [Vesely et al., 1981]

Cette représentation portant sur un événement accidentel particulier, son application à tout un système peut s'avérer fastidieuse, voire impossible puisque la représentation arborescente ne permet pas de représenter des conséquences multiples pour un même événement.

Arbres d'événements

A l'inverse, la représentation par un arbre d'événements part d'un événement initiateur, correspondant à la défaillance d'une partie d'un système, et décrit les conséquences qui en découlent [Rausand and Høyland, 2004].

L'arbre est construit de gauche à droite, chaque intersection correspondant à une fonction de sécurité : la branche du haut correspond alors au succès de cette fonction, et la branche du bas à sa défaillance. Un exemple d'arbre est présenté en figure 3.30.

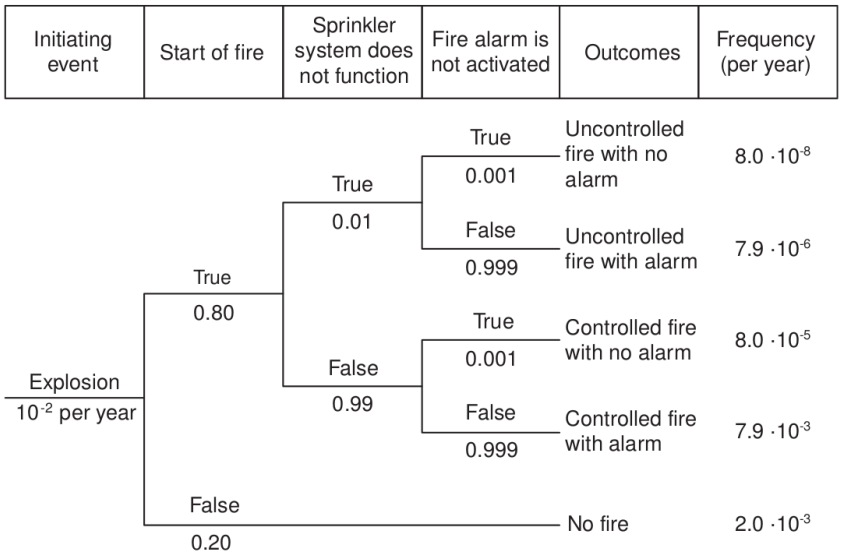


FIGURE 3.30 – Exemple d'arbre d'événements [Rausand and Høyland, 2004]

Les feuilles de l'arbre correspondent aux événements qui découlent de l'événement initiateur et à leur probabilité en fonction des succès et défaillances successifs des fonctions de sécurité. Cette représentation est toutefois très limitée pour la représentation de scénarios plus larges, puisqu'elle ne permet que de s'intéresser à un seul événement initiateur, et nécessite de considérer les fonctions de sécurité de manière séquentielle.

Nœuds papillons

La représentation de scénarios accidentels par des nœuds papillons peut être assimilée à l'association d'un arbre d'événements et d'un arbre de défaillances, joints au niveau d'un événement redouté central [Deust et al., 2008].

Cette représentation est centrée autour de la notion de barrières de sécurité. Une barrière est un système actif ou passif, technique ou humain, assurant une fonction de sécurité [Miche et al., 2009]. Il peut s'agir de barrières de vérification, ou prévention — qui seront situées en amont de l'événement redouté central dans le graphe — ou de barrières de rattrapage, ou protection — qui seront situées en aval. Les barrières techniques se rapportent à des systèmes techniques, tandis que les barrières humaines sont constituées d'une ou plusieurs opérations devant être réalisées par un opérateur humain. La typologie des différentes barrières est présentée en figure 3.32.

Afin de permettre le calcul des probabilités d'occurrence des différents événements redoutés, les événements initiaux sont annotés avec leur probabilité d'occurrence, assimilée à l'estimation de leur fréquence d'occurrence future [Deust et al., 2008]. Les barrières sont annotées avec leur niveau de confiance ; le niveau de confiance est la classe de probabilité pour qu'une mesure de maîtrise des risques n'assure plus la fonction de sécurité pour laquelle elle a été choisie [Deust et al., 2008].
La représentation d'un scénario par un nœud papillon étant centrée sur un événement particulier, elle ne permet pas de représenter dans un même modèle toutes les chaînes de causalité possibles dans un système donné. De plus, le formalisme ne distingue pas les liens de causalité des liens de

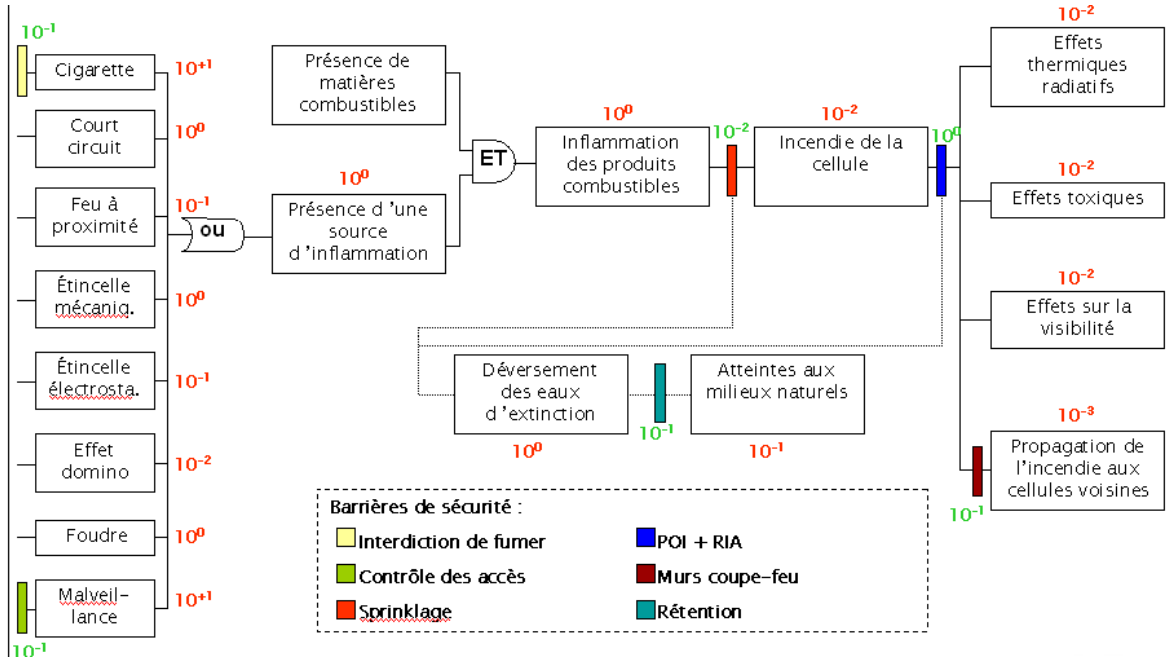


FIGURE 3.31 – Nœud papillon autour de l'événement redouté central "Inflammation de produits combustibles" [Bernuchon et al., 2006]

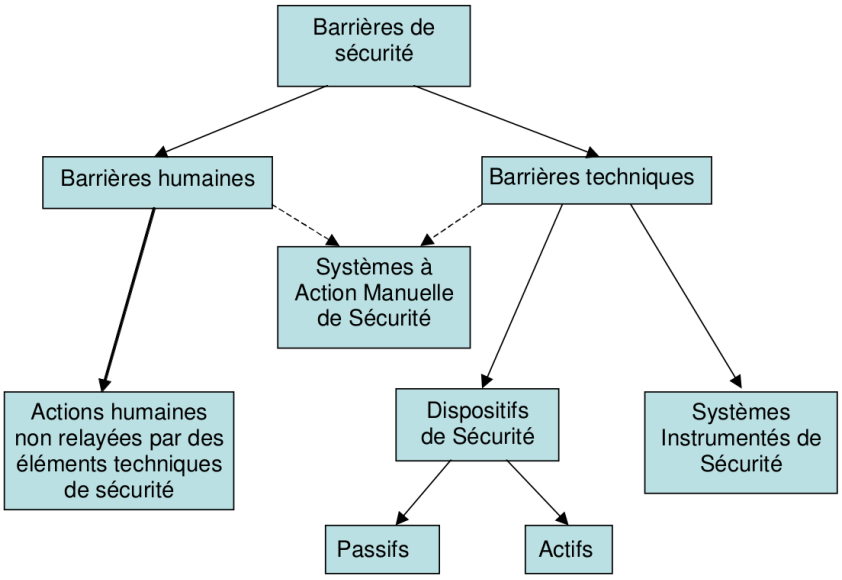


FIGURE 3.32 – Typologie des barrières de sécurité [Miche et al., 2009]

subsomption : dans l'exemple présenté dans la figure 3.31, les liens entre les différents événements à gauche de la porte OU et l'événement "Présence d'une source d'inflammation" n'expriment ainsi pas tous la subsomption, comme ça aurait été le cas avec un arbre de défaillances.

Réseaux bayésiens

Les formalismes présentés précédemment sont des formalismes graphiques, qui ne sont donc pas directement utilisables par des systèmes informatiques. [Amokrane, 2010] a donc proposé une opérationalisation des nœuds papillons en utilisant des réseaux bayésiens.

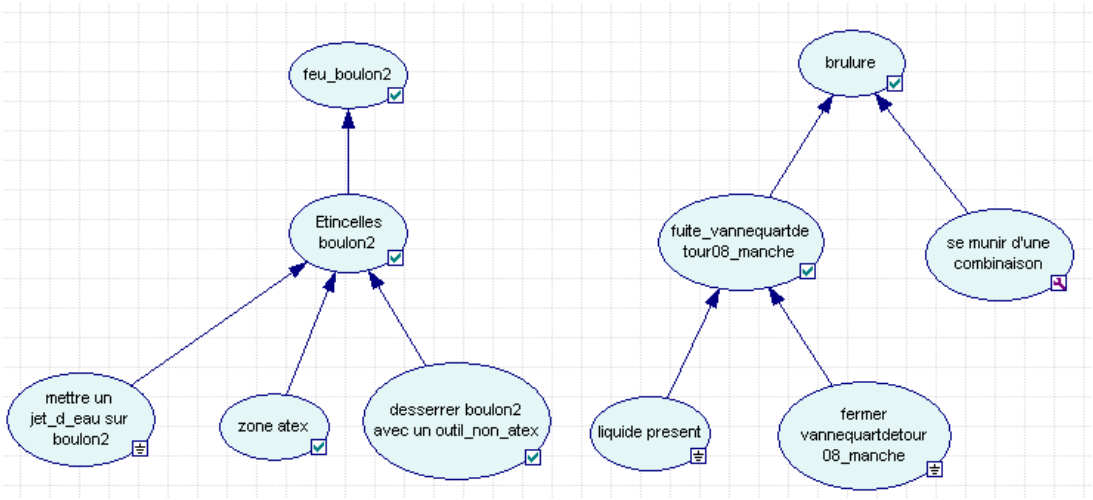


FIGURE 3.33 – Extrait du modèle de risques de [Amokrane, 2010]

Les noeuds correspondent aux événements et aux barrières identifiés lors de l'analyse de risques, et les tables de probabilité de chaque événement conditionnellement à ses causes sont déterminées par les experts du domaine. Un extrait du modèle ainsi créé est présenté en figure 3.33. Ces réseaux sont de plus associés à une ontologie permettant de catégoriser les événements accidentels et les barrières.
L'utilisation de réseaux bayésiens permet de calculer de manière dynamique la probabilité de réalisation d'un événement en fonction de l'état de l'environnement virtuel, cependant cette représentation pose certains problèmes. En particulier, l'absence d'alternatives et d'abstraction nécessite une duplication des noeuds pour prendre en compte chaque combinaison d'instances d'objets présents dans l'environnement.

Bilan

Les langages de représentation de scénarios issus de l'analyse de risques ont l'avantage d'être pensés pour être **utilisables par des experts du domaine**, non informaticiens. L'inconvénient de ces formalismes graphiques est qu'ils ne sont **pas directement interprétables** par des modules informatiques.
Une transposition de ces modèles, sous la forme d'arbres ET/OU pour les arbres de défaillance par exemple, est cependant possible. La **représentation sous forme de réseaux bayésiens** proposée par [Amokrane, 2010] permet ainsi de représenter à la fois les liens de causalité entre événements et les probabilités conditionnelles d'occurrence, et intègre de la sémantique via l'ajout d'une ontologie des risques. Toutefois, cette représentation **manque de possibilités d'abstraction**. De plus, il est difficile pour les experts de fournir des chiffres précis pour ce qui est des probabilités conditionnelles d'occurrence.
Les langages de représentation de scénarios accidentels sont pour la plupart **centrés sur un événement accidentel particulier**, et ne sont pas pensés pour fournir une vision globale des scénarios possibles pour une simulation donnée.
Dans l'ensemble, l'origine textuelle et graphique de ces langages fait qu'ils sont parfois **insuffisamment formalisés** : mélange de liens de causalité et de subsomption, notion d'événement floue (changement d'état du monde, situation, état initial...), etc. Les experts s'approprient le langage chacun à leur manière, ce qui rend la transposition informatique des modèles délicate.

3.3.4 Automates finis et graphes orientés acycliques

Dans les environnements virtuels, la représentation de scénarios prédéfinis visant à être exécutés dans l'environnement est souvent réalisée à l'aide d'**automates d'états finis**. Un automate fini est constitué d'un **ensemble fini d'états**, et d'un ensemble de **transitions** d'un état à un autre, et sera généralement représenté graphiquement sous la forme d'un graphe orienté. Dans le cas de scénarios, il s'agira le plus souvent de **graphes acycliques**, c'est à dire que le scénario possèdera un état initial et un ensemble d'états finaux. Les **réseaux de Petri** permettent d'étendre les automates finis en autorisant une même transition à avoir plusieurs états en sortie.

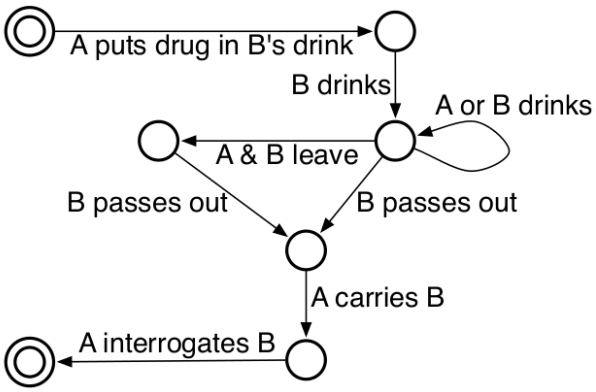


FIGURE 3.34 – Exemple de scénario représenté à l'aide d'un automate [Fitzgerald et al., 2009]

Un certain nombre de formalismes basés sur les automates ont été présentés dans la section 3.1.2 pour la représentation de l'activité. De tels formalismes peuvent également être utilisés pour représenter un scénario dans son ensemble, via la description exhaustive des événements et des transitions possibles entre ces événements. Deux cas se présentent alors. Soit les états de l'automate représentent les situations, et les transitions les actions possibles à partir de ces situations, comme c'est le cas pour le scénario représenté dans la figure 3.34, issue de [Fitzgerald et al., 2009]. Soit les états représentent les actions et les événements pouvant survenir dans l'environnement, et les transitions les séquences d'actions possibles, comme par exemple dans [Mollet and Arnaldi, 2006].
Ces formalismes sont utilisés pour la représentation des scénarios dans les approches basées sur des scénarios prédéfinis ou des graphes multilinéaires, mais également pour le suivi du déroulement des événements — par exemple, avec une représentation sous la forme de réseaux de Petri dans [Thomas et al., 2011].
Nous détaillerons ici deux langages basés sur des automates finis : Sluhrg [Devillers, 2001] et LORA [Mollet and Arnaldi, 2006], ainsi que YALTA [Burkhardt et al., 2009] un formalisme basé sur un graphe orienté acyclique qui vise à permettre la génération automatique d'automates finis.

Slurgh

Sluhrg (Scenario Language using HPTS running on GASP) [Devillers, 2001] est un langage de scénario dérivé du C++. Il est basé sur des automates finis hiérarchiques HPTS [Donikian, 2001]. Son objectif est de permettre la spécification à différents niveaux d'un scénario, de sorte à fournir une expérience prédictible et reproductible dans l'environnement virtuel.
Les états des automates correspondent à des situations dans l'environnement virtuel. Les transitions sont composées d'un ensemble de conditions et d'un ensemble d'actions. L'utilisation d'automates hiérarchiques permet de représenter un scénario soit comme une séquence d'événements élémentaires, soit comme une hiérarchie de sous-scénarios parallèles et de contraintes d'ordonnancement. Sluhrg permet ainsi de décrire des alternatives en attachant le déclenchement de certains sous-

scénarios à des conditions particulières. Il inclut également des instructions basées sur les relations temporelles d'Allen, qui permettent de définir très précisément la synchronisation entre les différents événements.
Sluhrg permet de décrire très finement un scénario ou un espace de scénarios, mais la complexité de sa syntaxe limite son utilisation à des programmeurs expérimentés. De plus, le choix d'un langage compilé empêche la modification dynamique de ces scénarios.

LORA

Le langage LORA (Language for Object Relation Application) [Mollet and Arnaldi, 2006] assimile le scénario de l'environnement virtuel à la procédure prescrite devant être réalisée par l'utilisateur. La représentation d'un scénario en LORA est composée d'un ensemble d'automates d'états finis hiérarchiques. Les étapes de ces automates correspondent soit à un sous-automate, soit à une action du modèle de domaine, c'est à dire à une transition appartenant à un automate du modèle d'interaction représenté en STORM (décrit dans la section 3.1.1). Ces étapes sont liées par des relations temporelles, de manière séquentielle ou parallèle puisqu'il est possible d'exprimer des alternatives à l'aide de conditions sur l'état du monde.
LORA ne permettant de représenter que les actions de l'utilisateur, [Gerbaud, 2008] a proposé une extension, LORA++, permettant de représenter des scénarios collaboratifs faisant intervenir plusieurs agents. Les étapes sont associées à des rôles, ces rôles pouvant être liés à l'utilisateur et/ou aux personnages virtuels.
La représentation du scénario en LORA permet de décrire le scénario comme un graphe multilinéaire d'actions, mais ne contient pas d'informations sur les effets de ces actions sur la simulation. De plus, il n'existe pas de distinction entre les relations qui relèvent simplement de la temporalité et celles qui relèvent de la causalité.

YALTA

YALTA (Yet Another Language for Task Analysis) est un langage permettant de représenter un scénario de manière graphique, sous la forme de graphes orientés acycliques adaptés des diagrammes d'activité UML [Burkhardt et al., 2009]. Un objectif de YALTA est de servir de support à la génération automatique d'automates d'états hiérarchiques HPTS++ [Lamarche and Donikian, 2002].
YALTA est un langage de modélisation de l'activité, mais permet également, grâce à la prise en compte de rôles, de modéliser de manière centralisée le scénario d'une simulation. Ici encore, ce dernier est assimilé à la procédure prescrite à réaliser dans l'environnement. Il est représenté sous la forme d'un ensemble de tâches, avec un formalisme textuel d'une part pour spécifier les propriétés des tâches, et un formalisme graphique d'autre part pour spécifier leur hiérarchie et leur ordonnancement temporel.

Les propriétés des tâches se réfèrent directement aux objets, actions et états décrits dans le modèle de domaine. L'ordonnancement est exprimé grâce à un ensemble de flux et de noeuds ; les noeuds de connexion permettent de spécifier la synchronisation entre différentes tâches ou encore la présence d'alternatives. De plus, les tâches sont munies de préconditions et de postconditions, qui permettent d'inférer les relations qui relèvent de la causalité. Un exemple de tâche modélisée avec YALTA est présenté dans la figure 3.35.
YALTA est un langage très complet, et qui, par conséquence, s'avère complexe à maîtriser, notamment car la représentation d'un espace de scénarios nécessite de décrire de manière exhaustive les flux et contraintes de synchronisation entre les actions et événements intervenant dans ces scénarios.

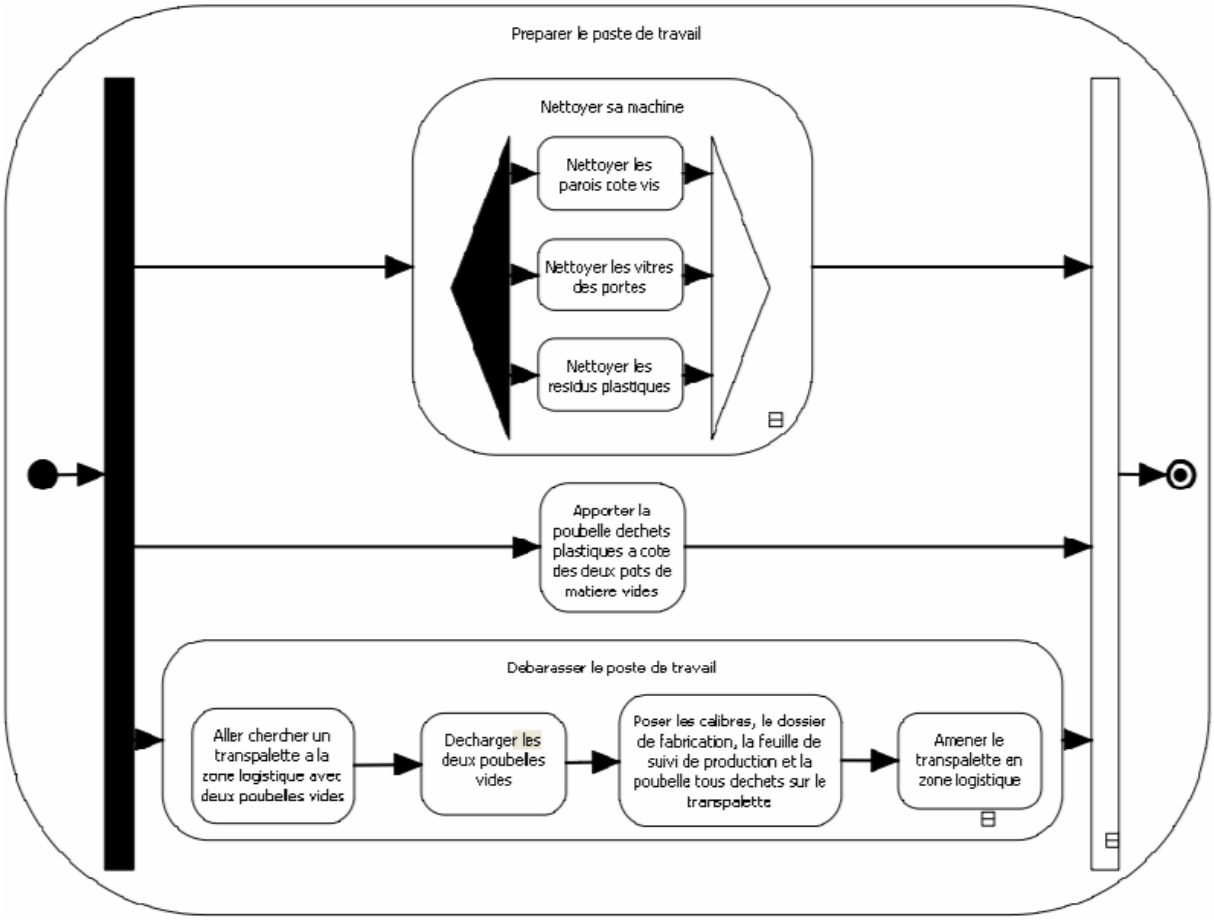


FIGURE 3.35 – Exemple de tâche modélisée avec YALTA [Burkhardt et al., 2009]

Bilan

Les automates finis et réseaux de Petri permettent de **décrire très finement** des espaces de scénarios. L'ajout d'instruction temporelles, comme dans Sluhrg, permet de spécifier la **synchronisation** entre les différents événements, et les éléments représentés dans le modèle peuvent référencer les états du monde, objets et actions définies dans le modèle de domaine, de sorte à ce que la représentation du scénario puisse être **directement exécutable** dans l'environnement ou puisse servir pour le **suivi du déroulement des événements**.
Cependant, ces représentations supposent une **définition exhaustive** des transitions entre événements, et supportent difficilement le passage à l'échelle. Les automates hiérarchiques permettent de clarifier la représentation, mais restent **difficiles d'accès pour les non informaticiens**.
Des travaux ont donc été entrepris pour proposer, à l'instar de YALTA, des formalismes plus intelligibles et qui permettent de générer automatiquement des automates finis. Cependant, ces formalismes nécessitent eux-aussi une définition exhaustive des transitions entre les événements des scénarios.

3.3.5 Plans et points clés partiellement ordonnés

Les systèmes faisant appel à des moteurs de planification utilisent au contraire des représentations des scénarios sous forme de plans. Un plan est constitué d'une **séquence d'actions** menant d'une **situation initiale** à une **situation finale** (situation "but").
Dans le domaine de la narration interactive, la représentation sous forme de plan est considérée comme particulièrement pertinente pour les scénarios, puisqu'elle permet de représenter la structure causale d'une histoire [Magerko, 2007]. Dans cette optique, de nombreux travaux utilisent non

pas de simple plans, mais des **plans partiellement ordonnés**. Ces derniers permettent en effet de distinguer ce qui, dans l'ordonnancement des actions, relève de liens de causalité, et ce qui relève simplement de l'ordonnancement temporel. Les relations de causalité sont représentées par des liens entre les actions, tandis que l'ordonnancement temporel est éventuellement décrit par des contraintes supplémentaires, permettant ainsi de réaliser un même plan en ordonnant les actions de différentes manières.
A la notion de plans est associée celles de **points clés**, ou *plot points*. Ces points clés sont liés au concept de *landmarks* en planification, qui correspondent aux étapes par lesquelles doit passer le plan pour aboutir à la situation but. Ces étapes sont classiquement extraites à partir du problème de planification, mais certains travaux utilisent des points clés prédéfinis pour décrire des espaces de scénarios d'intérêt. Ces points clés peuvent être décrits soit comme des ensembles d'états du monde qui devront être atteints durant le déroulement du plan, soit comme des actions que le plan devra contenir. A l'instar des actions d'un plan, les points clés peuvent être eux aussi **partiellement ordonnés** à l'aide de liens temporels.
Nous détaillerons ici les représentations utilisées dans deux systèmes : les plans partiellement ordonnés POCL, utilisés notamment dans Mimesis [Riedl et al., 2003], et les points clés partiellement ordonnés utilisés dans IDA [Magerko, 2005].

Plans partiellement ordonnés — Mimesis

Les travaux de R. Michael Young, entre autres sur le système Mimesis [Riedl et al., 2003], utilisent une représentation des scénarios sous la forme de plans POCL (Partial-Order Causal Link).

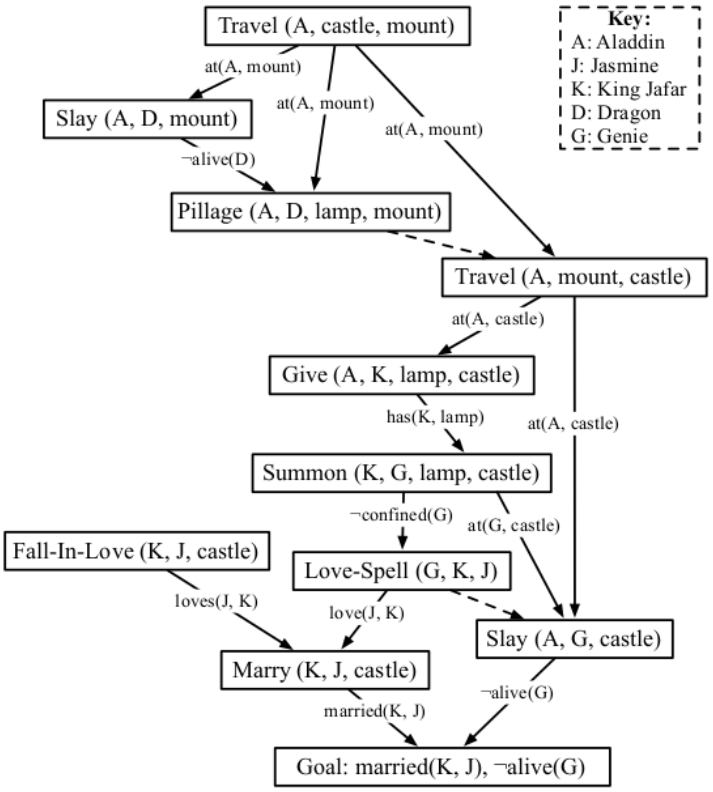


FIGURE 3.36 – Extrait d'un plan POCL [Riedl and Young, 2010]

Un plan POCL contient une situation initiale, une situation finale, un ensemble d'étapes, un ensemble de liens de causalité et éventuellement des contraintes d'ordonnancement entre plusieurs

étapes du plan. Un exemple de plan est présenté dans la figure 3.36. Chaque étape correspond à une action et possède un ensemble de paramètres. Les étapes contiennent également les préconditions et les effets liés à la réalisation de l'action. Ainsi, un plan n'est valide que si les précondition de chaque étape sont satisfaites soit par un lien de causalité provenant des effets d'une étape précédente, soit par l'état initial du monde.
Les auteurs ont étendu la planification POCL en proposant également la planification DPOCL (Decompositional Partial-Order Causal Link) [Young and Moore, 1994]. Celle-ci permet de construire des plans hiérarchiques en intégrant des étapes abstraites (par exemple, "Braquage de banque") qui se décomposent en un ensemble d'actions.

Points clés partiellement ordonnés — IDA

Le système IDA (Interactive Drama Architecture) proposé par [Magerko, 2005] utilise une représentation de scénarios sous la forme d'un graphe de points clés partiellement ordonnés. Un extrait de cette représentation est présenté dans la figure 3.37.
Chaque point clé contient : un ensemble de conditions, qui décrivent un ensemble de situations du monde dans lesquels chacune de ces assertions est vérifiée ; un ensemble d'actions devant être exécutées dans l'une de ces situations ; une contrainte temporelle indiquant la durée durant laquelle les assertions doivent rester vraies. Un lien entre deux points clés A et B indique que le point clé A doit être réalisé avant le point clé B.

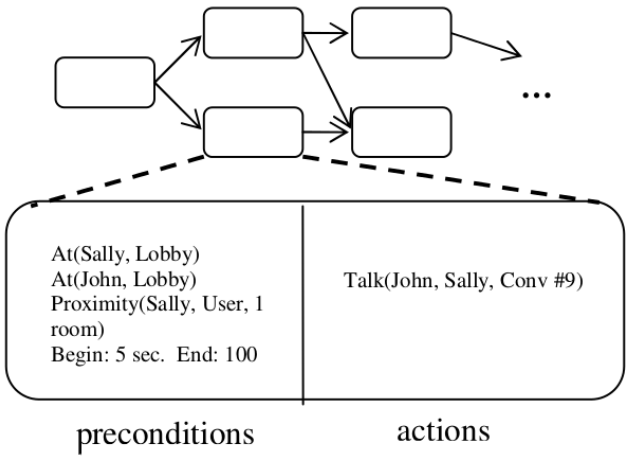


FIGURE 3.37 – Points clés partiellement ordonnés [Magerko, 2005]

Ce formalisme permet de représenter un espace de scénarios constitué de l'ensemble des scénarios possibles contenant les ordonnancements de points clés autorisés par les contraintes temporelles. La visibilité sur cet ensemble est cependant limitée, du fait des interdépendances entre les relations d'ordonnancement, les contraintes temporelles définies pour chaque point clé, et les préconditions associées qui peuvent être mutuellement exclusives d'un point clé à un autre, empêchant ainsi leur réalisation parallèle.

Bilan

La représentation sous forme de plans permet de représenter des scénarios générés dynamiquement par un moteur de planification, et en particulier de représenter les **liens de causalité** entre les actions et d'associer ces liens à des états particuliers de l'environnement virtuel.

Ces représentations permettent de décrire des **scénarios particuliers**, voire des ensembles de **variantes** d'un même scénario dans le cas de plans partiellement ordonnés. Par contre ils ne permettent pas de représenter des **alternatives**, ni d'indiquer des informations sur la **synchronisation** des différents événements.
La représentation sous forme de points clés partiellement ordonnés est moins contraignante en ce qu'elle permet de réduire la description à celle des **événements les plus importants** du scénario. Les liens entre les éléments du graphe relevant de la temporalité et non de la causalité, il n'est pas nécessaire de décrire de manière exhaustive toutes les actions intervenant dans le déroulement du scénario, comme c'est le cas pour les plans. Les points clés eux-mêmes peuvent être très simples — contenant par exemple une assertion sur l'état du monde — ou plus complexes, comme c'est le cas pour ceux d'IDA qui contiennent à la fois actions, préconditions et contraintes temporelles.
De plus, pour les points clés d'IDA comme pour les plans POCL, les étapes référencent des instances d'objets dans l'environnement ; ces représentations sont donc **limitées au niveau de l'abstraction** pour la représentation de trames scénaristiques ou d'espaces de scénarios.

3.3.6 Bilan et positionnement

Ces différents formalismes représentent tous les scénarios comme des **graphes**, mais y font figurer des éléments différents. Certains utilisent les **actions** comme éléments principaux, d'autres les **situations**. De même, les liens reliant ces éléments peuvent relever de relations de **causalité**, de **subsomption**, de **décomposition** ou encore d'**ordonnancement temporel**. Selon les besoins auxquels ils répondent, ils intègrent également des éléments spécifiques : alternatives et préconditions, éléments pédagogiques, informations sur la synchronisation entre les événements, motivations et états internes des personnages virtuels...
Les scénarios et espaces de scénarios représentés ne sont en effet pas destinés à la même utilisation. D'un côté, on trouve des formalismes voués à la **spécification manuelle de scénarios et d'espaces de scénarios** en amont de l'utilisation de l'environnement virtuel, et qui vont servir de support à ce dernier : scénarios pédagogiques, modèles d'analyses de risques opérationnalisés, automates finis, points clés partiellement ordonnées, etc. De l'autre, il s'agit de formalismes destinés à représenter des **scénarios linéaires**, **générés dynamiquement** pour être exécutés dans l'environnement virtuel ou pour servir de traces, ou créés *a posteriori* : plans et modèles de compréhension d'histoires.
Selon qu'il s'agisse de représenter un scénario particulier, une trame scénaristique ou un espace de scénarios, les besoins en termes de formalismes seront différents.
Ainsi, la **représentation d'un scénario particulier** va nécessiter un formalisme **interprétable**, qui décrive les **actions de l'utilisateur**, mais aussi celles des **personnages virtuels** et les **comportements des systèmes techniques simulés**, et référence directement les **instances** d'objets de l'environnement virtuel concernées. Afin de pouvoir suivre la réalisation de ce scénario, il est nécessaire que les liens de causalité reliant ses éléments soient associés à des états particuliers de l'environnement, en indiquant les **préconditions** nécessaires à l'exécution de chaque action pour permettre la détection des situations où le scénario n'est plus réalisable.
Puisque nous souhaitons générer dynamiquement des scénarios menant à des situations particulières, afin de les réaliser dans l'environnement virtuel, nous opterons pour une représentation des scénarios sous forme de **plans partiellement ordonnés**, qui pourront être élaborés par le moteur de scénarisation. Le formalisme utilisé est présenté dans la partie 6.2.
La représentation d'un **espace de scénarios d'intérêt** en vue de spécifier les objectifs scénaristiques fait appel à des propriétés différentes.
Tout d'abord, le formalisme utilisé doit être lui aussi **interprétable**, et en particulier permettre d'évaluer la **valeur de vérité** liée à la réalisation de chaque situation, afin de pouvoir situer de manière dynamique l'avancement des événements de l'environnment virtuel par rapport aux différents scénarios considérés.
Il doit également être **intelligible**, afin que les espaces de scénarios d'intérêt puissent être renseignés

directement par les experts. Dans notre cas, il s'agira de permettre de renseigner des connaissances issues d'analyses de risques, afin de pouvoir simuler des scénarios accidentels pertinents pour la formation ou l'aide à la décision.
Enfin, le formalisme doit être **expressif**, et permettre de représenter non seulement des **relations de causalité** entre événements mais également des **conjonctions** et des **disjonctions** au niveau de ces relations. On souhaite également pouvoir représenter des relations de **subsomption**, afin de raisonner sur les scénarios à haut niveau. De même, il est nécessaire que le formalisme permette une certaine **abstraction**, par l'utilisation de variables et de concepts plutôt que de référencer directement des instances d'objets. Enfin, il doit permettre la spécification d'un ensemble de **propriétés** sur les événements, comme leur probabilité d'occurrence et leur gravité.
Nous proposons un **méta-modèle de la causalité** nommé **CAUSALITY-DL**, présenté dans la section 7.1. CAUSALITY-DL est utilisé pour représenter des **espaces de scénarios d'intérêt**. Un parcours particulier dans un modèle de causalité correspond à une **trame scénaristique**, c'est à dire à un ensemble de points clés non instanciés et partiellement ordonnés.

Deuxième partie

Contributions

Chapitre 4

SELDON – Modèle pour la scénarisation d'environnements virtuels

Sommaire

Notre problématique porte sur la **scénarisation** d'environnements virtuels, et plus particulièrement sur la conciliation de la **liberté d'action** de l'apprenant, du **contrôle dynamique** de la simulation, de la **cohérence** des comportements présentés et de l'**adaptabilité** de l'environnement virtuel. La présentation des travaux de la littérature réalisée dans le chapitre 2 nous a permis de constater que les systèmes de scénarisation existants n'offraient pas de compromis satisfaisant entre ces quatre aspects. Les différentes approches favorisent en effet un ou deux de ces aspects au détriment des autres. Les systèmes basés sur l'utilisation de personnages semi-autonomes, en particulier, permettent à la fois liberté d'action, adaptabilité et contrôle mais pèchent au niveau de la cohérence des comportements des personnages. Les travaux récents s'intéressant au contrôle de personnages autonomes, en particulier les systèmes Thespian [Si, 2010] et Virtual Storyteller [Swartjes, 2010], se focalisent sur un contrôle du scénario à court terme qui respecte cette cohérence, mais ne permettent pas de contrôler le scénario dans sa globalité. Leur étude nous a cependant permis de dégager un certain nombre d'éléments qui nous semblent pertinents pour la proposition d'un modèle de scénarisation d'environnements virtuels.

Nous proposons ainsi le modèle **SELDON**, pour pour ScEnario and Learning situations adaptation through Dynamic OrchestratioN. SELDON est un modèle pour la **scénarisation extrinsèque** d'environnements virtuels, qui permet d'adapter les scénarios de manière dynamique via la **prédiction** et l'**ajustement** de l'évolution de **simulations autonomes**, à partir d'un **ensemble de modèles** partagés par les modules de simulation et les modules de scénarisation, de sorte à mener à des **situations pertinentes** en regard de l'utilisateur.

Ce chapitre présente le modèle SELDON. Nous présentons dans un premier temps l'approche générale du modèle, puis ses trois composantes principales, portant chacune sur un aspect de la problématique. Le moteur **TAILOR**, chargé de la **génération de situations d'apprentissage**, vise à répondre à la question suivante : *"Comment déterminer dynamiquement les situations d'apprentissage vers lesquelles guider le scénario, afin qu'elles soient pertinentes au regard du profil de l'apprenant et d'objectifs de formation ?"*. Le moteur **DIRECTOR**, chargé de la **génération et de la réalisation des scénarios dans la simulation**, porte sur la problématique suivante : *"Comment guider dynamiquement le scénario d'un environnement virtuel vers ces situations en respectant la liberté d'action de l'utilisateur et la cohérence des comportements de la simulation, tout en favorisant l'adaptabilité de cet environ-*

nement virtuel ?". Enfin, la suite de **langages HUMANS-DL** adresse quant-à-elle cette question : *"A partir de quelles connaissances peut-on réaliser cette scénarisation, et comment représenter ces connaissances ?"*.
Les chapitres suivants détaillent les contributions réalisées dans le cadre de cette thèse, c'est-à-dire le moteur DIRECTOR et l'ensemble de langages HUMANS-DL, le moteur TAILOR faisant l'objet de la thèse de doctorat de Kevin Carpentier [Carpentier et al., 2013]. Le chapitre 5 présente ainsi les langages de représentation du contenu, DOMAIN-DL et ACTIVITY-DL, qui permettent d'exprimer les connaissances nécessaires à l'exécution de la simulation. Le chapitre 6 présente le moteur DIRECTOR, qui permet de prédire et d'ajuster le scénario de la simulation pour mener à des situations particulières. Enfin, le chapitre 7 porte sur la représentation des objectifs scénaristiques, et en particulier sur le langage CAUSALITY-DL, qui permet la prise en compte par DIRECTOR de trames scénaristiques d'intérêt et de contraintes globales sur les propriétés des scénarios.

4.1 Approche

Le modèle SELDON, présenté dans la figure 4.1, est un modèle pour la **scénarisation extrinsèque** d'environnements virtuels. Cela signifie qu'il considère la scénarisation comme une étape supplémentaire de cadrage d'un environnement virtuel existant, et non comme partie intégrante du processus de conception de cet environnement. La scénarisation consiste donc ici à réduire l'espace des scénarios possibles de la simulation, en vue d'une utilisation particulière. La scénarisation extrinsèque permet de favoriser l'**adaptabilité** du système, en permettant de réutiliser un même module de scénarisation avec différentes simulations.

SELDON distingue ainsi le **module de scénarisation** du **module de simulation interactive**, qui gère la simulation du monde, la génération de comportements de personnages virtuels autonomes, et la gestion des interactions des utilisateurs.
La simulation est conçue pour pouvoir évoluer de manière autonome, selon une approche émergente. Un moteur de simulation du monde met à jour l'état du monde en fonction des règles de comportement des systèmes techniques simulés, des actions déclenchées par les utilisateurs et des actions des personnages virtuels. Ces derniers sont également autonomes, et leur comportement est géré par un moteur basé sur un système multi-agent et pouvant intégrer différentes architectures cognitives. Cette approche favorise elle aussi l'**adaptabilité** du système en permettant l'**émergence** de situations et de scénarios qui ne soient pas définis explicitement au préalable.
Le modèle SELDON vise à **adapter dynamiquement** le scénario de l'environnement virtuel pour le rendre pertinent par rapport aux **traces d'activité** de l'utilisateur. Le module de scénarisation sépare pour cela la génération des objectifs scénaristiques dynamiques de la génération et la réalisation des scénarios eux-mêmes. Dans un premier temps, le moteur de **génération des objectifs scénaristiques dynamiques** se base sur les traces d'activité pour générer un ensemble de situations ou de propriétés du scénarios qui soient pertinentes en regard du profil et de l'activité de l'utilisateur. Puis, le moteur de **prédiction et d'ajustement du scénario** utilise ces objectifs pour générer un scénario souhaitable. Pour cela, il utilise les modèles de connaissances qui sous-tendent la simulation pour prédire son évolution, et oriente cette évolution à partir d'un ensemble d'ajustements indirects sur la simulation. Ces ajustements sont transmis aux moteurs de simulation du monde et de génération des comportements des personnages virtuels afin d'influencer leur évolution, de sorte que le scénario prédit par le module de scénarisation se réalise effectivement dans l'environnement virtuel.
De même que les choix d'actions des personnages virtuels autonomes ne sont pas contrôlés directement par le module de scénarisation, SELDON favorise une approche ouverte pour le choix d'actions des utilisateurs. Ces derniers ont ainsi la possibilité de choisir à partir de l'ensemble des actions qui sont réalisables dans la simulation à un moment donné — et ce, que cette réalisation mène à un succès ou à un échec des conséquences qu'ils attendent pour cette action —, plutôt qu'à partir d'une

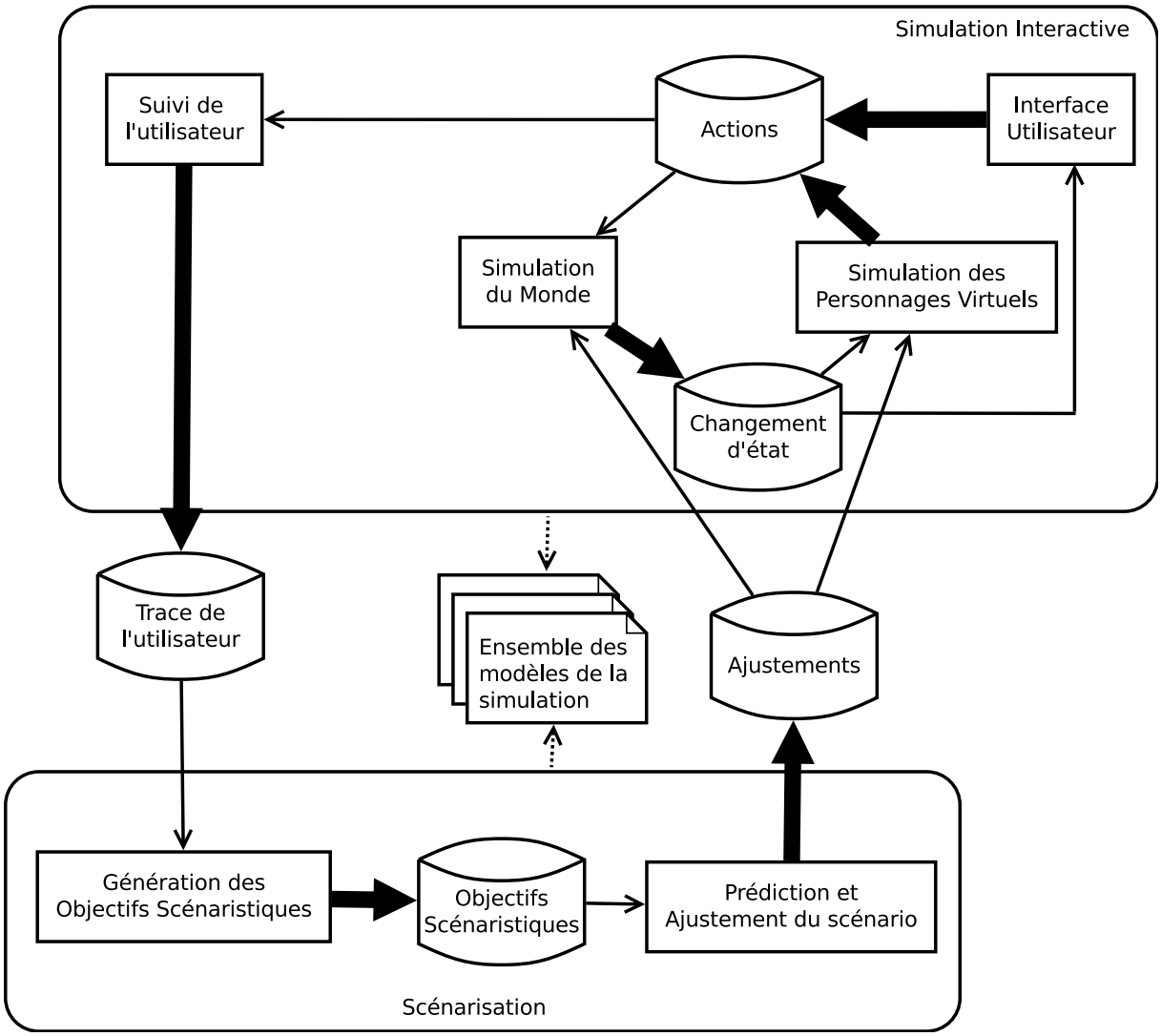


FIGURE 4.1 – Modèle SELDON

présélection dépendante d'un scénario prédéfini. Cette **liberté d'action** est rendue possible par la génération dynamique du scénario, les actions de l'utilisateur étant prédites à la manière de celles des personnages virtuels autonomes, et contrôlées indirectement dans une moindre mesure par le biais d'un contrôle environnemental.

Le modèle SELDON vise à favoriser la construction et la scénarisation d'environnements virtuels qui aient une forte **validité écologique**, c'est à dire tels que les situations et les scénarios générés soient proches de ceux rencontrés en conditions réelles. Les comportements des systèmes techniques et des personnages virtuels sont donc calculés à partir de modèles de connaissances issus d'analyses terrain. Ces mêmes modèles sont utilisés pour prédire et orienter indirectement ces comportements, afin de respecter leur cohérence. De même, les trames scénaristiques d'intérêt correspondent à des chaînes de causalité identifiées lors d'analyses terrain, en particulier d'analyses de risques. Ces connaissances sont représentées par le biais d'un **ensemble de langages de représentation** liés entre eux, qui expriment différentes vues sur une même réalité de terrain. Cet ensemble de langages est désigné sous le nom d'**HUMANS-DL**.

La génération des objectifs scénaristiques et la prédiction et l'ajustement des scénarios sont réalisés respectivement au travers des modules **TAILOR** et **DIRECTOR**, comme indiqué dans la figure 4.2.

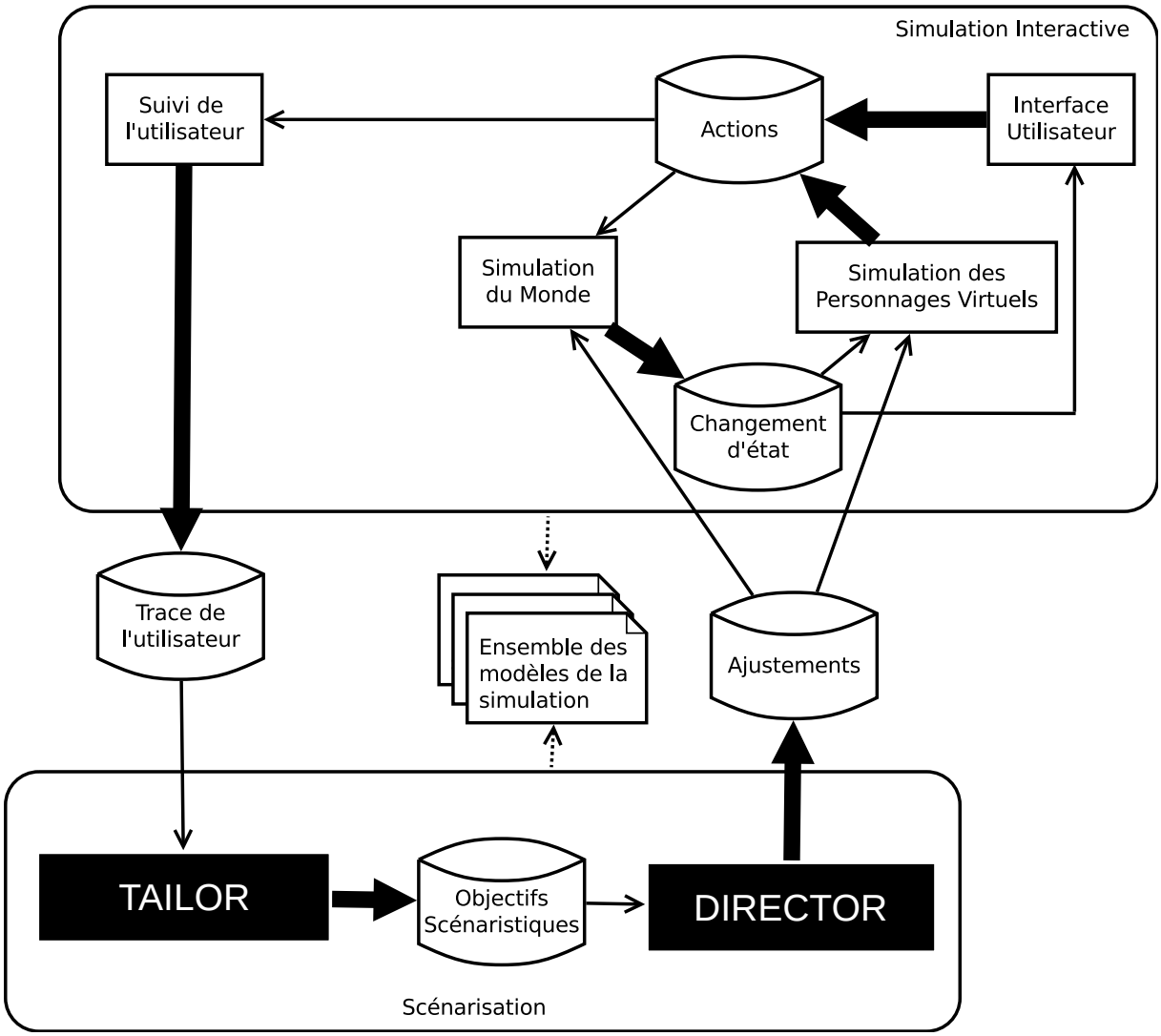


FIGURE 4.2 – Composantes principales du modèle SELDON

4.2 Vers quoi scénariser – le moteur TAILOR

Le modèle SELDON distingue ainsi la génération des objectifs scénaristiques dynamiques de leur réalisation, la génération étant l'objet du moteur **TAILOR**. Cette distinction permet d'avoir un système de scénarisation qui soit **modulaire**, offrant la possibilité d'utiliser le module TAILOR pour calculer automatiquement des objectifs scénaristiques, ou bien de remplacer ce module par un ensemble d'objectifs scénaristiques prédéfinis ou par une interface permettant à un formateur d'entrer directement des objectifs dans le système durant une session de formation, en fonction de son appréciation des performances de l'apprenant. Cette modularité concourt, ici encore, à notre objectif d'**adaptabilité**.
Le moteur TAILOR génère en sortie un ensemble d'objectifs scénaristiques dynamiques, sous la forme de situations prescrites, de situations proscrites, et de contraintes sur les propriétés globales du scénario. Ces objectifs sont décrits dans le chapitre 7.
TAILOR fait l'objet de la thèse de doctorat de Kevin Carpentier ; davantage de détails peuvent être trouvés dans [Carpentier et al., 2013].

4.3 Comment scénariser – le moteur DIRECTOR

Le moteur **DIRECTOR** vise à réaliser dans l'environnement virtuel ces objectifs scénaristiques définis dynamiquement. Afin de pouvoir prendre en compte des propriétés globales du scénario, il convient de considérer celui-ci dans son ensemble. DIRECTOR utilise donc une approche basée sur de la **planification** de scénario, plutôt que sur de la sélection dynamique des événements, car celle-ci ne permet d'opérer qu'un contrôle local.
La réalisation d'un scénario ainsi généré au travers du contrôle direct, comportemental ou motivationnel, de personnages virtuels semi-autonomes ou de changements instantanés des états des systèmes simulés nuirait à la cohérence des comportements. Plutôt que de considérer le scénario généré comme prescriptif, DIRECTOR adopte une **approche prédictive**, en utilisant les modèles de connaissances pour prédire l'évolution de la simulation — celle des systèmes techniques comme les comportements des personnages virtuels autonomes. Ces comportements peuvent alors être orientés par le biais d'ajustements qui en respectent la **cohérence**.
Ces ajustements, et la façon dont les scénarios sont générés et réalisés, sont décrits en détail dans le chapitre 6 portant sur DIRECTOR.

4.4 A partir de quoi scénariser – les langages HUMANS-DL

Afin de calculer les prochaines étapes de la simulation, de générer les objectifs scénaristiques et de prédire les scénarios, les différents moteurs du modèle SELDON doivent faire appel à un ensemble de connaissances dépendantes du domaine considéré.
Après avoir étudié, dans le chapitre 3, les langages utilisés pour la représentation du contenu scénaristique, des objectifs scénaristiques et des scénarios, nous avons constaté que les langages existants ne répondaient pas à nos besoins en termes d'expressivité, d'interprétabilité, d'intelligibilité et de modularité.
La proposition du modèle SELDON est donc associée à celle de la suite de langages HUMANS-DL, qui contient trois formalismes : DOMAIN-DL, ACTIVITY-DL et CAUSALITY-DL.
DOMAIN-DL et **ACTIVITY-DL** sont des langages de **représentation du contenu scénaristique**, utilisés pour écrire respectivement des modèles de domaine et des modèles d'activité. Les modèles qu'ils permettent de représenter sont utilisés notamment pour le calcul des états de la simulation, et pour la génération des scénarios prédictifs.
CAUSALITY-DL est un langage de **représentation d'espaces de scénarios d'intérêt** sous forme de graphe de causalité, utilisé pour exprimer des **objectifs scénaristiques statiques**. Les modèles qu'il permet d'écrire servent notamment à la génération de certains objectifs scénaristiques dynamiques (situations prescrites et proscrites) par le moteur TAILOR, à la génération des scénarios par DIRECTOR, mais également à la génération d'explications par le module de suivi de l'utilisateur, chargé de générer les traces de l'environnement virtuel.
Ces trois langages correspondent à différentes vues sur une même réalité de terrain, et ont vocation à être remplis par différents experts (ergonomes, analystes de risques, etc.).
Les langages DOMAIN-DL et ACTIVITY-DL sont présentés dans le chapitre 5, qui porte sur la représentation du contenu scénaristique, tandis que le langage CAUSALITY-DL est présenté dans le chapitre 7, qui porte sur la représentation des objectifs scénaristiques.

Chapitre 5

Représentation du contenu scénaristique

Sommaire

Le contenu scénaristique correspond à l'ensemble des événements unitaires pouvant intervenir dans le scénario, et, de manière plus large, dans l'environnement virtuel. Nous avons ici choisi d'opter pour une scénarisation extrinsèque, afin de favoriser l'adaptabilité de l'environnement virtuel. Le contenu scénaristique représenté va donc servir à la fois au moteur de scénarisation — ses éléments correspondant aux éléments de base des scénarios — et à la simulation — l'évolution de cette dernière étant calculée à partir des comportements y étant définis.
Suite à l'étude des langages et formalismes existants, nous avons identifié un ensemble de propriétés nécessaires pour pouvoir utiliser ces langages pour scénariser des environnements virtuels en assurant à la fois leur adaptabilité et leur cohérence : la **modularité**, l'**intelligibilité**, l'**expressivité** et l'**interprétabilité**.

L'intelligibilité est nécessaire car ces langages ont vocation à être directement utilisés par des experts (opérateurs, ergonomes, formateurs, etc.) pour écrire les différents modèles, de sorte à créer un environnement virtuel qui soit écologiquement valide. Le remplissage direct des modèles par les experts, sans passer par une transcription réalisée par des informaticiens, permet en effet de s'affranchir de la vision informatique de l'humain comme un système technique, et de laisser les experts créer et confronter leurs propres modèles sans passer par le filtre d'interprétation de l'informaticien. Or différents experts seront focalisés sur différents modèles, ou sur différentes composantes d'un même modèle : le fonctionnement d'un système technique particulier, les procédures prescrites, les compromis réalisés par les agents sur le terrain... De ce fait, nous cherchons à favoriser la modularité via la séparation du modèle du domaine et du modèle de l'activité, en permettant ainsi qu'ils soient renseignés par des experts différents, et réutilisés en totalité ou en partie d'une application à l'autre.
Nous proposons donc de représenter le contenu scénaristique à travers deux modèles :

- un modèle du domaine, qui décrit les objets du monde et leurs états, les actions pouvant être réalisées sur ces objets et les comportements qui en résultent ;
- un modèle de l'activité, qui décrit les enchaînements et le contexte de réalisation de ces actions en termes de tâches et de buts, tels que les opérateurs se les représentent.

Afin d'assurer l'interprétabilité de ces modèles, il est nécessaire qu'ils soient liés entre eux, et les différents éléments unitaires du modèle d'activité seront donc définis en référence aux éléments du modèle du domaine.
Nous avons ainsi participé, en collaboration avec notre équipe de recherche, à la proposition des deux langages qui sont décrits dans ce chapitre : DOMAIN-DL, pour la modélisation du domaine, et ACTIVITY-DL, pour la modélisation de l'activité.

5.1 DOMAIN-DL – Représentation du domaine

Le modèle du domaine décrit les objets du monde, ainsi que les actions et les comportements qui leur sont associés. Pour le représenter, nous avons proposé le langage DOMAIN-DL. Les travaux sur DOMAIN-DL ont été réalisés avec Kevin Carpentier [Carpentier et al., 2013], et font suite aux réflexions sur le langage COLOMBO et ses limites [Edward et al., 2010] [Edward, 2011]. Nous détaillons ici l'approche choisie, ainsi que le formalisme lui-même.

5.1.1 Approche

Besoins

Nous avons identifié précédemment quatre propriétés nécessaires pour les langages de représentation du contenu scénaristique : modularité, intelligibilité, interprétabilité et expressivité.
Pour le langage de représentation du domaine, la **modularité** est nécessaire afin de pouvoir ajouter, supprimer ou modifier facilement des types d'objets, des actions ou encore des instances d'objets au modèle. Elle est également nécessaire pour l'intégration de modèles de domaine existants.
Le modèle ayant vocation à être renseigné par les experts, le langage doit être, dans une certaine mesure, **intelligible** et accessible à des non-informaticiens.
Le langage doit également être **interprétable**, pour que les modules informatiques utilisant les modèles puissent interroger directement l'état du monde, exécuter les comportements pour faire évoluer la simulation, ou encore raisonner sur les règles qui régissent les actions et notamment leurs préconditions et leurs effets.
Enfin, il doit être assez **expressif** pour permettre de représenter les actions des personnages virtuels comme celles des utilisateurs et leurs effets sur le monde, de même que les comportements intrinsèques des objets qui ne sont pas liés à des actions spécifiques, et ceci qu'il s'agisse de comportements ponctuels ou continus.

En particulier, les exemples suivants ont été identifiés lors d'analyses terrain sur nos différents cas d'application comme devant pouvoir être représentés dans le langage :

- **des actions ayant des effets binaires ou numériques**, comme une action "ouvrir" permettant de faire passer une vanne d'un état "fermé" à un état "ouvert" ou une action "tourner" prenant un angle en paramètre et faisant faire à la poignée de la vanne une rotation de l'angle en question ;
- **des préconditions déterminant soit la faisabilité d'une action, soit son succès** : par exemple, l'actionnement d'un extincteur n'est faisable que si sa goupille de sécurité est retirée, tandis que son utilisation n'aura l'effet voulu sur l'incendie que si la classe de l'extincteur est en accord avec la nature du feu ;
- **des actions ayant des conséquences sur des objets liés dynamiquement aux objets cibles**, par exemple une action "tourner" sur la poignée d'un bras de chargement d'une pompe à essence, qui permet de verrouiller le bras sur lequel est monté la poignée à la vanne du camion sur lequel est actuellement branché ce bras ;
- **des liens sémantiques entre plusieurs états**, comme le fait qu'une porte ayant un angle d'ouverture supérieur à 45˚soit considérée comme ouverte ;
- **des comportements liés à un concept donné**, et non à une instance d'objet particulière, comme un comportement de rotation ;
- **des comportements faisant évoluer un état en continu** : l'action "utiliser" sur un extincteur permet par exemple de déverser n litres de mousse par pas de temps ;
- **des comportements déclenchés automatiquement sous certaines conditions**, par exemple l'extinction d'un feu lorsque le rapport entre la quantité de mousse déversée et l'ampleur du feu a atteint un certain seuil, ou le débordement d'une cuve lorsque le nombre de litres de liquide versés dans la cuve dépasse sa capacité.

Proposition

A l'instar de [Chevaillier et al., 2009], nous distinguons trois types de connaissances nécessaires à la description d'un modèle de domaine :

- les **connaissances statiques sur le domaine**, c'est à dire les objets présents dans l'environnement, leurs états et relations ;
- les **possibilités d'interaction**, c'est à dire les actions réalisables sur ces objets par les agents, qu'ils soient des utilisateurs ou des personnages virtuels autonomes ;
- les **comportements**, qui définissent les changements d'états ou de relations entre les objets causés soit par des comportements intrinsèques soit par des actions.

Pour représenter ces trois types de connaissances dans un méta-modèle du domaine, nous adoptons une posture ontologique. Une ontologie, selon [Gruber, 1993], est la spécification explicite d'une conceptualisation. [Uschold et al., 1996] précise cette notion en proposant la définition suivante, traduite par [Charlet, 2002] :

> « Une ontologie implique ou comprend une certaine vue du monde par rapport à un domaine donné. Cette vue est souvent conçue comme un ensemble de concepts – e.g. entités, attributs, processus –, leurs définitions et leurs interrelations. On appelle cela une conceptualisation. [...] Une ontologie peut prendre différentes formes mais elle incluera nécessairement un vocabulaire de termes et une spécification de leur signification. [...] Une ontologie est une spécification rendant partiellement compte d'une conceptualisation. »

L'utilisation d'une ontologie pour la modélisation d'un domaine apporte de nombreux avantages, comme mentionnés par [Noy and Mcguinness, 2001]. Elle permet la **réutilisation** des connaissances du domaine et le partage d'une **compréhension commune** de l'information et de sa structuration par les différentes personnes impliquées dans la conception de l'environnement virtuel. Elle permet

également de rendre **explicites** les suppositions et les hypothèses qui sont faites lors de la création du modèle, du fait de la subjectivité inhérente au processus de modélisation, ainsi que de **raisonner** sur le modèle à l'exécution, notamment par la réalisation d'**inférences**, c'est à dire la création de nouvelles connaissances à partir des connaissances explicitées dans le modèle.
L'utilisation d'une ontologie permet également de séparer les connaissances sur le domaine des connaissances opérationnelles. Nous considérons ainsi un quatrième type de connaissances : les **connaissances dynamiques sur le domaine**. Ces connaissances vont concerner les instances d'objets actuellement présentes dans l'environnement, leurs états courants et l'historique de ces états, de même que les relations entre elles (par exemple, le fait qu'un bras de chargement soit actuellement branché sur une vanne donnée). Il s'agit là à la fois des connaissances indiquées par les concepteurs lors de l'initialisation de l'environnement, de sorte que les objets du modèle du domaine correspondent à ce qui est présent dans le modèle géométrique d'un environnement virtuel particulier, et des connaissances générées de manière dynamique suite à l'évolution de la simulation et des interactions des utilisateurs et des personnages virtuels. Ces connaissances sont représentées dans le modèle par des instances liées aux concepts qui correspondent aux trois types de connaissances décrits précédemment.
Les formalismes ontologiques étant peu adaptés pour représenter les aspects dynamiques des systèmes (temporalité, causalité, conditionnalité, etc.), la dynamique des comportements est spécifiée par un ensemble de règles d'évolution associé à l'ontologie.

5.1.2 Description du langage

DOMAIN-DL permet d'écrire les modèles du domaine comme des ontologies. Il est basé pour cela sur le langage **OWL-DL** [Bechhofer et al., 2004], un standard W3C qui permet ainsi la réutilisation d'ontologies existantes, contrairement au langage MOSS utilisé pour COLOMBO [Edward, 2011] qui est moins répandu. Les règles sont représentées avec la syntaxe **Jena** [McBride, 2002].
Ces ontologies reposent sur le méta-modèle présenté dans la figure 5.1.

Un modèle du domaine est ainsi constitué d'un ensemble d'**entités** (`Entity`) liées entre elles. Ces entités correspondent aux objets et aux concepts qui seront manipulables en regard du fonctionnement du monde. Nous détaillerons ici les entités suivantes : **objets** (`Object`), **agents** (`Agent`), **états** (`State`), **composants** (`Component`), **actions** (`Action`), **comportements** (`Behaviour`) et **événements** (`Event`).

Objets

Un **objet** (`Object`) est une entité, concrète ou abstraite, ayant une existence propre, c'est-à-dire pouvant être décrite ou manipulée sans qu'il soit nécessaire de connaître d'autres objets. Les **objets concrets** (`ConcreteObject`) ont une existence concrète dans le monde réel, mais ils ne sont pas nécessairement associés à une représentation géométrique dans le monde virtuel. Par exemple, un ressort se trouvant à l'intérieur d'une autre pièce ou un nuage de gaz incolore constituent des objets concrets. Un **objet géométrique** est un objet possédant une représentation géométrique dans l'environnement virtuel. Les **objets abstraits** (`AbstractObject`) n'ont au contraire pas d'existence concrète dans le monde réel. Il peut s'agir par exemple d'un processus de production, qui sera associé à des notions de coût et de délais.

Agents

Un **agent** (`Agent`) est un objet concret capable d'interagir de manière autonome avec son environnement, c'est à dire à la fois de le percevoir et d'effectuer des actions dessus. Un agent peut

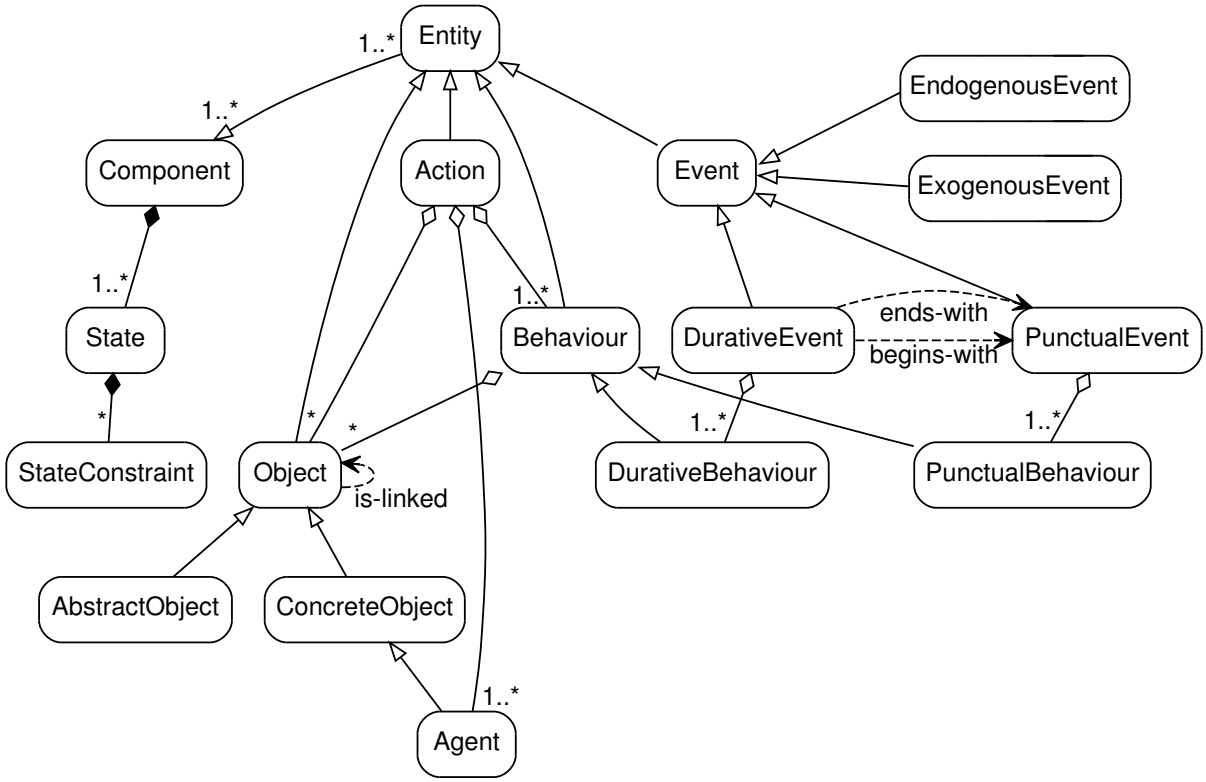


FIGURE 5.1 – Méta-modèle ontologique de DOMAIN-DL

être un utilisateur humain ou un personnage virtuel autonome. Dans ces deux cas, il s'agira d'un objet géométrique, c'est à dire que l'agent aura une représentation dans le monde virtuel (dans le cas de l'utilisateur, il s'agira de son *avatar*). Les agents ne sont cependant pas limités aux objets géométriques. Un formateur pouvant déclencher des événements dans l'environnement correspondra ainsi également à un agent, pour lequel seront définies des actions particulières. Il s'agira dans ce cas d'un objet non-géométrique, puisque l'agent n'aura pas de position dans l'environnement virtuel.

Etats

Un **état** (`State`) est une condition instantanée dans laquelle se trouve un objet ou un système. Au niveau du modèle, un état est représenté comme un ensemble de couples *(attribut, valeur)* liés sémantiquement. Il peut s'agir par exemple de la valeur de l'angle d'ouverture d'une porte, et d'un booléen définissant si celle-ci est considérée comme "ouverte" ou "fermée".
Les états sont liés à des **contraintes** (`StateConstraint`) permettant de donner les valeurs possibles pour les différents attributs. Dans le cas précédent, par exemple, on pourra spécifier que la valeur de l'angle est un nombre décimal compris entre 0 et 90 degrés.
Ils peuvent également être associés à des **règles de cohérence** (`ConstraintRules`) qui permettent de vérifier que les valeurs des attributs respectent les contraintes qui ont été définies, ainsi que de s'assurer de la cohérence interne des valeurs des attributs au sein d'un même état. Un exemple de règle de cohérence est présenté dans le tableau 5.1.

Composants

Un **composant** (`Component`) décrit une facette d'une entité. Il peut être lié à un ensemble d'états. Un composant "Cassable" serait par exemple lié à un état "Cassure" associé à une valeur booléenne.

```
[(?x :angle ?y)
GreaterThan(?y, 45)
->
SetValue(?x, :open, domain:True)]
```

TABLE 5.1 – Exemple de règle de contrainte liant les valeurs des attributs d'un même état

Les composants permettent de regrouper les entités ayant des propriétés communes signifiantes par rapport au domaine considéré.
La notion de composant correspond en fait à une vue particulière sur l'entité ; les relations entre une entité et un composant relèvent donc de la subsomption. Il est ainsi possible pour une entité de relever de plusieurs composants — une porte pourra être à la fois "Ouvrable" et "Cassable" —, comme pour un composant d'inclure plusieurs entités — une vanne sera également "Ouvrable". La figure 5.2 présente un extrait de modèle de domaine contenant plusieurs composants.

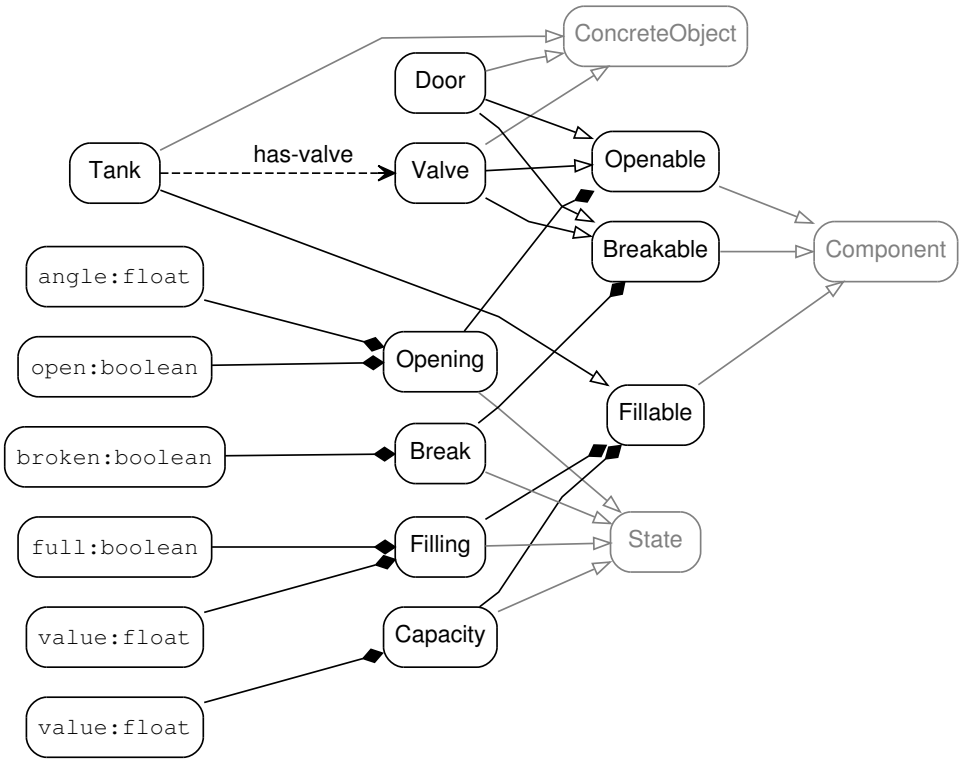


FIGURE 5.2 – Exemple de composants en DOMAIN-DL

Actions

Une **action** (`Action`) est un processus physique ou cognitif, pouvant être mis en œuvre par un agent à l'issue d'une décision. Une action peut être ponctuelle ou continue dans le temps.
Une action est liée à un (ou plusieurs) agent(s), ainsi qu'à un ensemble de paramètres éventuels. Ceux-ci sont modélisés sous la forme de composants, liés à l'action par des sous-relations de la relation `action-target`. Ainsi, l'action `Open` pourra par exemple être réalisée sur tous les objets qui instancient des sous-concepts du composant `Openable`.
Le concept d'action lui-même est un sous-concept des composants `Activable` et `Timestampable`, c'est à dire qu'une action peut être active ou non à un instant t, et qu'elle peut être tracée.
Les actions sont liées à des règles d'action, représentées en Jena, qui indiquent si l'action est physique-

ment réalisable ou non. Cela ne signifie cependant pas que l'action sera réussie, au sens où l'utilisateur ou le personnage virtuel pourrait se la représenter. Par exemple, l'action "Tourner" sur une poignée de porte sera réalisable même si la porte est verrouillée, cependant elle n'aura pas pour effet d'ouvrir la porte dans ce cas là. Les effets d'une action sont en effet gérés par l'activation d'un ou plusieurs comportements.

Comportements

Un **comportement** (`Behaviour`) correspond à une manière d'évoluer de l'objet ou du système qui en est doté. Ces comportements sont représentés par des concepts, liés à des règles de comportement écrites dans le formalisme Jena. Ces règles expriment des conditions et des effets sur les objets cibles des comportements (dans le cas d'un comportement déclenché par une action, il s'agit des objets cibles de l'action qui l'a déclenché), mais également sur les objets liés dynamiquement à ces objets cibles.
La figure 5.3 et la règle donnée dans le tableau 5.2 présentent un exemple d'un tel comportement : réaliser l'action "Tourner" sur une poignée liée à un bras de chargement a pour effet non seulement de faire effectuer une rotation à la poignée (`TurnBehaviour`), mais également de déclencher un second comportement `LockBehaviour` qui verrouille le bras lié à la poignée (relation statique créée lors de l'instanciation du modèle du domaine) et la vanne sur laquelle est branchée ce bras (relation dynamique créée au cours de la simulation).

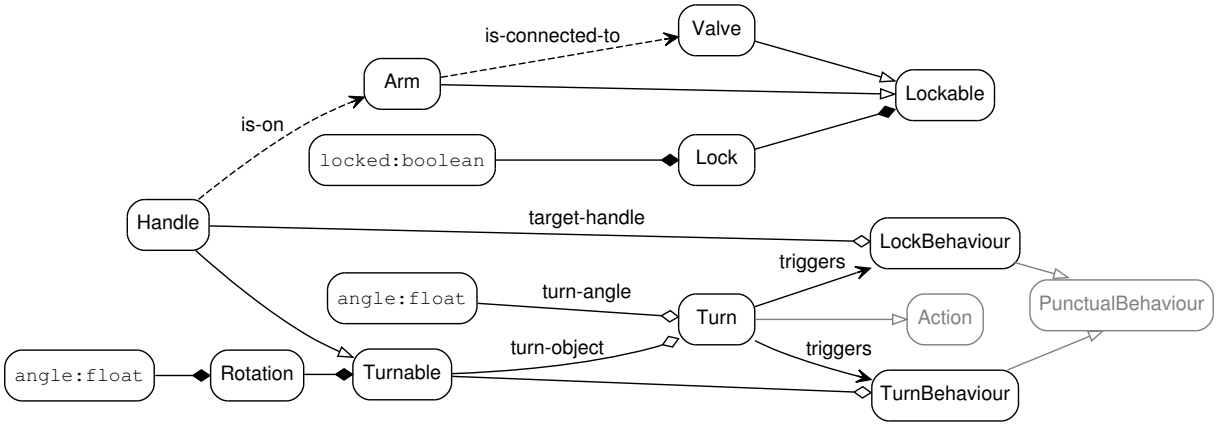


FIGURE 5.3 – Exemple de comportement en DOMAIN-DL

Un comportement peut être **ponctuel** (`PunctualBehaviour`) ou continu dans le temps. On parlera alors de comportement **duratif** (`DurativeBehaviour`). Les comportements ne sont pas nécessairement liés à des actions, et peuvent être actifs en fonction de certains états — par exemple, le remplissage d'un récipient lorsqu'un robinet est ouvert — ou bien être déclenchés par certains événements.

Evénements

Un **événement** (`Event`) est un fait marquant, qui survient à un moment donné. Un événement peut correspondre à une agrégation de changements d'états liés de manière causale ou au maintien des valeurs d'un ensemble d'états sur une certaine durée, dès lors qu'ils sont signifiants du point de vue de l'observateur. On parlera de **Situation** pour désigner un ensemble de couples ⟨état, valeur⟩ ou de relations entre objets. Un événement correspond donc au passage d'une situation à une autre, ou au maintien d'une même situation.

```
[LockBehaviour:
(?beh rdf:type :LockBehaviour)
(?beh domain:has-active-state ?state)
(?state domain:active domain:True)
(?beh domain:target-handle ?handle)
(?handle rdf:type :Handle)
(?handle :is-on ?arm)
(?arm :is-connected-to ?valve)
(?valve :has-lock-state ?lockstateV)
(?lockstateV :locked domain:False)
(?arm :has-lock-state ?lockstateA)
->
SetValue(?state, domain:active, sys:False)
SetValue(?lockstateV, :locked, domain:True)
SetValue(?lockstateA, :locked, domain:True)]
```

TABLE 5.2 – Règle gérant la dynamique du comportement `LockBehaviour`

Un événement peut être ponctuel ou duratif. Un **événement ponctuel** (`PunctualEvent`) correspond au changement instantané d'un état ou d'un ensemble d'états. Un **événement duratif** (`Durative Event`) correspond au changement continu d'un état ou ensemble d'états, ou au maintien d'un état ou ensemble d'états signifiant sur une certaine durée. Les événements duratifs sont délimités par des événements ponctuels indiquant leur début et leur fin. Un événement peut avoir une valeur indicative, ou bien déclencher un ou plusieurs comportements, à la manière d'une action.
Un événement peut être endogène ou exogène. Un **événement endogène** est la résultante de comportements du domaine, que ceux-ci relèvent d'actions de l'utilisateur, d'actions des personnages virtuels ou de comportements propres au système lui-même. Un **événement exogène** est déclenché de manière externe au système modélisé. Les événements considérés comme endogènes ou exogènes sont donc fortement dépendants de la couverture du modèle : par exemple, la sonnerie d'un téléphone dans l'environnement virtuel pourra, selon que l'appel provient d'un agent présent dans l'environnement ou non, être considérée respectivement comme un événement endogène ou exogène.
Les événements endogènes représentés dans le modèle ontologique sont associés à des règles, qui permettent de déterminer s'ils sont actifs ou non. Un exemple d'événement est présenté dans la figure 5.4 et les tableaux 5.3, 5.4 et 5.5.

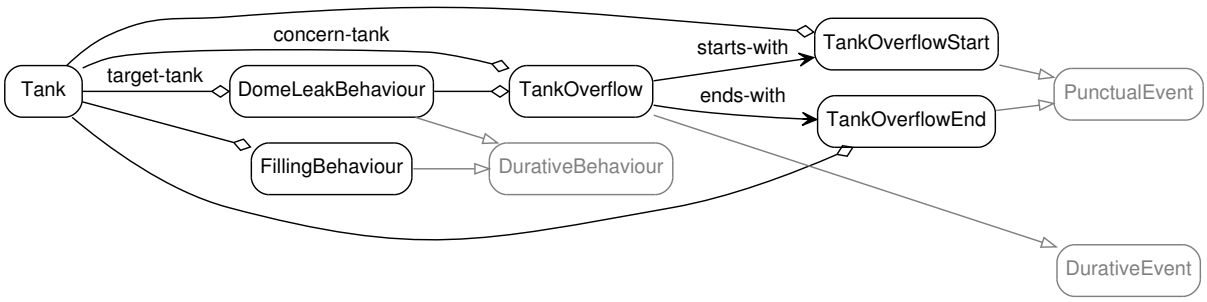


FIGURE 5.4 – Exemple d'événement en DOMAIN-DL

5.1.3 Bilan

Nous avons proposé le langage **DOMAIN-DL**, qui utilise une représentation ontologique et un ensemble de règles de comportement pour permettre l'écriture de modèles du domaine visant à être

```
[TankOverflowActivation:
(?tank :has-fillable-state ?fillstate)
(?fillstate :capacity ?capacity)
(?fillstate :filling ?quantity)
Greater(?quantity, ?capacity)
noValue((?x rdf:type :TankOverflow),
 (?x :concerns-tank ?tank),
 (?x domain:has-active-state ?state),
 (?state domain:active domain:True))
->
CreateEvent(:TankOverflowStart, ?startev)
SetValue(?startev, :concern-tank, ?tank)
CreateEvent(:TankOverflow, ?ev)
(?ev domain:has-active-state ?evstate)
SetValue(?evstate, domain:active, domain:True)
SetValue(?ev, :concern-tank, ?tank)
CreateBehaviour(:DomeLeakBehaviour, ?beh)
(?beh domain:has-active-state ?behstate)
SetValue(?behstate, domain:active, domain:True)
SetValue(?beh, :target-tank, ?tank)]
```

TABLE 5.3 – Règle permettant de détecter le début de l'événement `TankOverflow`

```
[TankOverflowDeactivation:
(?beh rdf:type :DomeLeakBehaviour)
(?beh :targets-tank ?tank)
(?beh domain:has-active-state ?behstate)
(?behstate domain:active domain:True)
(?ev rdf:type :TankOverflow)
(?ev :concerns-tank ?tank)
(?ev domain:has-active-state ?evstate)
(?evstate domain:active domain:True)
(?tank :has-fillable-state ?fillstate)
(?fillstate :capacity ?capacity)
(?fillstate :filling ?quantity)
Greater(?capacity, ?quantity)
->
SetValue(?evstate, domain:active, domain:False)
(?behstate domain:active domain:False)
CreateEvent(:TankOverflowEnd,?endev)
SetValue(?endev, :concern-tank, ?tank)]
```

TABLE 5.4 – Règle permettant de détecter la fin de l'événement `TankOverflow`

opérationnalisés dans des environnements virtuels.
La **modularité** de ce langage est assurée par l'adoption d'une posture ontologique, qui permet l'intégration de modèles pré-existants. La possibilité de définir des liens de subsomption entre les entités, de décrire les objets en fonction de leurs composantes et de découper des comportements complexes en comportements unitaires permet de modéliser le domaine à différents niveaux, et donc d'éviter au maximum d'introduire de la redondance dans le modèle lorsqu'il s'agit d'ajouter de nouveaux types d'objets ou des variations de comportements.

```
[DomeLeakBehaviour:
(?beh rdf:type :DomeLeakBehaviour)
(?beh domain:has-active-state ?state)
(?state domain:active domain:True)
(?beh :target-tank ?tank)
(?tank :has-fillable-state ?fillstate)
(?fillstate :capacity ?capacity)
->
Difference(?capacity, 1, ?capacity)]
```

TABLE 5.5 – Règle gérant la dynamique du comportement `DomeLeakBehaviour`

DOMAIN-DL permet d'expliciter les concepts du domaine et les liens entre ces concepts, et de les représenter de manière graphique et non uniquement sous forme de code informatique, ce qui assure l'**intelligibilité** du langage. Cette explicitation permet d'assurer l'inter-compréhension entre les différents acteurs impliqués dans la conception de l'environnement virtuel.
L'utilisation de formalismes informatiques — OWL-DL [Bechhofer et al., 2004] et Jena [McBride, 2002] — assure l'**interprétabilité** des modèles, qui peuvent être directement utilisés par le moteur de simulation du monde. D'autre part, le modèle est inspectable, de par la représentation des comportements sous forme de règles explicites, et la possibilité d'effectuer des requêtes et des inférences sur la base de connaissances ontologique.
DOMAIN-DL est de plus doté d'une grande **expressivité**. Les règles de comportements et les primitives issues de Jena [McBride, 2002], permettent de représenter des comportements complexes, qu'ils soient ponctuels ou continus, notamment via la prise en compte des liens dynamiques entre les objets. Ces comportements étant séparés des actions, il est possible de modéliser des comportements intrinsèques du système, et non plus seulement les effets directs des actions, comme c'est le cas avec de nombreux langages. La distinction des préconditions d'une action des conditions nécessaires au déclenchement des comportements qui y sont liés permet d'avoir des vues différentes sur une même action, et de considérer son "succès" comme dépendant de la tâche pour laquelle elle est réalisée, et non plus comme inhérent à cette action. La représentation des événements comme un ensemble signifiant d'états permet d'exprimer n'importe quelle agrégation qui serait pertinente du point de vue de l'utilisateur. Le méta-modèle permettant de définir des relations de subsomption entre les entités, il est de plus possible de désigner ces événements et situations à plus ou moins haut niveau. Enfin, la description des entités par les composants qu'elles possèdent pourrait permettre d'utiliser les modèles de domaine pour du raisonnement par analogie, que ce soit au niveau des objets ou des actions.

5.2 ACTIVITY-DL – Représentation de l'activité

Associé à DOMAIN-DL, ACTIVITY-DL est le langage qui permet d'écrire les modèles de l'activité. Nous détaillons ici l'approche choisie, puis le langage lui-même.

5.2.1 Approche

Besoins

De même que pour le langage de modélisation du domaine, nous recherchons pour le langage de modélisation de l'activité les quatres propriétés identifiées précédemment : modularité, intelligibilité, interprétabilité et expressivité.
Ici encore, la **modularité** est nécessaire pour l'ajout, la suppression ou la modification de tâches, de propriétés de ces tâches (préconditions, objets cibles, etc.) ou encore de leur ordonnancement.

En particulier, l'adaptation d'un environnement virtuel existant pour une application donnée (par exemple une formation visant un site ou une procédure spécifique) nécessite la spécifiation de portions d'activité propres à cette activité en plus ou à partir des modèles d'activité existants.
Cette modélisation étant effectuée par des experts, notamment en ergonomie du travail, le langage doit lui-aussi être **intelligible** et accessible à des non-informaticiens.
Afin de pouvoir être utilisé par les différents modules informatiques comme support de génération ou de prédiction de l'activité des personnages virtuels ou des utilisateurs, le langage doit être **interprétable**. En particulier, il est nécessaire qu'il soit construit autour du modèle du domaine afin de pouvoir lier les tâches avec les actions, objets et états qu'elles référencent.
Enfin, il est important que le langage soit suffisamment **expressif** pour permettre la représentation de comportements humains, complexes et non-idéaux [1]. Il doit ainsi pouvoir représenter à la fois l'ordonnancement hiérarchique et logique des tâches — qui sera assimilé aux motivations des personnages virtuels et de l'utilisateur et à la façon dont ils se représentent l'activité —, leur ordonnancement temporel ainsi que leur contexte de réalisation. Ces connaissances doivent être formalisées de sorte à permettre à la fois la génération d'activité (pour la génération des comportements des personnages virtuels autonomes), le suivi d'activité (pour le suivi et l'analyse des actions de l'utilisateur) et la prédiction d'activité (pour la génération de scénarios prédictifs).

Proposition

Nous avons choisi d'opter pour une **modélisation hiérarchique** basée sur la notion de **tâches**, qui opérationnalise des principes issus de langages utilisées en **ergonomie**.
Plutôt que le séquencement temporel ou logique de l'activité, ACTIVITY-DL vise à permettre la description de la manière dont un opérateur se représente sa tâche au niveau cognitif. Il s'agit donc d'une **analyse cognitive** plutôt que logique de la tâche, ce qui se traduit à la fois au niveau des propriétés du langage (par exemple, la modélisation des activités limites tolérées par l'usage) et au niveau des méthodes qui seront utilisées par les ergonomes pour collecter des données sur les représentations construites par les opérateurs.
ACTIVITY-DL étend en ceci les travaux réalisés précédemment sur **HAWAI-DL** [Amokrane et al., 2008] [Edward et al., 2008], qui ont été présentés dans la section 3.1.2. La formalisation proposée pour HAWAI-DL s'est en effet avérée limitée pour la prédiction des actions des personnages dans une optique de scénarisation.
Les principaux concepts du langage sont un ensemble de tâches/buts et de sous-tâches/sous-buts, qui traduisent le point de vue de l'opérateur sur son activité, les relations entre ces buts, le déroulement possible des actions en fonction du contexte et leurs conditions de satisfaction. Ce déroulement est spécifié de différentes manières. D'une part, par des constructeurs temporels qui expriment la **décomposition temporelle** des tâches mères, et par des constructeurs logiques qui expriment leur **décomposition logique**. D'autre part, par un ensemble de **conditions** qui déterminent le contexte de réalisation de la tâche et sont classifiées en regard de la manière dont elles sont considérées par l'opérateur, selon qu'il s'agit d'état du monde à atteindre afin de réaliser la tâche, ou à prendre en compte dans la décision de la réaliser ou non.

5.2.2 Description du langage

ACTIVITY-DL repose sur l'**organisation hiérarchique** d'un ensemble de **tâches**, dont les propriétés (actions, objets cibles, états pris en compte dans les conditions, etc.) se réfèrent à des entités décrites dans le modèle de domaine représenté en DOMAIN-DL. Les modèles décrits en ACTIVITY-DL sont représentés en XML, et associés à une représentation graphique sous forme d'arbre ; un ex-

1. Une réflexion sur la nécessité d'intégrer des aspects facteurs humains dans les modèles informatiques de représentation des connaissances est proposée dans [Lourdeaux, 2012].

emple est présenté dans la figure 5.5.

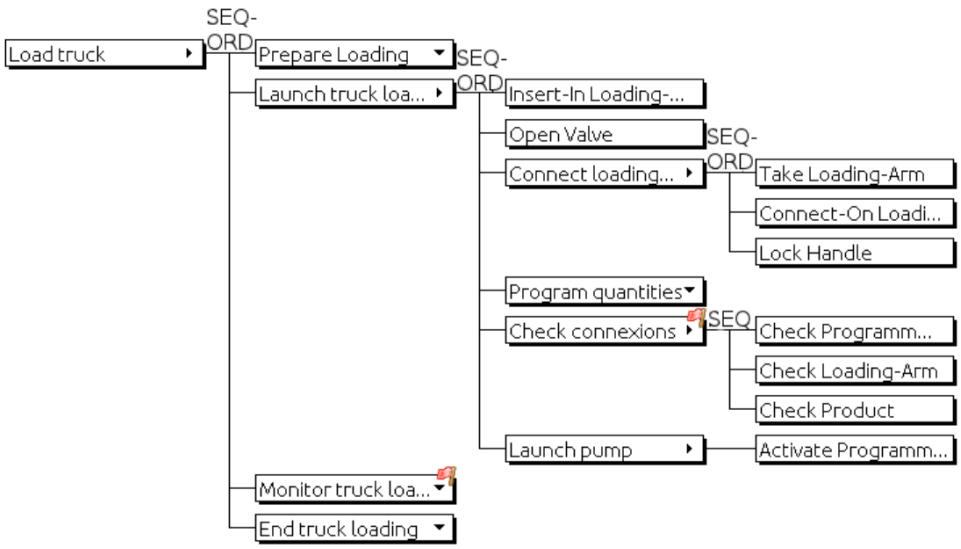


FIGURE 5.5 – Représentation d'un modèle en ACTIVITY-DL

Les modèles sont décrits selon le méta-modèle présenté dans la figure 5.6.

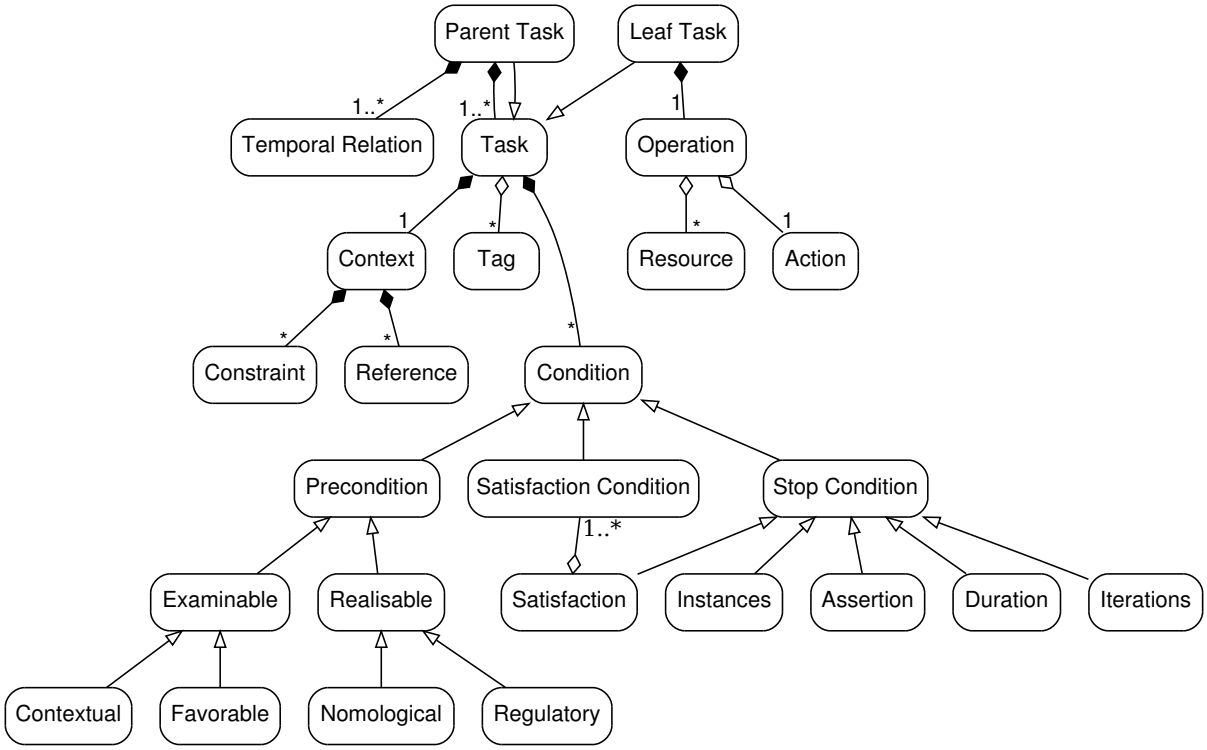


FIGURE 5.6 – Méta-modèle d'ACTIVITY-DL

Nous détaillerons ici les concepts suivants : **tâche** (`Task`), **opération** (`Operation`), **relation temporelle** (`Temporal Relation`), **condition** (`Condition`), **contexte** (`Context`) et **marqueur** (`Tag`).

Tâches

ACTIVITY-DL est basé sur un ensemble de tâches organisées hiérarchiquement, qui reflètent le point de vue de l'opérateur sur son activité (ce n'est donc pas une représentation de la procédure prescrite). Chaque tâche correspond à un but, qui peut être décomposé en sous-buts jusqu'à atteindre le niveau de granularité le plus bas qui est celui des actions unitaires. On parlera de tâches mères pour les tâches décomposables, et de tâches feuilles pour les tâches de plus bas niveau.
Les tâches mères reflètent l'organisation de l'activité selon trois aspects :

1. l'organisation hiérarchique se fait au travers de la décomposition en tâches filles ;
2. l'organisation temporelle se fait au travers des constructeurs ;
3. l'organisation logique se fait au travers des conditions de satisfaction.

Les tâches feuilles correspondent au plus bas niveau de granularité de la description. Elles sont directement associées à l'exécution d'une action, qui est représentée par une opération.

Opérations

Une **opération** (`Operation`) est composée d'une action, ainsi que d'éventuelles ressources additionnelles qui jouent le rôle des paramètres de l'action. Les ressources peuvent être soit des variables qui seront identifiées avec des instances d'objets du monde, soit des valeurs numériques. Les actions et les prédicats qui relient une action à ses ressources sont extraits du modèle du monde. Un exemple de tâche fille comportant une opération est présenté dans le tableau 5.6 ; cette opération reprend les concepts du modèle du domaine présenté dans l'exemple de la figure 5.3.

Task		`Turn the Handle`
Operation	**Action**	`Turn`
	Ressources	`turn-object: ?handle` `turn-angle: 90`

TABLE 5.6 – Exemple de tâche contenant une opération

Relations temporelles

Les relations temporelles entre les tâches filles d'une même tâche mère sont représentées à l'aide de relations d'Allen [Allen, 1983], rappelées dans le tableau 5.7. Ces relations expriment les relations possibles entre les intervalles de temps correspondant à l'exécution des tâches. Un ensemble de relations possibles est associé à chaque couple de tâches. Si une tâche mère possède plus de deux tâches filles, la table de composition d'Allen est utilisée pour calculer les relations manquantes.

Afin de faciliter la saisie et d'améliorer la lisibilité du modèle, ACTIVITY-DL contient également un ensemble de constructeurs prédéfinis, qui correspondent à des agrégats de relations d'Allen couramment utilisés. Ces constructeurs sont partiellement inspirés de ceux présents dans des langages de modélisation de l'activité MAD* [Sebillotte and Scapin, 1994] et GTA [Van Der Veer et al., 1996]. Par exemple, l'association des deux relations $\{<, m\}$ qui correspond à l'exécution séquentielle de deux tâches dans un ordre donné (soit directement à la suite, soit avec un délai entre les deux), peut être exprimée à l'aide du constructeur `SEQ-ORD`. Les différents constructeurs sont présentés dans le tableau 5.8.

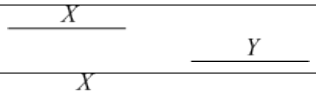
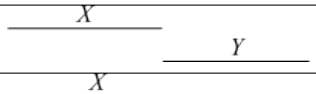
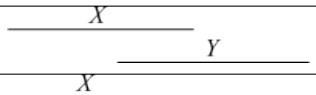
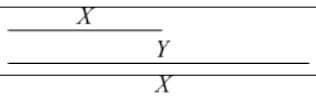
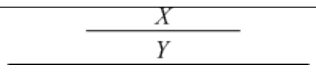
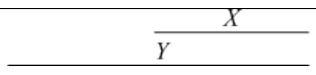
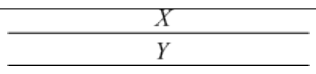
Relations	Illustration	Description
$X < Y, Y > X$		X a lieu avant Y
$XmY, YmiX$		Y commence quand X finit
$XoY, YoiX$		X et Y se superposent
$XsY, YsiX$		X a lieu au début de Y
$XdY, YdiX$		X a lieu pendant Y
$XfY, YfiX$		X a lieu à la fin de Y
$X = Y$		X et Y ont lieu en même temps

TABLE 5.7 – Relations temporelles d'Allen [Allen, 1983]

Constructeur	Intitulé	Relations	Description
IND	Indépendant	$A\{<,>,m,mi,o,oi,s,si,d,di,f,fi,=\}B$	Tâches ne partageant ni ressources, ni cibles, et dont les exécutions n'influent pas les unes sur les autres. Elles n'ont donc aucune contrainte spécifique d'ordonnancement.
SEQ	Séquentiel	$A\{<,>,m,mi\}B$	Tâches n'ayant pas d'intervalle de temps commun, s'exécutant les unes à la suite des autres, sans ordre spécifique.
SEQ-ORD	Ordonné	$A\{<,m\}B$	Tâches n'ayant pas d'intervalle de temps commun, s'exécutant les unes à la suite des autres dans un ordre déterminé.
PAR	Parallèle	$A\{o,oi,s,si,d,di,f,fi,=\}B$	Tâches étant exécutées avec au moins un intervalle de temps commun.
PAR-SIM	Simultané	$A\{=\}B$	Tâches partageant le même intervalle de temps : ayant la même durée et débutant et finissant en même temps.
PAR-START	Début	$A\{s,si,=\}B$	Tâches ayant un intervalle de temps commun et débutant en même temps.
PAR-END	Fin	$A\{f,fi,=\}B$	Tâches ayant un intervalle de temps commun et finissant en même temps.

TABLE 5.8 – Constructeurs définis dans ACTIVITY-DL

Conditions

Les tâches peuvent posséder un ensemble de conditions, qui vont spécifier le contexte de leur exécution. Ces conditions peuvent être de trois types :

1. préconditions ;
2. conditions de satisfaction ;
3. conditions d'arrêt.

Les préconditions et conditions de satisfaction sont exprimées à l'aide d'Assertions, associées à des opérateurs logiques : `AND`, `OR`, `XOR` et `NOT`.

Assertions Les **assertions** (`Assertion`) permettent d'exprimer ou de tester un élément de connaissance sur le monde, et peuvent être évaluées à vrai ou faux. Elles sont constituées d'un ensemble de triplets *(sujet prédicat objet)*. On distingue deux types d'assertions : les assertions sur le domaine, qui ont trait à l'état du monde (par exemple : `(?valve01 :is-open :False)`), et les assertions sur l'activité, qui ont trait à l'état de l'activité (par exemple : `(Open-Valve :is-done :True)`). Plutôt que de référencer directement des instances d'objets, les assertions utilisent des variables typées — reconnaissables ici à leurs identifiants qui commencent par `?`.
En plus des prédicats du modèle du domaine, les assertions peuvent contenir des opérateurs plus complexes, appelés **primitives**. Les primitives prennent un ou plusieurs arguments, et sont utilisées soit pour calculer une valeur numérique (par exemple `product(?a, ?b, ?c)`) soit pour obtenir une valeur booléenne suite à un test (par exemple `lessOrEqual(?x, ?y)`). Les primitives considérées correspondent aux primitives de base de Jena [McBride, 2002]. Un exemple d'assertion contenant des primitives est présenté dans le tableau 5.9.

```
(?tank :has-content-value ?x), (?tank :has-programmed-quantity-value ?y),
(?tank :has-max-capacity-value ?z), sum(?x, ?y, ?w), greaterThan(?z, ?w)
```

TABLE 5.9 – Exemple d'assertion contenant des primitives

Préconditions Le contexte dans lequel une tâche peut ou doit être réalisée est exprimé par les **préconditions** (`Precondition`). Celles-ci sont classées selon deux catégories, selon le traitement qui en est fait par rapport à la tâche : les **préconditions à réaliser** (`Realisable Precondition`), et les **préconditions à examiner** (`Examinable Precondition`). Le tableau 5.10 détaille les différentes catégories de préconditions.

Préconditions à réaliser Pour qu'une tâche donnée puisse être exécutée, ses préconditions à réaliser doivent être vraies. Dans le cas contraire, elles devront être rendues vraies avant d'exécuter la tâche. Par exemple, si la tâche "Ouvrir la porte" possède une précondition à réaliser "La porte est déverrouillée", et que le monde contient l'état "La porte est verrouillée", alors exécuter la tâche demandera au préalable l'exécution de l'action "Déverrouiller" afin de rendre la précondition vraie. Il existe deux types de préconditions à réaliser : nomologiques et règlementaires.
Les **préconditions nomologiques** (`Nomological Precondition`) sont liées aux contraintes physiques du monde [2] : si on cherche à exécuter la tâche alors qu'une précondition nomologique est fausse, la tâche échouera. Elles incluent notamment les préconditions liées aux actions, qui sont directement importées depuis le modèle du domaine. Par exemple, la tâche feuille "Allumer la perceuse" contient une opération qui se rapporte à l'action "Appuyer" et à l'objet "Bouton". Les règles de l'action "Appuyer" ont pour précondition "Bouton non enfoncé", ainsi cette précondition va être ajoutée comme précondition nomologique de la tâche. Les préconditions nomologiques peuvent également être des conditions nécessaires pour que les effets désirés de la tâche soient atteints. Par exemple, "Allumer la perceuse" aura également pour précondition "La perceuse est branchée au circuit électrique". Elles correspondront dans ce cas aux préconditions des comportements déclenchés par l'action dans le modèle du domaine, et qui seront considérés comme pertinents par rapport à la tâche.
Les **préconditions réglementaires** (`Regulatory Precondition`) se rapportent aux bonnes pratiques, et expriment les conditions nécessaires pour que la tâche se déroule dans le respect des procédures. Elles peuvent concerner des domaines tels que la sécurité, l'hygiène ou la posture professionnelle. Pour réaliser une tâche donnée, il est nécessaire de s'assurer que ses préconditions réglementaires

2. Le terme nomologique est issu du domaine de la psychologie cognitive, d'après la définition proposée par [Goel, 1992] : "Nomological constraints are constraints dictated by natural law".

Catégorie	Précondition	Description
Conditions à réaliser	Nomologique	Les préconditions nomologiques correspondent à l'état dans lequel le monde doit se trouver pour que la tâche soit physiquement réalisable. Elles incluent les préconditions des règles d'exécution de l'action liée à la tâche, définies dans le modèle du domaine.
		Exemple : Pour réaliser la tâche "Ouvrir la porte", il faut que la porte soit fermée.
	Règlementaire	Les préconditions règlementaires correspondent aux états du monde nécessaires à la bonne réalisation de la tâche selon la procédure.
		Exemple : Pour réaliser la tâche "Débrancher le tuyau", il faut porter ses gants de protection.
Conditions à examiner	Contextuelle	Les préconditions contextuelles correspondent à l'état du monde dans lequel il est pertinent de réaliser la tâche. Lorsque ces conditions ne sont pas vérifiées, la tâche peut être ignorée.
		Exemple : *La réalisation de la tâche "Nettoyer le tuyau" n'est pertinente que si le tuyau est sale.*
	Favorable	Les préconditions favorables correspondent aux états du monde dans lesquelles la réalisation de la tâche sera privilégiée par rapport à d'autres tâches. Elles permettent de choisir entre plusieurs tâches lorsqu'il existe plusieurs alternatives permettant de réaliser une même tâche mère.
		Exemple : *Si le boulon est rouillé, la réalisation de la tâche "Dégripper le boulon" est à privilégier.*

TABLE 5.10 – Préconditions définies dans ACTIVITY-DL

soient respectées. Autrement, la réalisation de la tâche serait possible, mais ne respecterait pas la procédure. Par exemple, la tâche “Débrancher le tuyau” nécessite que la condition “L'agent porte ses gants de protection” soit vraie. Si l'agent ne porte pas ses gants lorsqu'il souhaite débrancher le tuyau, il devrait aller les chercher.

Préconditions à examiner Les préconditions à examiner spécifient les états du monde qui sont significatifs par rapport à la tâche, mais qui ne nécessitent pas spécifiquement d'actions dans le but de les atteindre. Ces conditions sont examinées lors de la décision de réaliser ou non la tâche. Par exemple, la tâche “Utiliser un extincteur” a une précondition à examiner “Un objet est en feu”. Cependant, si cette condition est fausse, il n'est pas pertinent de mettre le feu à un objet dans le but de la rendre vraie.
Les **préconditions contextuelles** (`Contextual Precondition`) décrivent les états du monde dans lesquels la tâche est considérée comme pertinente. Si ces conditions sont fausses, la réalisation de la tâche n'a pas de sens, et peut donc être ignorée. Par exemple, la tâche “Nettoyer le tuyau” a pour précondition contextuelle “Le tuyau est sale”. Ainsi, si cette condition est vraie, la tâche pourrait être réalisée, mais dans le cas contraire elle n'a pas d'intérêt.
Les **préconditions favorables** (`Favorable Precondition`) sont utilisées pour mettre des priorités ou choisir entre plusieurs tâches dans le cas d'une alternative. Elles représentent les conditions dans lesquelles la réalisation d'une tâche donnée serait privilégiée. Par exemple, la tâche “Alerter l'opérateur” se décompose en deux sous-tâches alternatives : “Appeler l'opérateur par l'interphone” et “Se déplacer à la station de travail de l'opérateur”. Etant donné que la station de travail de l'opérateur est située dans un autre bâtiment que celui du manager, la tâche “Appeler l'opérateur par l'interphone”

aura comme précondition favorable “Il pleut”. Quand ses conditions favorables sont vraies, une tâche aura ainsi une désirabilité plus haute que ses tâches sœurs. Cependant, quand une condition favorable est fausse, la tâche pourrait toujours être le meilleur choix, en fonction des conditions favorables des autres tâches. Dans cet exemple, l’état “Il ne pleut pas” ne signifie pas que l’on choisirait forcément la tâche “Se déplacer à la station de travail de l’opérateur”.

Conditions de satisfaction Les conditions de satisfaction (`Satisfaction Condition`) expriment le contexte dans lequel la tâche sera considérée comme réussie. Elles peuvent contenir des assertions sur le domaine (par exemple, la tâche “Ouvrir la porte” sera réussie si l’état “Porte ouverte” est atteint), ou bien sur l’activité (par exemple, la tâche “Mettre les Equipements de Protection Individuels” sera réussie lorsque ses sous-tâches “Mettre les lunettes” ET “Mettre le casque” ET “Mettre les gants” seront réussies).

Conditions d’arrêt Les conditions d’arrêt (`Stop Conditions`) précisent le contexte dans lequel l’exécution de la tâche devra être stoppée, que celle-ci soit réussie ou échouée. Il existe différents types de préconditions, présentés dans le tableau 5.11. Des conditions d’arrêt de différents types peuvent être associées à une même tâche, liées entre elles à l’aide d’opérateurs logiques.

Condition d’arrêt	**Description**
satisfaction	La tâche est considérée comme réalisée lorsque ses conditions de satisfaction sont vérifiées.
	Exemple : La tâche "Utiliser un extincteur" doit être arrêtée une fois que la conséquence "Feu éteint" est atteinte.
assertions	La tâche est considérée comme réalisée lorsqu’un certain état du monde est atteint.
	Exemple : La tâche "Utiliser un extincteur" doit être arrêtée lorsque l’état du monde "Extincteur vide" est atteint.
durée	La tâche est considérée comme réalisée après une certaine durée.
	Exemple : Pour refroidir une brûlure, la tâche "Placer la brûlure sous l’eau froide" doit être exécutée pendant 10 minutes.
itérations	La tâche est considérée comme réalisée au bout d’un certain nombre d’itérations.
	Exemple : Pour déverrouiller la porte, il faut répêter deux fois la tâche "Faire un tour de clé".
instances	La tâche est considérée comme réalisée après avoir itéré sur un certain ensemble d’instances.
	Exemple : La tâche "Vérifier les pneus" doit être réalisée sur l’ensemble des pneus du camion.

TABLE 5.11 – Conditions d’arrêt définies dans ACTIVITY-DL

Contexte

Les ressources associées aux tâches sont représentées comme des variables typées, afin d’être en mesure de transposer le même arbre d’activité sur différentes instances des objets du monde, ce qui ne serait pas possible en référençant directement les instances d’objets dans l’activité. Dès lors, un mécanisme est nécessaire pour lier les variables d’une tâche à l’autre afin de spécifier qu’elles se

réfèrent à la même instance. Par exemple, il est nécessaire de préciser que les tâches sœurs “Amener le bras de chargement”, qui possède une ressource `?x` de type “Bras de chargement”, et “Brancher le bras de chargement sur la vanne”, qui possède deux ressources `?x` de type “Bras de chargement” et `?y` de type “Vanne”, font référence à la même instance de bras de chargement.
Pour ce faire, ACTIVITY-DL utilise la notion de **contexte de tâche** (`Context`), qui déclare les variables qui sont passées d'une tâche mère à ses tâches filles. Le contexte de la tâche contient des références, chacune composée d'un identifiant de variable et d'un type. Le contexte d'une tâche contient toutes les références déclarées dans le contexte de sa tâche parente. Pour les tâches feuilles, les identifiants des variables peuvent être directement utilisés comme ressources pour une opération. Un exemple de tâche mère partageant un contexte avec ses tâches filles est présenté dans le tableau 5.12.

(a) Tâche mère

Task		`Connect the Loading Arm and the Valve`
Subtasks		`Bring the Loading Arm` `Plug the Loading Arm on the Valve`
Constructor		`SEQ-ORD`
Context	**References**	`(?x rdf:type :LoadingArm)` `(?y rdf:type :Valve)`

(b) Première tâche fille

Task		`Bring the Loading Arm`
Operation	**Action**	`Bring`
	Ressources	`brought-object: ?x`
Context	**References**	`(?x rdf:type :LoadingArm)` `(?y rdf:type :Valve)`

(c) Seconde tâche fille

Task		`Plug the Loading Arm on the Valve`
Operation	**Action**	`Plug`
	Ressources	`male-object: ?x` `female-object: ?y`
Context	**References**	`(?x rdf:type :LoadingArm)` `(?y rdf:type :Valve)`

TABLE 5.12 – Exemple de tâche partageant un contexte avec ses sous-tâches

Le contexte de la tâche peut également contenir des **contraintes**, qui expriment les relations entre les différents objets du contexte. Ces contraintes sont modélisés par des assertions. Un exemple de contrainte est présenté dans le tableau 5.13.
Les variables qui sont utilisées au sein de la tâche, mais pas déclarées dans le contexte, sont traitées comme des **variables anonymes**, ce qui signifie qu'elles peuvent être identifiées avec n'importe quelle instance. Ceci est particulièrement utile dans les conditions, pour exprimer l'existence (*“il existe au moins un X qui . . . ”*) et la négation (*“il n'existe pas de X qui . . . ”*). Un exemple d'utilisation de variables anonymes est présenté dans le tableau 5.14.

Marqueurs

ACTIVITY-DL utilise un système de **marqueurs** (`Tag`) génériques pour identifier, catégoriser, préciser ou mettre en avant certains éléments du modèle d'activité. Une tâche peut posséder un ensemble de marqueurs. Il est également possible de placer des marqueurs sur les préconditions d'une tâche. Ces marqueurs peuvent être soit des marqueurs binaires, ou “flags” (par exemple “optionnel”)

(a) Tâche mère

Task		`Lock the Loading Arm`
Subtasks		`Turn the Handle`
Context	**References**	`(?x rdf:type :LoadingArm)`

(b) Tâche fille

Task		`Turn the Handle`
Operation	**Action**	`Turn`
	Ressources	`turn-object: ?y` `turn-angle: 90`
Context	**References**	`(?x rdf:type :LoadingArm)` `(?y rdf:type :Handle)`
	Constraints	`(?y :is-on ?x)`

TABLE 5.13 – Exemple de tâche utilisant des contraintes

Task		`Plug the Loading Arm on the Valve`
Operation	**Action**	`Plug`
	Ressources	`male-object: ?x` `female-object: ?y`
Context	**References**	`(?x rdf:type :LoadingArm)` `(?y rdf:type :Valve)`
Preconditions	**Nomological**	`(AND (NOT (?x :is-connected-to ?object1))` `(NOT (?object2 :is-connected-to ?y)))`

TABLE 5.14 – Exemple de tâche contenant des variables anonymes

soit des marqueurs valués qui sont associés à une valeur numérique ou textuelle (par exemple “priorité = 3”).
Il existe un ensemble de marqueurs prédéfinis, présentés dans le tableau 5.15, et des marqueurs spécifiques peuvent être déclarés pour les besoins d’une application donnée. Par exemple, pour une application liée à la formation à la maîtrise des risques, certaines tâches pourraient être marquées avec un marqueur “risque”, de façon à les mettre en exergue lors des retours et de l’évaluation de l’activité de l’utilisateur.

Activités Limites Tolérées par l’Usage (ALU) Un marqueur particulier défini dans ACTIVITY-DL est celui qui désigne les Activités Limites Tolérées par l’Usage (ALU). Les ALU sont des déviations de l’activité prescrite pouvant être tolérées dans les faits, sur certains sites et dans certaines conditions, dans le but d’atteindre des compromis. Il s’agit d’un concept dérivé de recherches en psychologie cognitive, qui reflète un compromis local et souvent informel entre des acteurs d’un certain domaine [Garza and Fadier, 2007]. Par exemple, il peut s’agir de ne pas effectuer une tâche de vérification afin d’atteindre un compromis entre productivité et sécurité. Les tâches qui constituent des ALU sont marquées comme telles dans le modèle grâce aux marqueurs spécifiques `ALU+` et `ALU-`.
On distingue en effet deux types d’ALU :

- **ALU+** : les tâches interdites, qu’il est tacitement toléré de réaliser sous certaines conditions ;
- **ALU-** : les tâches obligatoires, qu’il est tacitement toléré de ne pas réaliser sous certaines conditions.

Les ALU sont systématiquement associées à une ou plusieurs préconditions (en général de type contextuelles), marquées comme CLU (Conditions Limites tolérées par l’Usage). Les CLU correspondent aux conditions dans lesquelles la déviation peut être observée.

Marqueur	Description
itératif	Tâches devant être répétées plusieurs fois, jusqu'à ce que leurs conditions d'arrêt soient vérifiées.
optionnel	Tâches pouvant être ignorées si leurs conditions de satisfaction sont déjà vérifiées.
expert+	Tâches ou préconditions n'étant prises en compte que par les experts.
expert-	Tâches ou préconditions n'étant pas prises en compte par les experts.
ALU+	Tâches interdites, qu'il est toléré de réaliser quand leurs conditions CLU sont vérifiées.
ALU-	Tâches obligatoires, qu'il est toléré de ne pas réaliser quand leurs conditions CLU sont vérifiées.
CLU	Préconditions contextuelles qui déterminent la réalisation ou la non-réalisation des tâches ALU.

TABLE 5.15 – Marqueurs prédéfinis dans ACTIVITY-DL

5.2.3 Bilan

Nous avons proposé le langage **ACTIVITY-DL**, qui utilise une représentation hiérarchique des tâches ainsi qu'un ensemble de constructeurs et de conditions pour permettre l'écriture de modèles d'activité visant à être opérationnalisés dans des environnements virtuels.
La **modularité** du langage est assurée notamment par l'utilisation d'une structure hiérarchique, qui permet une certaine souplesse dans la modification du modèle, en particulier pour l'ajout d'alternatives pour la réalisation d'un même but. Le système de variables et de contraintes permet de désigner les cibles des actions sans devoir passer par des instances, et donc de ne pas avoir à modifier le modèle d'activité lorsque de nouveaux objets sont ajoutés à l'environnement virtuel. En particulier, la possibilité de spécifier des variables anonymes et des itérations sur des ensembles d'instances déterminés par des contraintes permet de conserver l'intégrité du modèle d'activité en cas de changements dans le modèle du domaine — par exemple, de passer facilement d'un modèle où les camions contiennent quatre cuves à un modèle où ils en contiennent six.
L'utilisation d'un formalisme inspiré de travaux en ergonomie, et donc proches des langages utilisés par les ergonomes qui devront remplir les modèles d'activité, participe à son **intelligibilité**. De plus, la représentation hiérarchique rend les modèles à la fois plus lisibles et plus simples à éditer, par rapport à une représentation par un ensemble d'opérateurs de planification ou de règles logiques.
Ce langage étant représenté de manière informatique, selon le formalisme XML, il est également **interprétable**. L'utilisation de relations d'Allen permet de déterminer précisément les contraintes temporelles de réalisation des tâches, en particulier pour la génération de comportements des personnages virtuels. Les tâches référencent directement les concepts du modèle du domaine, ce qui permet également d'effectuer du raisonnement sur les préconditions, conditions de satisfaction ou conditions d'arrêt des tâches.
Enfin, ACTIVITY-DL est particulièrement **expressif**, en cela qu'il permet de décrire la représentation que l'opérateur se fait de sa tâche, à la fois au niveau de ses motivations, du contexte dans lequel sont réalisées les activités, ainsi que de l'ordonnancement logique et temporel entre les tâches. En particulier, la classification des préconditions permet de séparer les conditions de l'action des conditions de la tâche, et d'expliciter les suppositions émises par les ergonomes lors de la création du modèle, qui se sont avérées problématiques lors des travaux précédents de l'équipe sur la modélisation en HAWAI-DL. En effet, lorsque les ergonomes rentrent une séquence de tâches dans un modèle, il était souvent implicitement convenu que les tâches précédentes constituaient des prérequis pour faire la tâche. De même, pour les tâches triviales, les ergonomes n'indiquaient pas les préconditions (par ex-

emple, qu'il n'est possible d'ouvrir une porte que si elle n'est pas déjà ouverte). ACTIVITY-DL permet de clarifier ces "non-dits" : les préconditions nomologiques sont modélisées avec le fonctionnement du système, soit au niveau de l'action, soit au niveau des comportements qui lui sont liés, et peuvent être directement intégrés aux préconditions lors de l'ajout d'une nouvelle tâche par l'ergonome.

Chapitre 6

DIRECTOR – Moteur de planification et réalisation de scénarios prédictifs

Sommaire

Le moteur DIRECTOR vise à guider dynamiquement le scénario d'une simulation interactive peuplée de personnages virtuels autonomes pour réaliser des objectifs scénaristiques donnés, en respectant la liberté d'action de l'utilisateur et la cohérence des comportements de la simulation, tout en favorisant l'adaptabilité de l'environnement virtuel.
DIRECTOR utilise des techniques de **planification** pour générer un scénario répondant aux objectifs scénaristiques. Cependant, plutôt que de considérer le scénario généré comme prescriptif, DIRECTOR traite ce dernier comme une représentation du **scénario souhaité**. Pour cela, il utilise les modèles de connaissances qui sous-tendent la simulation pour **prédire son évolution** — au niveau des systèmes

techniques comme des comportements des personnages virtuels autonomes et des utilisateurs. Pour modifier cette prédiction et influencer l'évolution de la simulation, DIRECTOR utilise un ensemble d'**ajustements**, qui permettent d'intervenir indirectement sur le déroulement des événements sans en perturber la cohérence.
Nous présentons tout d'abord les différents ajustements employés par DIRECTOR, puis le formalisme utilisé pour représenter les scénarios. Nous nous attardons ensuite sur la particularité de la génération de plans prédictifs, avant de présenter les processus de génération des opérateurs de prédiction, ainsi que le processus global de génération, d'exécution et de suivi du scénario dans l'environnement virtuel.

6.1 Ajustements

Il existe un certain nombre de moyens permettant d'influencer le déroulement des événements dans une simulation : modification des états des objets, déclenchement d'événements, envoi d'ordres aux personnages virtuels, paramétrisation de leurs états internes, etc.
Cependant, dans le cas où la simulation contient des personnages virtuels autonomes ou semi-autonomes, la plupart de ces moyens d'actions sont problématiques. Nous avons ainsi pu voir dans le chapitre 2 que le contrôle de personnages virtuels semi-autonomes par des ordres au niveau comportemental ou motivationnel mettait en péril la cohérence de leurs comportements. De même, les changements forcés d'états des objets (par exemple, un objet cassé redevenant intact) contrarient l'explicabilité du scénario global.
Il est donc nécessaire d'identifier un certain nombre de leviers permettant de modifier le cours du scénario sans pour autant faire obstacle à sa **cohérence**. Ces **ajustements** doivent à la fois favoriser la **validité écologique** de l'environnement virtuel en respectant l'intégrité des modèles de domaine et d'activité, et assurer l'**explicabilité** des comportements individuels et du scénario dans son ensemble.
Pour exercer un niveau de contrôle satisfaisant sur une simulation, nous estimons qu'il est nécessaire de pouvoir influencer à la fois l'issue des comportements des systèmes techniques simulés et les décisions prises par les personnages virtuels. Influencer les choix d'actions des personnages virtuels — sans pour autant leur donner d'ordres — peut se faire de deux manières : soit en agissant sur l'état du monde, et donc sur la représentation qu'en a le personnage — ce que [Blumberg and Galyean, 1997] nomment **contrôle environnemental**—, soit en agissant sur l'état mental du personnage virtuel — ce qui correspond au **contrôle motivationnel**. Si le contrôle motivationnel direct peut nuire à la cohérence des comportements, il est également possible d'effectuer un contrôle motivationnel indirect, en prévoyant un changement d'état mental du personnage suite à un changement d'état d'un objet. On se retrouve alors à nouveau dans une situation de contrôle environnemental. Il est cependant nécessaire de s'assurer que ces ajustements soient cohérents à la fois au niveau des personnages virtuels et au niveau des systèmes techniques ; ainsi, si le contrôle environnemental est réalisé en déclenchant des comportements et en modifiant les états des objets de manière arbitraire, les comportements des personnages virtuels resteront certes cohérents, mais le scénario global ne sera pas explicable.
Nous proposons avec DIRECTOR trois types d'ajustements permettant d'assurer cette cohérence :

- les **happenings**, c'est à dire le déclenchement d'événements exogènes ;
- le **late commitment**, qui consiste à préciser progressivement durant la simulation des états laissés incertains à l'initialisation ;
- les **contraintes d'occurrence**, qui permettent d'outrepasser les choix aléatoires pour les comportements probabilistes.

6.1.1 Happenings

La manière la plus simple d'intervenir sur l'environnement virtuel est par le déclenchement d'événements particuliers, nommés **happenings**. Comme décrit dans la section 5.1, les événements représentés dans le modèle du domaine peuvent être soit endogènes, soit exogènes. Contrairement aux événements endogènes, qui résultent d'un comportement du domaine — que celui-ci fasse suite à une action ou bien à un autre comportement du système technique—, les événements exogènes ne sont pas considérés comme étant la conséquence d'un autre phénomène modélisé, et peuvent donc apparaître dans l'environnement virtuel sans pour autant nécessiter la présence d'une cause permettant de les expliquer. Par exemple, l'arrivée d'un orage ou la réception d'un appel téléphonique pourront correspondre dans certains modèles de domaine à des événements exogènes. Le fait qu'un événement soit considéré comme endogène ou exogène dépend en effet fortement de la couverture du modèle considéré.
Les happenings correspondent aux événements qui sont à la fois **ponctuels** et **exogènes**. Tandis que la valeur d'activation des événements endogènes est déterminée par des règles d'activation qui inspectent les états ou changement d'états du monde auxquels correspondent l'événement, les happenings sont activés explicitement, à la manière des actions. La création et l'activation d'un happening résulte de la réception par le moteur de simulation d'un message contenant l'événement et ses paramètres, par exemple `(PhoneCall has-phone:Phone01)`. L'activation d'un happening va ainsi déclencher un certain nombre de comportements. Les règles associées ont la particularité de ne pas posséder de préconditions. Un exemple est présenté dans le tableau 6.1. Cependant, les comportements associés peuvent posséder des préconditions qui vont déterminer l'application ou non de leurs effets. Par exemple, l'événement `PhoneCall` n'aura aucun effet si le téléphone est éteint, que ce soit pour le comportement `RingingBehaviour` correspondant à la sonnerie du téléphone ou pour le comportement `SparkBehaviour` correspondant à la création d'une étincelle.

```
[PhoneCallTriggering:
(?evt rdf:type :PhoneCall)
(?evt domain:has-active-state ?state)
(?state domain:active domain:True)
(?phone rdf:type :Phone)
(?evt :concerns-phone ?phone)
->
SetValue(?state, domain:Active, domain:False)
CreateBehaviour(:RingingBehaviour, ?beh)
(?beh domain:has-active-state ?behstate)
SetValue(?behstate, domain:active, domain:True)
SetValue(?beh, :target-phone, ?phone)
CreateBehaviour(:SparkBehaviour, ?beh2)
(?beh2 domain:has-active-state ?behstate2)
SetValue(?behstate2, domain:active, domain:True)
SetValue(?beh2, :target-phone, ?phone)]
```

TABLE 6.1 – Règle liée au déclenchement de l'événement exogène `PhoneCall`

Les événements exogènes et duratifs sont également activés et désactivés de manière externe, à l'aide d'happenings qui correspondent au début et à la fin de l'événement. Ainsi, l'événement duratif `Rain` correspondant à une averse sera activé lors du déclenchement de l'événement ponctuel `RainStart`, correspondant au début de l'averse, et arrêté au moment du déclenchement de l'événement ponctuel `RainStop`.

6.1.2 Late commitment

Dans la plupart des environnements virtuels, le scénario de la simulation est limité par les conditions initiales. Cependant, il existe dans certains cas des états qu'il n'est pas nécessaire de définir dès le début de la simulation. Prenons l'exemple d'une vanne qui possède un joint, qui, s'il est usé, peut causer une fuite lorsque le bouchon de la vanne est enlevé. Tant que la vanne a son bouchon, l'état du joint n'est pas connu de l'utilisateur, et ce n'est qu'en enlevant le bouchon qu'il pourra inférer l'état du joint à partir de la présence ou non d'une fuite. On pourrait donc imaginer garder cet état indéterminé jusqu'à ce que le bouchon soit enlevé, et qu'il soit déterminé à ce moment en fonction de si l'on souhaite déclencher ou non la fuite. Le même principe peut être appliqué à tous les états qui ne sont pas directement perceptibles par l'utilisateur, du moment qu'ils n'influencent pas les actions des personnages virtuels ou les réactions des systèmes techniques.
Le **late commitment**, proposé par [Swartjes, 2010], consiste à permettre l'ajout, au cours de la simulation, d'un ensemble de faits qui seront considérés rétrospectivement comme faisant partie des conditions initiales. Ces faits sont ajoutés en fonction des besoins du scénario, à partir d'un ensemble de *framing operators* qui définissent les éléments pouvant y être ajoutés. Notre modèle du late commitment s'inspire de ce principe, mais se rapproche davantage des travaux de [Riedl, 2004] sur l'Initial State Revision, en cela que ce dernier déclare explicitement les états considérés comme indéfinis et les ensembles de valeurs pouvant être prises par ces états. Les travaux sur l'ISR portant sur la génération d'un scénario hors-ligne, la nécessité de spécifier les valeurs de ces états apparaît au cours du processus de planification, lorsque ces états interviennent dans les préconditions d'une action. DIRECTOR réalisant au contraire la génération du scénario et l'exécution des ajustements de manière dynamique au cours du déroulement de la simulation, il s'agira ici de spécifier ces valeurs soit suite aux besoins du scénario, soit suite aux besoins de la simulation elle-même, lorsque les états interviennent dans les règles de comportement des objets de l'environnement virtuel.
Notre modèle permet ainsi de déclarer, à partir d'un modèle de domaine donné, un certain nombre d'états pouvant faire l'objet d'un tel mécanisme. On parlera d'une opération de "commitment" lorsqu'il s'agira de réduire l'ensemble des valeurs possibles associé à un tel état. Ces états sont indiqués dans le modèle de domaine comme étant des sous-concepts d'un composant `Commitable`. Les valeurs initiales des attributs correspondant à ces états sont alors indéfinies : les attributs booléens sont associés à la valeur `domain:Unknown`, tandis que les attributs numériques sont associés à une valeur sous forme d'intervalle. Cela permet ainsi de raffiner progressivement la valeur des attributs numériques : dans le cas où une règle de comportement nécessite seulement de savoir si une valeur est supérieure ou non à un seuil donné, il sera possible de réduire l'intervalle sans pour autant spécifier une valeur particulière.

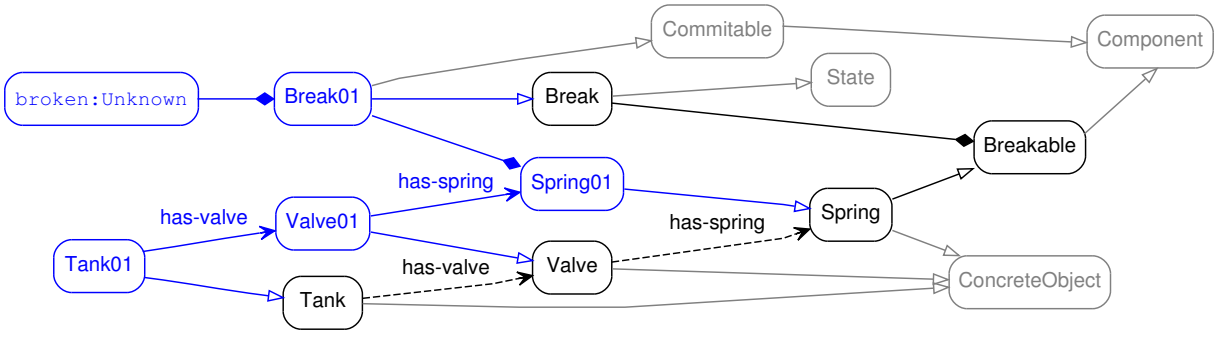


FIGURE 6.1 – Exemple d'état "commitable" dans le modèle de domaine

Les états pouvant être considérés comme "commitables" ne sont pas les états génériques spécifiés dans le modèle du domaine (ex : ouverture, rotation, etc.), mais au contraire des **instances d'états** associées à des objets particuliers. Par exemple, on peut vouloir déclarer que l'état d'ouverture d'une vanne est inconnu, puisqu'il n'est pas perceptible immédiatement, tandis que l'état d'ouver-

ture d'une porte doit être spécifié dès l'initialisation. Le fait qu'un état soit commitable ou non ne peut donc être précisé directement au niveau des concepts d'états de l'ontologie. Pour spécifier ces états, on utilise donc des **règles d'initialisation** qui précisent les propriétés des états pouvant faire l'objet d'opérations de "commitment", permettant ainsi des descriptions complexes. Un exemple de règle d'initialisation est présenté dans le tableau 6.2, tandis que la figure 6.1 correspond à la portion de modèle de domaine associée.

```
[SpringBreakInitialisation:
(?tank :has-valve ?valve)
(?valve :has-spring ?spring)
(?spring :has-break-state ?state)
->
SetValue(?state domain:is-a domain:Commitable)]
```

TABLE 6.2 – Exemple de règle d'initialisation de la propriété `Commitable` d'un état

Les opérations de commitment peuvent être directement effectuées par DIRECTOR, afin de réaliser un ajustement sur le scénario. Elle peuvent également être initiées par le moteur de simulation du monde : lorsque les connaissances sur un état donné sont insuffisantes pour calculer la valeur de vérité d'une précondition d'un comportement, le moteur de simulation peut envoyer une requête à DIRECTOR lui demandant de préciser la valeur de cet état. Dans le cas où DIRECTOR a déjà prévu un ajustement à réaliser sur cette valeur, il peut envoyer en retour l'opération de commitment prévue ; dans le cas contraire, la valeur de cet état n'ayant pas d'influence sur le scénario planifié, il optera pour une valeur par défaut. Dans le cas où la requête du moteur de simulation vise à lever l'incertitude sur une précondition particulière, cette condition pourra être transmise à DIRECTOR, qui pourra ainsi adopter une approche de *least commitment* en spécifiant un intervalle plutôt qu'une valeur spécifique. Le diagramme de séquence correspondant à ces messages est présenté dans la figure 6.2.

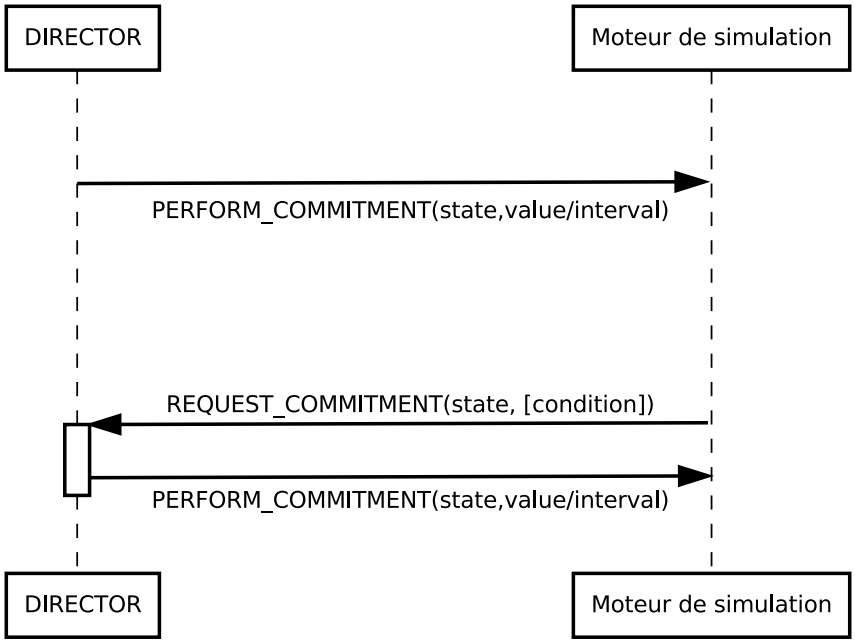


FIGURE 6.2 – Diagramme de séquence concernant les opérations de "commitment"

Le *late commitment* ne concerne pas seulement les états des systèmes techniques simulés, mais peut également s'appliquer aux états internes des personnages virtuels autonomes. Ici encore, il est

nécessaire que le comportement du personnage reste cohérent, et donc que l'état concerné n'intervienne pas dans son processus de prise de décision tant qu'il est indéfini. Par exemple, sur une procédure donnée, il est possible que les opérateurs experts et les opérateurs novices se comportent de manière similaire pour réaliser les premières tâches de la procédure. Il est donc possible de laisser le niveau d'expérience d'un personnage virtuel indéfini jusqu'à ce qu'il intervienne dans une précondition ou un marqueur d'une des tâches considérées dans le modèle d'activité. C'est dans ce cas le moteur de simulation des personnages virtuels qui prendra le rôle du moteur de simulation représenté dans la figure 6.2 pour demander à DIRECTOR de lever l'incertitude sur la valeur.

6.1.3 Contraintes d'occurrence

Le troisième type d'ajustements pouvant être réalisé par DIRECTOR est la spécification de **contraintes d'occurrence** sur des comportements probabilistes.
Selon la granularité de la modélisation du domaine, il peut exister un certain nombre de règles régissant des comportements qui soient simplifiées grâce à l'utilisation de probabilités. Par exemple, dans le cas d'une fuite, il est possible dans certains cas que des éclaboussures atteignent les personnes présentes aux alentours. Si la modélisation du domaine ne va pas jusqu'au niveau de la simulation des fluides, il est possible de représenter de telles conséquences avec une règle de probabilité, par exemple en modélisant qu'il y a 30% de chances que des éclaboussures se produisent. Dans de tels cas, il devient possible de forcer de manière dynamique l'occurrence, ou la non-occurrence du comportement, plutôt que de déterminer celle-ci de manière aléatoire.
Ces règles sont modélisées en DOMAIN-DL à l'aide de la primitive `Draw`, qui permet d'effectuer un tirage aléatoire et de comparer directement le résultat de ce tirage à une valeur de probabilité donnée, représentant la fréquence d'occurrence du comportement. Ce tirage est associé à un identifiant. Un exemple de règle est présenté dans le tableau 6.3.

```
[LeakSplattering:
(?ev rdf:type :Leak)
(?ev domain:has-active-state ?state)
(?state domain:active domain:True)
(?ev :concern-object ?obj)
(?agt rdf:type domain:Agent
(?agt :is-close-to ?obj)
Draw(“LeakSplattering”, 0.3)
->
CreateBehaviour(:SplatteringBehaviour, ?beh)
(?beh domain:has-active-state ?behstate)
SetValue(?beh, :target-source, ?obj)
SetValue(?beh, :target-dest, ?agt)
SetValue(?behstate, domain:active, domain:True)]
```

TABLE 6.3 – Règle de déclenchement probabiliste du comportement `SplatteringBehaviour` suite à l'événement `Leak`

Tant que la probabilité originelle associée au tirage est supérieure à 0 et inférieure à 1, il est possible de contraindre l'occurrence du comportement en associant à l'identifiant du tirage une nouvelle valeur de probabilité. Ainsi, l'envoi par DIRECTOR d'un message contenant `(“LeakSplattering”, 1)` aura pour effet d'assurer le déclenchement du comportement `Splattering` lors de la prochaine activation de l'événement `Leak`.

6.2 Représentation des scénarios

Ces ajustements sont utilisés par DIRECTOR pour modifier le scénario de l'environnement virtuel. Pour représenter ces scénarios, DIRECTOR utilise un **formalisme de représentation** sous forme de **plans partiellement ordonnés**.
Nous avons présenté dans la section 3.3 différents formalismes utilisés pour la représentation de scénarios particuliers et d'espaces de scénarios. Suite à cette étude, nous avons pu extraire différentes propriétés nécessaires pour la représentation d'un scénario particulier destiné à être suivi et réalisé dans un environnement virtuel. D'une part, le formalisme utilisé doit être **interprétable**. D'autre part, il doit être suffisamment **expressif** pour permettre de représenter les **actions de l'utilisateur**, mais aussi celles des **personnages virtuels** et les **comportements des systèmes techniques simulés**, auxquels nous ajouterons les **ajustements** décrits dans la section précédente, ainsi que les liens de causalité entre ces éléments.

6.2.1 Plan partiellement ordonné

Pour représenter ces éléments, DIRECTOR utilise un formalisme sous forme de plans partiellement ordonnés, associé à un moteur de génération de scénarios construit autour d'un moteur de planification. La planification permet en effet de considérer un scénario dans son ensemble, afin de mener à une situation finale particulière, ce qui n'est pas le cas de moteurs basés sur de la sélection locale d'événements.

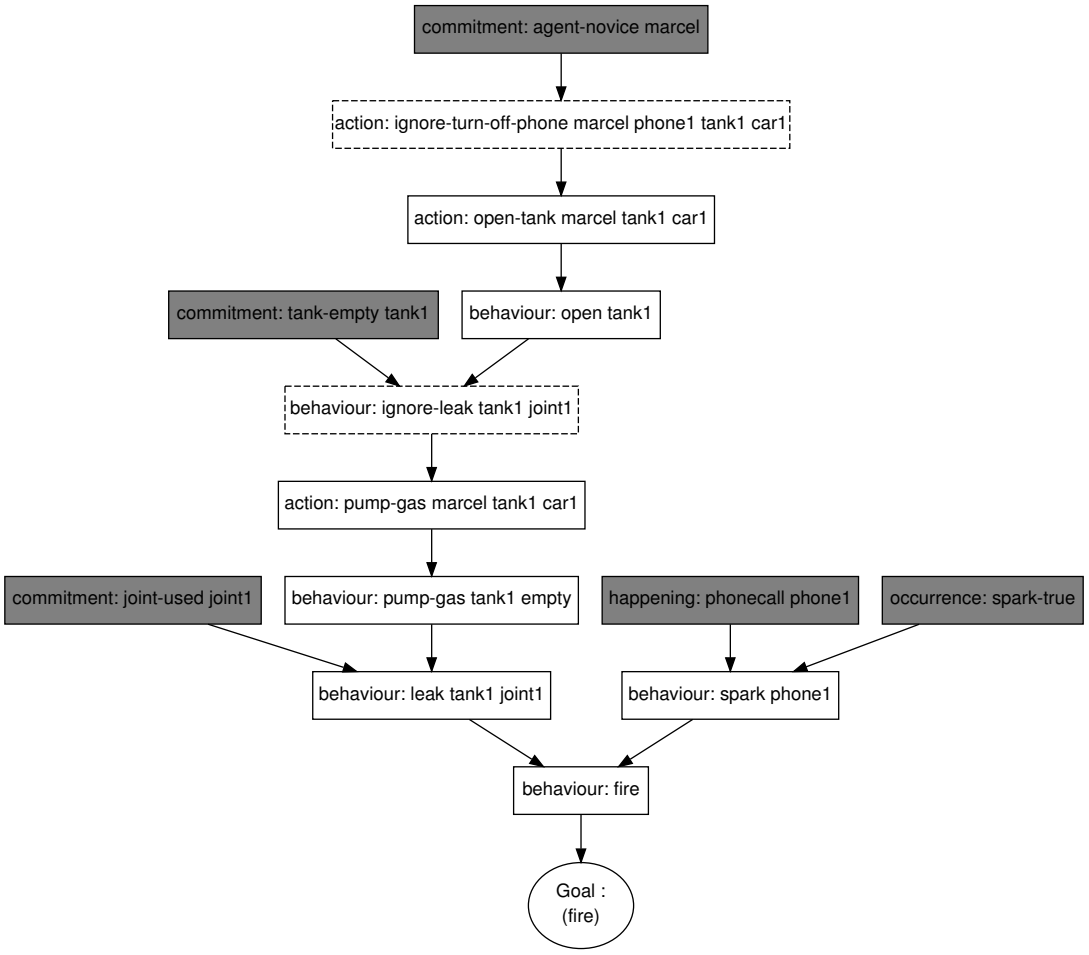


FIGURE 6.3 – Exemple de scénario représenté sous forme de plan partiellement ordonné

Cette représentation sous forme de plans partiellement ordonnés permet de représenter de manière explicite les liens de causalité entre éléments du scénario, et pas uniquement les liens de temporalité. En particulier, elle permet de représenter explicitement les préconditions d'une étape du scénario qui sont rendues vraies par une étape précédente, de façon à pouvoir raisonner sur le scénario et notamment détecter les situations où celui-ci est en échec.
Les scénarios utilisés ici ont vocation à désigner un déroulement particulier des événements, en représentant un ordonnancement partiel d'actions et de comportements menant à une situation finale. Ils ne concernent cependant qu'un sous-ensemble des objets et des agents présents dans l'environnement virtuel. Un scénario n'est donc pas une description exhaustive de tous les changements d'états ayant lieu dans la simulation ; il peut être réalisé de différentes manières du point de vue du scénario global de l'environnement virtuel, selon la façon dont les étapes qui le composent seront ordonnées, et selon les événements qui se dérouleront en parallèle de manière indépendante de ce scénario.

Scénario

Un scénario est un tuple $\langle E, I, L, S_I, S_F \rangle$ tel que E est un ensemble d'étapes, I est un ensemble de contraintes d'instanciation des paramètres des étapes de E, L est un ensemble de liens de causalité entre ces étapes, S_I est un ensemble d'assertions correspondant à l'état initial et S_F est un ensemble d'assertions correspondant à l'état final. Chaque étape est associée à un opérateur.

Un exemple de scénario est présenté dans la figure 6.3

6.2.2 Opérateurs

Les opérateurs sont les éléments de base du scénario. Un opérateur possède un ensemble de paramètres, qui correspondent aux objets qu'il concerne. Ces paramètres sont désignés par des variables typées ; les opérateurs utilisés au niveau des étapes d'un plan donné sont instanciés, c'est à dire que les variables qu'ils utilisent sont associées à des instances d'objets de l'environnement virtuel.
Les opérateurs contiennent également des préconditions — des assertions qui doivent être vérifiées pour que l'opérateur soit applicable — et des effets — des assertions qui deviennent vraies une fois que l'opérateur a été appliqué. Le lien de causalité $o_1 \xrightarrow{c} o_2$ indique que les effets de l'opérateur o_1 établissent une condition c nécessaire pour que l'opérateur o_2 puisse être appliqué.
Les opérateurs peuvent être de deux types : opérateurs de **prédiction** ou opérateurs d'**ajustement**.

Opérateurs de prédiction

Les opérateurs de prédiction correspondent aux actions et comportements attendus de la part des entités autonomes peuplant la simulation. Ils peuvent correspondre soit à des actions de l'utilisateur ou des personnages virtuels (`Action`), soit à des comportements des systèmes techniques (`Behaviour`).

Les **opérateurs d'action** correspondent à la prédiction des prises de décision des personnages virtuels autonomes ou des utilisateurs. Leurs préconditions permettent de préciser le contexte dans lesquels une telle décision peut être prise. Ils sont générés à partir du modèle d'activité. L'exemple présenté dans le tableau 6.4 correspond à la tâche `OpenValve`, contenue dans la procédure donnée en figure 5.5.

Les **opérateurs de comportement** correspondent à la prédiction des comportements des systèmes

techniques simulés. Ces comportements peuvent faire suite à une action, ou bien à un autre comportement. Ils sont générés à partir du modèle de domaine. L'exemple présenté dans le tableau 6.5 correspond au comportement activé par l'action précédente.

La génération des opérateurs de prédiction à partir des modèles de domaine et d'activité est décrite dans la section 6.4.

Operator	`Action-OpenValve`

Parameters	`?a - agent`
	`?v - valve`
	`?t - truck`

Preconditions	`(task-state Open-Valve ?a pending)`
	`(has-turn ?a)`
	`(not (has-acted ?a))`
	`(parent-parameters-LaunchTruckLoading ?t ?a)`
	`(has-valve ?t ?v)`
	`(opening-state-value ?v false)`

Effects	`(not (task-state Open-Valve ?a pending))`
	`(task-state Open-Valve ?a ongoing)`
	`(activate-behaviour Opening)`
	`(parameters-Opening ?v)`
	`(has-acted ?a)`

TABLE 6.4 – Opérateur d'action correspondant à la tâche `OpenValve`

Operator	`Behaviour-Opening`
Parameters	`?obj - object`

Preconditions	`(activate-behaviour Opening)`
	`(parameters-Opening ?obj)`
	`(opening-state-value ?obj false)`

Effects	`(not (activate-behaviour Opening))`
	`(not (parameters-Opening ?obj))`
	`(not (opening-state-value ?obj false))`
	`(opening-state-value ?obj true)`

TABLE 6.5 – Opérateur de comportement correspondant à l'ouverture d'un objet

Opérateurs d'ajustement

Les opérateurs d'ajustement reflètent les ajustements présentés dans la section 6.1, et peuvent donc correspondre au déclenchement d'un événement exogène (`Happening`), au raffinement de la valeur d'un état indéfini (`Commitment`) ou à une contrainte sur le déclenchement d'un comportement probable (`OccurrenceConstraint`).

Les **opérateurs d'happening** correspondent au déclenchement d'un événement exogène ponctuel. Ils sont générés automatiquement à partir des règles Jena associées à ces événements. Ces opérateurs ont la particularité de ne pas posséder de préconditions, puisque les événements concernés peuvent être déclenchés quel que soit l'état de la simulation. L'exemple d'opérateur présenté dans le tableau 6.6 correspond à l'événement présenté dans le tableau 6.1.

Operator	`Happening-PhoneCall`
Parameters	`?phone - Phone`
Preconditions	-
Effects	`(activate-behaviour Spark ?phone)` `(activate-behaviour Ringing ?phone)`

TABLE 6.6 – Opérateur de planification correspondant au happening `PhoneCall`

Les **opérateurs de commitment** correspondent au raffinement d'une valeur incertaine, d'après le principe du *late commitment*. Ils sont générés à partir des règles d'initialisation des états "commitables", et des types associés à ces états. L'exemple d'opérateur présenté dans le tableau 6.7 est généré à partir de la règle d'initialisation présentée dans le tableau 6.2.

Operator	`Commitment-SpringBreak-True`

Parameters	`?spring - Spring`
	`?tank - Tank`
	`?valve - Valve`

Preconditions	`(has-valve ?tank ?valve)`
	`(has-spring ?valve ?spring)`
	`(break-state-value ?spring unknown)`

Effects	`(not (break-state-value ?spring unknown))` `(break-state-value ?spring true)`

TABLE 6.7 – Opérateur de planification raffinant la valeur de l'état de cassure d'un ressort

Les **opérateurs de contraintes d'occurrence** permettent de forcer le déclenchement ou le non-déclenchement de conséquences liées à des règles probabilistes. Deux opérateurs de contraintes d'occurrence sont créés pour chaque identifiant utilisé dans une primitive `Draw` : un qui permet de forcer l'occurrence du comportement associé à l'identifiant, et un qui permet de forcer sa non-occurrence. L'exemple d'opérateur présenté dans le tableau 6.8 permet d'influencer le tirage aléatoire réalisé dans la règle 6.3.

Operator	`Occurrence-LeakSplattering-True`
Parameters	-
Preconditions	-
Effects	`(force-occurrence LeakSplattering)`

TABLE 6.8 – Opérateur de planification contraignant l'occurrence du comportement `SplatteringBehaviour`, lié au tirage aléatoire `LeakSplattering`

6.3 Particularité de la génération de plans prédictifs

Contrairement aux scénarios générés par des systèmes de scénarisation à base de planification classiques, comme Mimesis [Riedl et al., 2003], les scénarios générés par DIRECTOR n'ont pas vocation à être exécutés dans l'environnement virtuel en déclenchant des actions des personnages virtuels et des comportements des systèmes techniques, mais décrivent une prédiction souhaitée des événements de la simulation, associée à un ensemble d'ajustements nécessaires pour réaliser cette prédiction.
Le but de la phase de génération des scénarios n'est donc pas de générer le plan optimal entre une

situation initiale S_I et une situation finale S_F, en regard de sa longueur, de son intérêt narratif, etc. Il s'agit au contraire de trouver un ensemble d'ajustements permettant d'amener la simulation de S_I à S_F, et de prédire les actions des personnages virtuels autonomes et les comportements des systèmes techniques à partir de S_I et compte tenu de ces ajustements.
Dès lors, il n'est pas possible de traduire directement les actions réalisables par les personnages virtuels en opérateurs de planification, et d'utiliser ces opérateurs pour calculer un plan entre S_I et S_F : il n'y aurait alors aucune garantie que les actions composant le plan soient effectivement choisies par les personnages virtuels autonomes durant la simulation.
De plus, les situations que l'on cherche à atteindre correspondent le plus souvent à des situations accidentelles auxquelles confronter un apprenant, ou bien à étudier dans le cadre d'un système d'aide à la décision. La recherche d'un plan optimal en terme de nombre d'étapes pour aller vers de telles situations donnerait ainsi des résultats absurdes, les personnages virtuels semblant adopter des comportements de sabotage, voire de suicide, pour arriver au plus vite à l'accident. La figure 6.4 représente un scénario généré de cette manière, à partir des actions et des situations utilisées pour le scénario représenté dans la figure 6.3.

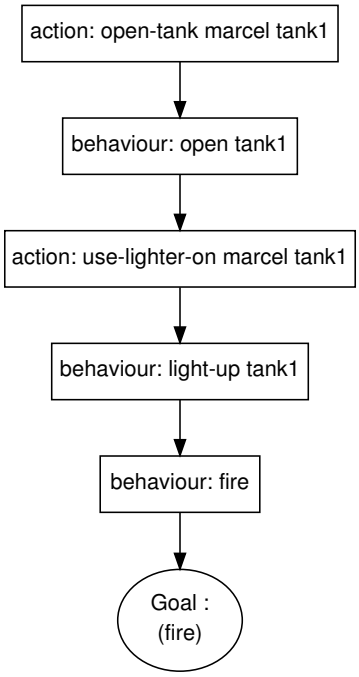


FIGURE 6.4 – Exemple de scénario planifié à partir des actions

Il serait alors envisageable d'utiliser non pas les actions, mais les ajustements, comme étapes du plan, en faisant alors correspondre à chaque étape un scénario candidat, c'est à dire une prédiction de la simulation compte tenu des ajustements appliqués jusque là. Cela nécessiterait cependant de simuler un scénario entier pour chaque combinaison d'opérateurs, et, sans garantie que la situation but soit atteinte dans un scénario candidat donné, il serait impossible de savoir combien d'étapes prédire dans ces scénarios.
Il est donc nécessaire d'inclure dans les opérateurs considérés à la fois les opérateurs d'ajustement et les opérateurs de prédiction, et de cadrer ces derniers afin que leur application soit déterministe et reflète le déroulement réel de la simulation. En particulier, il est nécessaire que les opérateurs correspondant aux actions des personnages virtuels expriment les connaissances contenues dans le modèle d'activité, et non uniquement celles du modèle du domaine. Pour un personnage virtuel donné, un seul opérateur d'action doit être applicable dans une situation donnée : celui correspondant à l'action que le personnage virtuel autonome choisirait effectivement de réaliser dans la simulation.

6.4 Génération d'opérateurs de prédiction

Les opérateurs de prédiction sont générés automatiquement à partir des modèles qui sous-tendent la simulation, afin que, pour chaque situation possible, un seul opérateur soit applicable.
Deux types d'opérateurs de prédiction sont générés :

- les **opérateurs d'action** sont générés à partir du modèle d'activité représenté en **ACTIVITY-DL** ;
- les **opérateurs de comportement** sont générés à partir du modèle du domaine représenté en **DOMAIN-DL**.

6.4.1 Génération des opérateurs d'action

Les opérateurs d'action correspondent aux prédiction des prises de décision des agents. Les effets de ces opérateurs incluent le déclenchement des comportements associés à l'action. Ces opérateurs sont générés à partir de modèles représentés en ACTIVITY-DL, le langage de représentation de l'activité décrit dans la section 5.2. Nous décrivons ici les règles de traduction utilisées pour les propriétés suivantes du langage : tâches feuilles, tâches mères, constructeurs séquentiels ordonnés ou non, préconditions favorables, nomologiques, contextuelles et règlementaires, conditions de satisfaction, tâches expertes et Activités Limites tolérées par l'Usage, ainsi que la gestion du tour par tour (alternance des actions des différents agents). Les constructeurs parallèles ainsi que les tâches itératives n'ont pour le moment pas été traités.

Tâches feuilles

Les tâches feuilles du modèle d'activité correspondent à des actions concrètes. Chaque tâche est associée à un opérateur d'action, tel celui ayant été présenté dans le tableau 6.4.
Une tâche peut être indiquée comme étant en attente (`pending`), en cours (`ongoing`) ou réalisée (`done`). Les opérateurs correspondant aux actions peuvent être appliqués si la tâche correspondante est indiquée comme en attente, et ont pour effet d'activer les comportements liés à l'action, et d'indiquer la tâche comme étant en cours de réalisation.
Plusieurs opérateurs peuvent correspondre à une même action si celle-ci est présente dans plusieurs taches feuilles du modèle d'activité. Les opérateurs diffèrent alors par leurs préconditions, qui correspondent aux conditions de réalisation des tâches feuilles.

Tâches mères et tâches filles ordonnées

L'organisation hiérarchique des tâches du modèle d'activité, qui reflète les différents niveaux de granularité des motivations des agents, est traduite par le biais d'**opérateurs abstraits**, qui expriment l'adoption d'une tâche mère, c'est à dire d'un but (`Abstract-Parent-<Task>`) ou la fin d'une tâche (`Abstract-End-<Task>`). De même que les paramètres d'une action sont transmis aux comportements déclenchés par cette action, le contexte d'une tâche mère est transmis à ses tâches filles par l'intermédiaire d'un prédicat `parameters-parent-<Task>`.
Dans un premier temps, le plan généré contient ces opérateurs abstraits, puis il est nettoyé pour ne garder dans le scénario que les opérateurs d'action qui correspondent à la réalisation d'une action concrète par un agent. La figure 6.5 contient ainsi un plan généré à l'aide d'opérateurs abstraits correspondant à une tâche mère liée à ses tâches filles par un constructeur de type `SEQ-ORD`, et la figure 6.6 correspond à ce même plan une fois nettoyé.

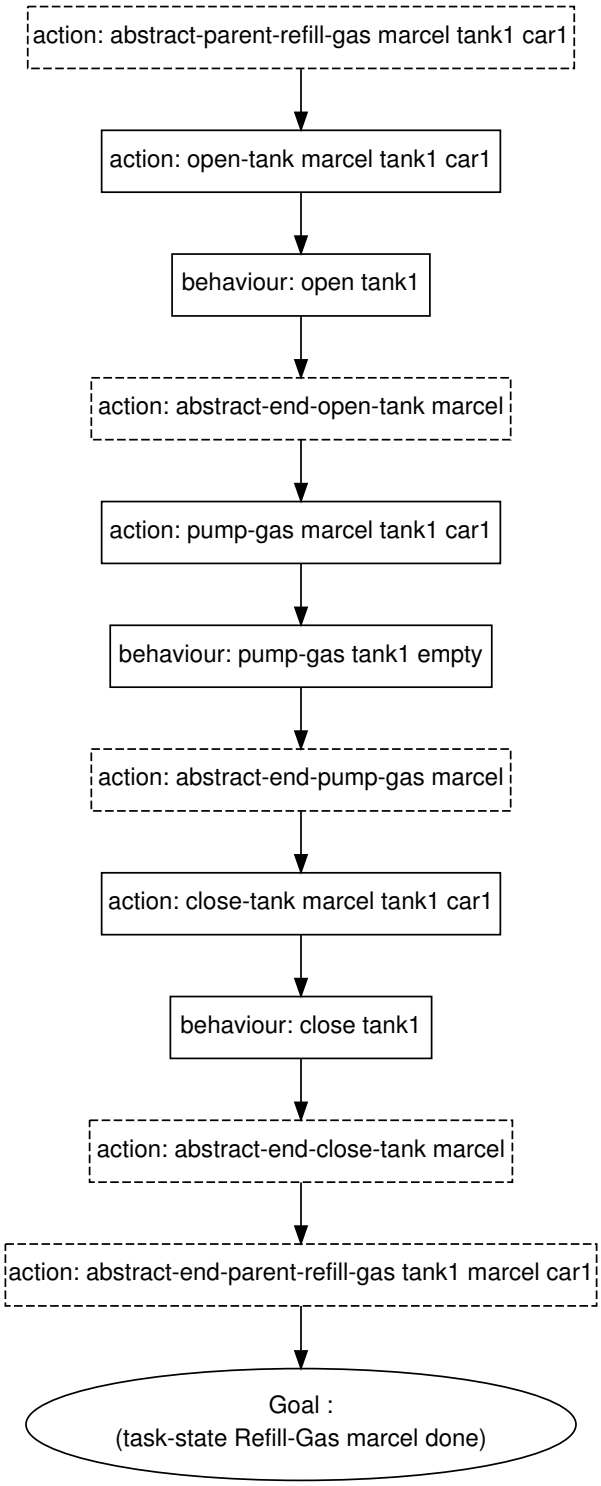


FIGURE 6.5 – Exemple de plan contenant des opérateurs abstraits

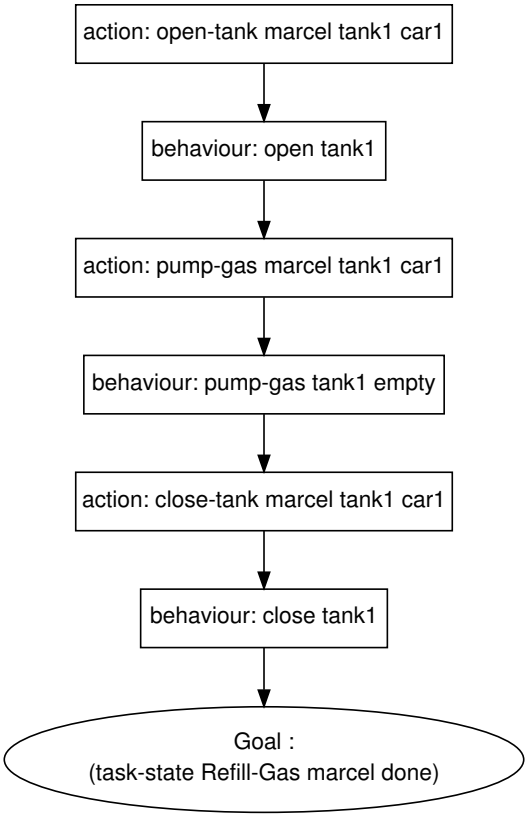


FIGURE 6.6 – Exemple de plan 6.5 une fois les tâches abstraites retirées

Le constructeur `SEQ-ORD` exprime l'ordonnancement strict des tâches filles d'une même tâche mère. Dans ce cas, la première tâche fille est indiquée comme en attente dans les effets de l'opérateur correspondant à l'adoption de la tâche mère, et chaque tâche suivante est ajoutée en attente dans les effets de l'opérateur correspondant à la fin de la tâche qui la précède. Des exemples d'opérateurs abstraits correspondant à l'adoption d'une tâche mère et à la fin d'une tâche feuille sont présentés respectivement dans les tableaux 6.9 et 6.10.

Operator	`Action-Abstract-Parent-Refill-Gas`

Parameters	`?a - agent`
	`?t - tank`
	`?c - car`

Preconditions	`(not (task-state Refill-Gas ?a ongoing))`
	`(has-turn ?a)`
	`(not (has-acted ?a))`
	`(has-tank ?car ?tank)`
	`(forall (?a2 - Agent)`
	`(not (parent-parameters-Refill-Gas ?tank ?a2 ?car))`

Effects	`(task-state Refill-Gas ?a ongoing)`
	`(task-state Open-Tank ?a pending)`
	`(parent-parameters-Refill-Gas ?tank ?a ?car)`

TABLE 6.9 – Opérateur abstrait correspondant à une tâche mère avec un constructeur SEQ-ORD

Operator	`Action-Abstract-End-Open-Tank`
Parameters	`?a - agent`
Preconditions	`(task-state Open-Tank ?a ongoing)` `(not (active-behaviour))`

Effects	`(not (task-state Open-Tank ?a ongoing))`
	`(task-state Open-Tank ?a done)`
	`(task-state Pump-Gas ?a pending)`

TABLE 6.10 – Opérateur abstrait correspondant à une fin de tâche

Tâches filles non ordonnées et conditions favorables

Par défaut, les tâches filles non ordonnées sont réalisées dans l'ordre dans lequel elles sont déclarées au niveau de la tâche mère. Si certaines de ces tâches possèdent des préconditions favorables, et que celles-ci sont vérifiées, alors elles seront réalisées en priorité. Si les préconditions favorables de plusieurs tâches filles sont vraies, alors ces tâches seront là encore réalisées dans l'ordre dans lequel elles sont déclarées. L'algorithme d'ordonnancement utilisé pour ajouter aux opérateurs d'action les préconditions correspondantes est présenté dans l'annexe D.
La figure 6.7 présente deux plans correspondant à une procédure liée aux Equipements de Protection Individuels (EPI) : l'agent peut choisir de revêtir ses lunettes, son casque et ses gants sans contrainte d'ordonnancement. Une condition favorable est spécifiée : dans un cas de pluie, l'agent commencera par mettre son casque. L'opérateur correspondant à cette tâche est présenté dans le tableau 6.11.

Operator	`Action-Put-On-Helmet`
Parameters	`?a - agent` `?helmet - helmet`

Preconditions	`(task-state Put-On-Helmet ?a pending)`
	`(parent-parameters-Put-On-PPE ?a)`
	`(has-turn ?a)`
	`(not (has-acted ?a))`
	`(or (task-state Put-On-Goggles ?a done)`
	`(weather-state-value rain))`

Effects	`(not (task-state Put-On-Helmet ?a pending))`
	`(task-state Put-On-Helmet ?a ongoing)`
	`(activate-behaviour Put-On)`
	`(parameters-Put-On ?a ?helmet)`
	`(active-behaviour)`
	`(has-acted ?a)`

TABLE 6.11 – Opérateur correspondant à une sous-tâche non ordonnée possédant une condition favorable

Conditions nomologiques

Les conditions nomologiques d'une tâche indiquent les états devant être atteints pour que la tâche soit physiquement réalisable. Si ce n'est pas le cas, alors l'agent effectue une action supplémentaire pour les atteindre avant de réaliser la tâche [1]. Au niveau des opérateurs de prédiction, cela

1. Si la situation correspondant à la condition nomologique ne peut être atteinte en une seule action, il serait possible pour l'agent de planifier une suite d'actions à effectuer pour atteindre cet état. Cela n'est pour l'instant pas pris en compte dans le modèle.

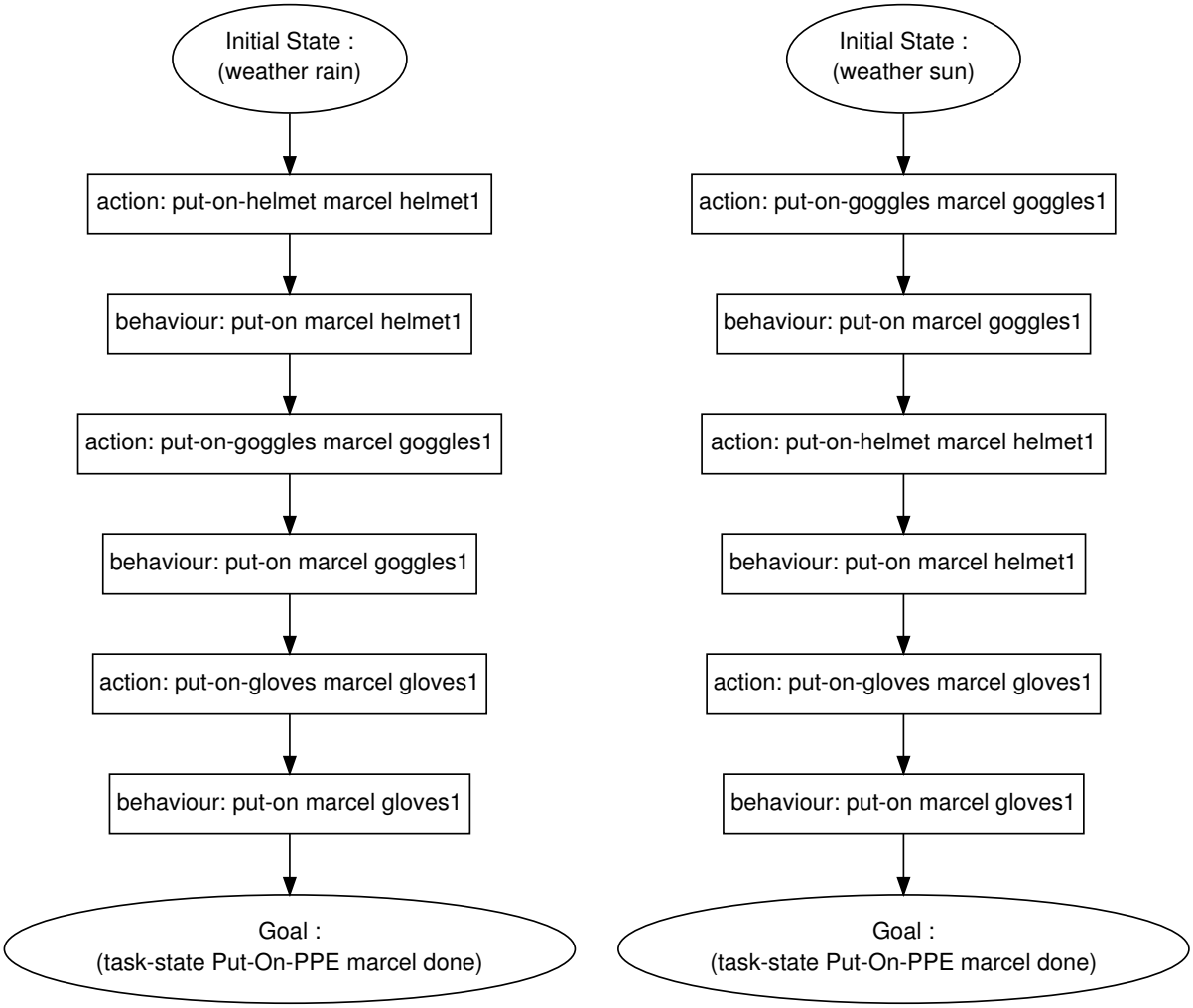


FIGURE 6.7 – Exemple des plans générés à partir d'opérateurs correspondant à des tâches séquentielles non ordonnées

se traduit par l'ajout de trois opérateurs : un opérateur `Action` correspondant à la réalisation de l'action et ayant comme précondition le fait que la condition nomologique soit fausse, un opérateur abstrait `End` correspondant à la fin de cette action, et un opérateur abstrait `Ignore`, pouvant être appliqué si la condition nomologique est vraie. Pour chaque opérateur correspondant à une tâche mère ou de même niveau et ayant pour effet de mettre la tâche en attente, l'identifiant de la tâche est remplacé par celui de l'action nomologique (ou de l'action correspondant à la première condition nomologique déclarée si la tâche en possède plusieurs). Les opérateurs nomologiques reprennent également les préconditions de l'opérateur correspondant à la tâche afin de respecter les contraintes d'ordonnancement.
La génération de ces opérateurs à partir d'une condition nomologique fait appel au modèle du domaine, afin d'identifier l'action permettant de déclencher le comportement qui réalise la condition. Les tableaux 6.12 et 6.13 présentent les opérateurs correspondant à une action `Unlock`, permettant d'atteindre la condition nomologique (`lock-state-value ?tank false`) nécessaire à la tâche `Open Tank`.

Operator	Action-Nomological-Unlock
Parameters	?a - agent ?t - tank ?c - car
Preconditions	(parent-parameters-Refill-Gas ?tank ?a ?car) (has-turn ?a) (not (has-acted ?a)) (has-tank ?car ?tank) (lock-state-value ?tank true) (task-state Nomological-Unlock ?a pending)
Effects	(not (task-state Nomological-Unlock ?a pending)) (task-state Nomological-Unlock ?a ongoing) (activate-behaviour Unlock) (parameters-Unlock ?tank) (active-behaviour) (has-acted ?a)

TABLE 6.12 – Opérateur correspondant à une action réalisant une condition nomologique

Operator	Action-Abstract-Ignore-Unlock
Parameters	?a - agent ?t - tank ?c - car
Preconditions	(parent-parameters-Refill-Gas ?tank ?a ?car) (has-turn ?a) (not (has-acted ?a)) (has-tank ?car ?tank) (lock-state-value ?tank false) (task-state Nomological-Unlock ?a pending)
Effects	(not (task-state Nomological-Unlock ?a ongoing)) (task-state Nomological-Unlock ?a done) (task-state Open-Tank ?a pending)

TABLE 6.13 – Opérateur abstrait correspondant à une condition nomologique déjà réalisée

Conditions contextuelles et conditions de satisfaction

Les conditions contextuelles et les conditions de satisfaction d'une tâche peuvent mener un agent à ignorer cette tâche, dans le cas où les conditions contextuelles ne sont pas vérifées ou dans celui où les conditions de satisfaction le sont déjà. Si une tâche possède ces types de conditions, elle est donc associée à un opérateur `Ignore` ayant dans ses préconditions une disjonction contenant chacune des conditions de satisfaction, ainsi que la négation de chacune des conditions contextuelles. Par exemple, l'opérateur correspondant à l'action d'ouverture du réservoir aura comme précondition que le réservoir ne soit pas déjà ouvert, tandis que l'opérateur `Ignore` associé aura la précondition opposée.
Les conditions de satisfaction sont également ajoutées comme préconditions des opérateurs abstraits de fin de tâche `End`. Dans le cas des tâches mères, le traitement est différent selon que la tâche est satisfaite si toutes ses tâches filles sont satisfaites (`AND`), ou que la tâche est satisfaite si au moins l'une de ses tâches filles est satisfaite (`OR`).

Dans le premier cas, la conjonction des prédicats (`task-state <Task> done`) est ajoutée aux préconditions de l'opérateur `End-Parent`, comme c'est le cas pour l'opérateur présenté dans le tableau

Operator	Action-Abstract-End-Parent-Refill-Gas
Parameters	?a - agent ?c - car
Preconditions	(task-state Open-Tank ?a done) (task-state Pump-Gas ?a done) (task-state Close-Tank ?a done) (task-state Refill-Gas ?a ongoing) (parent-parameters-Refill-Gas ?tank ?a ?car)
Effects	(task-state Refill-Gas ?a done) (not (task-state Refill-Gas ?a ongoing)) (not (parent-parameters-Refill-Gas ?tank ?a ?car))

TABLE 6.14 – Opérateur abstrait correspondant à la fin d'une tâche mère

6.14.
Dans le second cas, l'opérateur End-Parent peut être appliqué si au moins l'une des tâches filles est indiquée comme terminée, et a pour effet d'enlever le statut "en cours" de l'ensemble des tâches filles restantes. De plus, chaque tâche fille possède également comme précondition le fait qu'aucune de ses tâches soeurs ne soit indiquée comme terminée [2].

Conditions règlementaires, tâches expertes et ALUs

La traduction du modèle d'activité en opérateurs de planification prédictifs est fortement dépendante des modèles d'agents utilisés, c'est à dire des hypothèses émises sur la façon dont le module de personnages virtuels va interpréter le modèle d'activité, et sur les mécanismes de prise de décision des utilisateurs. Cette dépendance est particulièrement importante lorsqu'il s'agit de considérer les comportements qui dévient des comportements prescrits, c'est à dire les violations conscientes ou les erreurs inconscientes effectuées par les agents.
Dans un premier temps, nous avons travaillé sur une version basique du modèle des agents ne considérant que deux propriétés : le niveau d'expertise et le niveau de stress. Un agent peut ainsi être novice ou expert dans le domaine, et être stressé ou non. Il s'agit d'une version très simplifiée des modèles utilisés dans les travaux précédents de notre équipe de recherche sur le suivi de l'apprenant [Amokrane, 2010] et la génération de comportements de personnages virtuels [Edward, 2011] [Lhommet, 2012].
Ainsi, le traitement des **conditions règlementaires** des tâches va être conditionné au niveau de stress de l'agent. Les conditions règlementaires d'une tâche sont traduites, à l'instar des conditions nomologiques, par la création d'un opérateur d'action supplémentaire précédant l'opérateur correspondant à la tâche, à la différence que celui-ci possède également comme précondition le fait que l'agent ne soit pas stressé. L'opérateur Ignore associé est ainsi applicable si la condition règlementaire est déjà atteinte ou si l'agent est stressé.
De même, le traitement des **tâches expertes**, taguées comme étant réalisables uniquement par les experts (expert+) ou bien uniquement par les non-experts (expert-) correspondra à l'ajout d'un opérateur Ignore ayant comme précondition le niveau d'expertise de l'agent, et l'ajout de la précondition opposée à l'opérateur Action.
La réalisation des **Activités Limites tolérées par l'Usage (ALUs)** dépend également du niveau d'expertise et du niveau de stress de l'agent : les tâches ALU+ (resp. ALU-) seront réalisées (resp. ignorées) par les agents experts et stressés lorsque les conditions CLU (Conditions Limites tolérées par l'Usage)

2. Ceci suppose que les conditions de satisfaction des tâches feuilles soient équivalentes aux effets de l'action, et que ces actions ne puissent pas échouer si les préconditions nomologiques de la tâche sont respectées. Cette limite est traitée dans la section 10.2.4.

associées sont vérifiées, et ignorées (resp. réalisées) dans les autres cas. Par exemple, s'il s'agit d'une tâche qu'il est toléré de ne pas réaliser, l'opérateur `Ignore` associé aura comme précondition le fait que l'agent soit un expert, le fait qu'il soit stressé, et les conditions `CLU` de la tâche, et l'opérateur `Action` possèdera la précondition inverse.

Gestion du tour par tour

Dans le cas où l'environnement virtuel contient plusieurs agents — personnages virtuels autonomes ou utilisateurs —, ceux-ci vont être considérés au niveau du scénario comme agissant en "tour par tour", c'est à dire en effectuant une action chacun, les uns à la suite des autres.
Cette alternance se fait à l'aide de l'opérateur abstrait `EndTurn`, retranscrit dans le tableau 6.15. Cet opérateur peut être appliqué une fois que tous les comportements déclenchés, directement ou non, par l'action précédente ont été désactivés, et a pour effet de modifier l'agent courant en fonction de l'ordre de tours ayant été défini. Cet ordre est indiqué à l'aide de prédicats précisant quel agent agit immédiatement après un agent donné.
Par défaut, ces prédicats sont calculés de sorte que tous les agents effectuent une action avant que le premier agent de la liste puisse agir à nouveau. Ainsi, si l'environnement virtuel contient trois agents, `gaston`, `marcel` et `robert`, l'état initial transmis au moteur de planification contiendra les prédicats `(has-turn gaston)`, `(turn-order gaston marcel)`, `(turn-order marcel robert)` et `(turn-order robert gaston)`. Si un seul agent est présent, alors il sera déclaré comme agissant après lui-même : `(turn-order marcel marcel)`. Un exemple de plan généré à partir d'un problème contenant deux agents est présenté dans la figure 6.8.

Operator	`Action-Abstract-EndTurn`
Parameters	`?current - agent` `?next - agent`

Preconditions	`(forall (?beh - Behaviour) (not (activate-behaviour ?beh))`
	`(has-turn ?current)`
	`(has-acted ?current)`
	`(turn-order ?current ?next)`

Effects	`(not (active-behaviour))`
	`(not (has-turn ?current))`
	`(not (has-acted ?current))`
	`(has-turn ?next)`

TABLE 6.15 – Opérateur abstrait gérant le changement de tour

6.4.2 Génération des opérateurs de comportement

Les opérateurs de comportement correspondent à la prédiction des comportements des systèmes techniques simulés, qu'ils soient déclenchés par des actions des agents ou bien à la suite d'autres comportements. Leurs effets incluent les changements d'états causés par ces comportements. Ces opérateurs sont générés à partir de modèles représentés en DOMAIN-DL, le langage de représentation du domaine décrit dans la section 5.1.
A chaque comportement du modèle de domaine sont associés deux opérateurs de prédiction : un opérateur `Behaviour`, tel que celui présenté dans le tableau 6.5, qui correspond à l'exécution du comportement en question, et un opérateur abstrait `Ignore`, qui correspond à la non-exécution de ce comportement lorsque les préconditions de ce dernier ne sont pas vérifiées.
Ces opérateurs sont applicables lorsque le comportement est indiqué comme activé, ce qui correspond au prédicat `(activate-behaviour <Behaviour>)`. Ce prédicat peut être ajouté dans les ef-

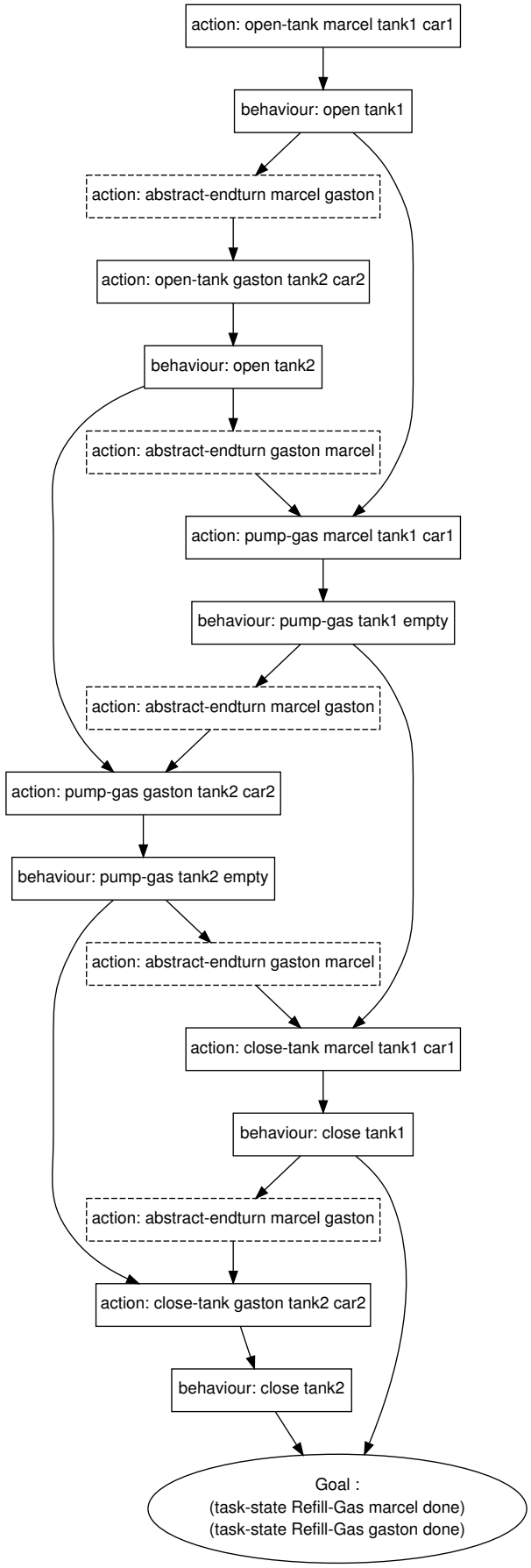


FIGURE 6.8 – Exemple de plan généré avec deux agents

fets d'un opérateur d'action, s'il s'agit du comportement lié à cette action dans le modèle du domaine, ou bien dans les effets d'un autre opérateur de comportement, lorsque le premier est une conséquence du second.
Ces liens entre comportements sont générés à partir du modèle du domaine : pour chaque état modifié par le premier comportement, on récupère les événements dont les préconditions dépendent de cet état, et on ajoute ces préconditions aux opérateurs de comportements indiqués comme déclenchés par ces événements. Par exemple, l'événement correspondant à une fuite causée par un joint défectueux dépend de l'état du joint, de l'ouverture du réservoir et du remplissage du réservoir, et déclenche un comportement de fuite aboutissant à la formation d'une flaque de liquide inflammable. L'activation du comportement de fuite est donc ajoutée aux effets des opérateurs correspondant aux comportements d'ouverture et de remplissage du réservoir. Les conditions sur ces trois états sont également ajoutées aux préconditions de l'opérateur correspondant au comportement de fuite, et la négation de ces conditions est ajoutée à l'opérateur `Ignore` associé, présenté dans le tableau 6.16. La figure 6.9 présente trois plans générés à partir de ces opérateurs, selon différents états initiaux.
Pour les comportements probabilistes, le tirage aléatoire est remplacé par les contraintes d'occurrence dans les préconditions des opérateurs. Les tableaux 6.17 et 6.18 présentent des exemples de tels opérateurs.

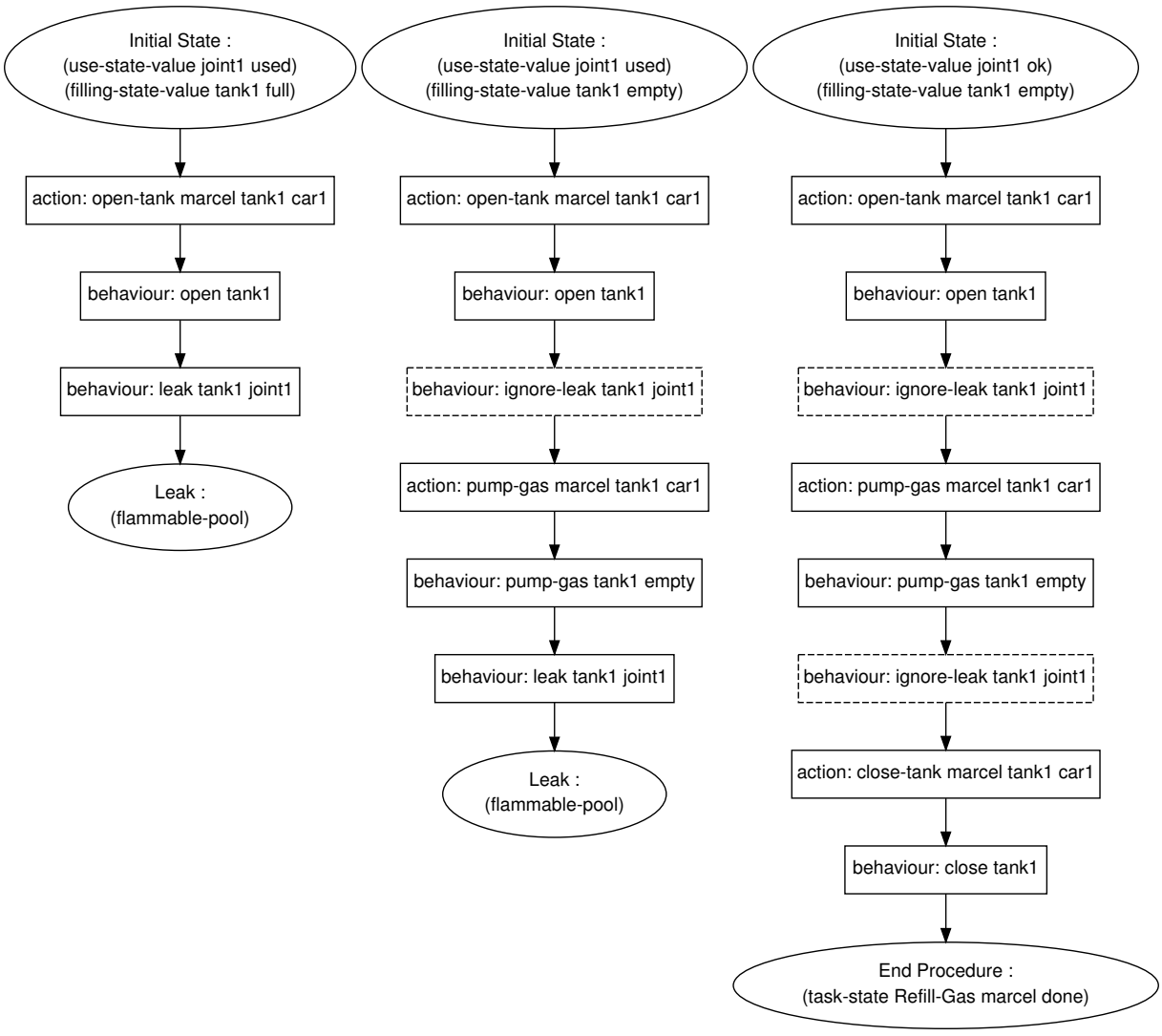


FIGURE 6.9 – Exemples de plans générés à partir d'opérateurs de comportement ignorables

Operator	Behaviour-Ignore-Leak
Parameters	?t - Tank ?j - Joint
Preconditions	(activate-behaviour Leak) (parameters-Leak ?t) (has-joint ?t ?j) (or (use-state-value ?j ok) (fill-state-value ?t empty) (open-state-value ?t false))
Effects	(not (activate-behaviour Leak)) (not (parameters-Leak ?t))

TABLE 6.16 – Opérateur de comportement correspondant au non-déclenchement d'une fuite

Operator	Behaviour-Spark
Parameters	?phone - Phone
Preconditions	(activate-behaviour Spark) (parameters-Spark ?phone) (force-occurrence Spark) (on-state-value ?phone true)
Effects	(not (activate-behaviour Spark)) (not (parameters-Spark ?phone)) (not (force-occurrence Spark)) (ignition-source) (activate-behaviour Fire)

TABLE 6.17 – Opérateur correspondant au déclenchement du comportement probabiliste Spark

Operator	Behaviour-Ignore-Spark
Parameters	?phone - Phone
Preconditions	(activate-behaviour Spark) (parameters-Spark ?phone) (or (force-non-occurrence Spark) (on-state-value ?phone false))
Effects	(not (activate-behaviour Spark)) (not (parameters-Spark ?phone)) (not (force-occurrence Spark)) (not (force-non-occurrence Spark))

TABLE 6.18 – Opérateur correspondant au non-déclenchement du comportement probabiliste Spark

6.5 Processus de génération, suivi et exécution du scénario

La génération des opérateurs de prédiction et d'ajustement à partir des modèles de domaine et d'activité est réalisée en amont de la simulation. Durant la simulation, DIRECTOR utilise ces opérateurs pour générer des scénarios, qui seront ensuite suivi et exécutés dans l'environnement virtuel. Les étapes de ce processus sont représentées dans la figure 6.10. A la réception des objectifs scénaristiques envoyés par le module TAILOR, DIRECTOR génère un scénario à partir de l'état courant du monde. Puis, à chaque message indiquant l'activation d'une action ou d'un comportement, il met à jour l'avancement dans ce scénario et exécute les éventuels ajustements nécessaires. Dans le cas où le déroulement des événements de l'environnement virtuel dévie du scénario généré, un nouveau scénario est généré à partir de l'état courant.

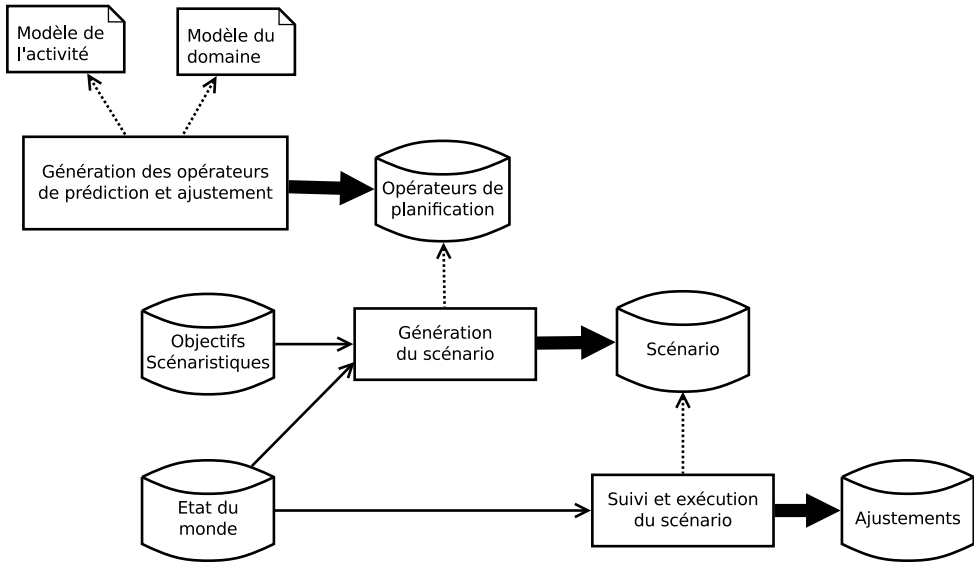


FIGURE 6.10 – Processus de génération, suivi et exécution du scénario par DIRECTOR

6.5.1 Génération du scénario

La première étape de la génération du scénario consiste à utiliser un moteur de planification pour générer un plan à partir de l'ensemble des opérateurs d'ajustement et de prédiction, de l'état courant du monde, et de la situation but ayant été définie dans les objectifs scénaristiques.
Le plan ainsi généré contient des étapes liées à des opérateurs d'action, de comportement et des ajustements, mais également à des opérateurs de prédiction abstraits, qui correspondent aux tâches de haut niveau des agents, ou à l'absence d'action ou d'événement (opérateurs `Ignore`). Un exemple de plan est présenté dans la figure 6.11.

Les étapes liées à des opérateurs abstraits correspondant à des tâches de haut niveau ou à des fins de tâches sont ensuite supprimées du plan ; des liens de causalité sont rajoutés entre les étapes précédant et suivant les étapes supprimées. Les étapes correspondant à des non-événements — liées à des opérateurs `Ignore` — sont conservées dans le plan, en vue de la phase de suivi du scénario.
Les étapes liées à l'opérateur `EndTurn` sont elles aussi supprimées, mais les liens de causalité entre les étapes précédentes (liées aux actions d'un agent *A*) et les étapes suivantes (liées aux actions d'un agent *B*) ne sont pas recréés. En effet, les agents n'agissant pas en tour par tour dans l'environnement virtuel mais en parallèle, l'ordonnancement entre les actions des différents agents, nécessaire pour la génération du plan prédictif, n'a plus lieu d'être dans le scénario. La figure 6.12 présente un exemple de plan une fois les opérateurs abstraits supprimés.

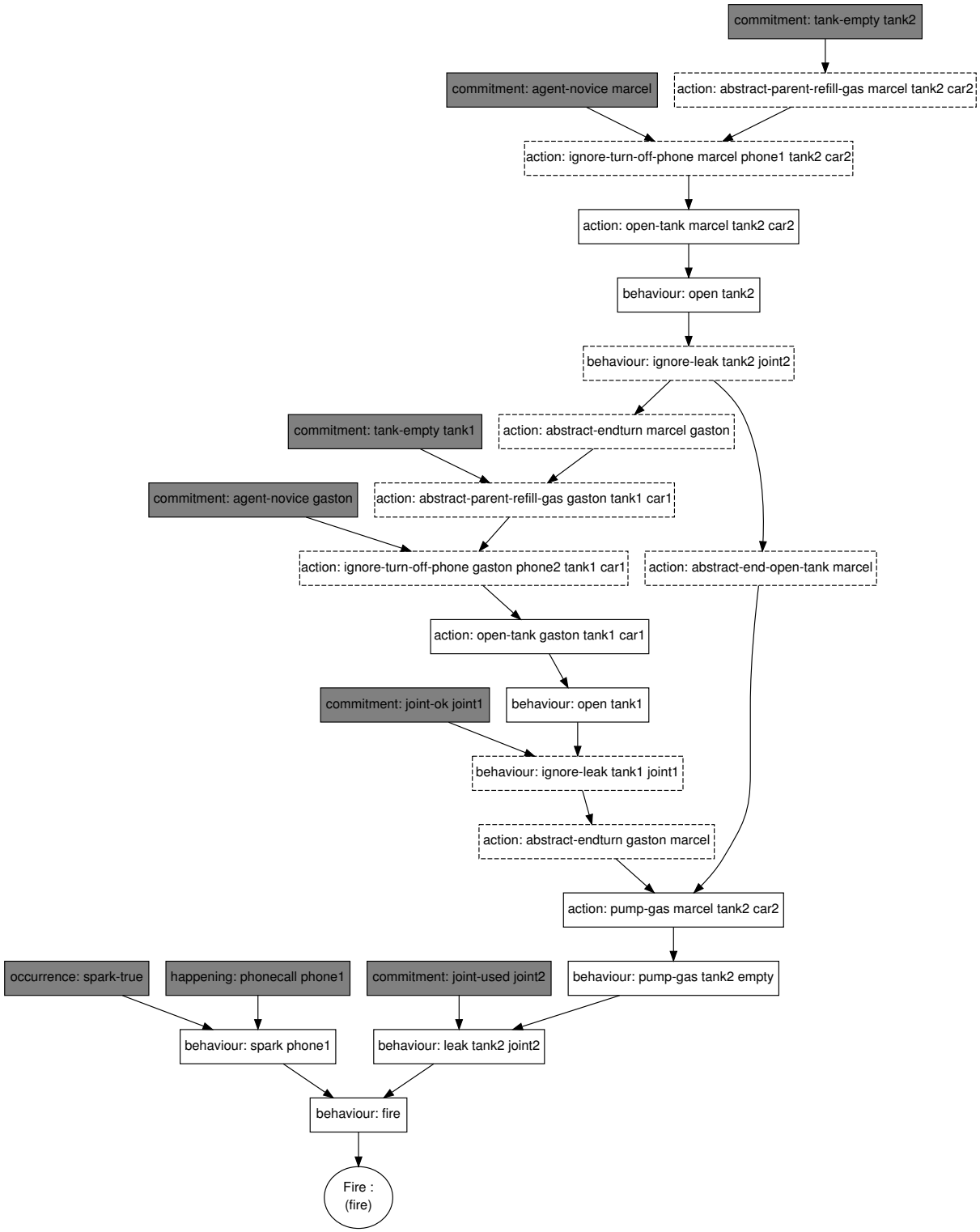


FIGURE 6.11 – Plan généré contenant des opérateurs abstraits

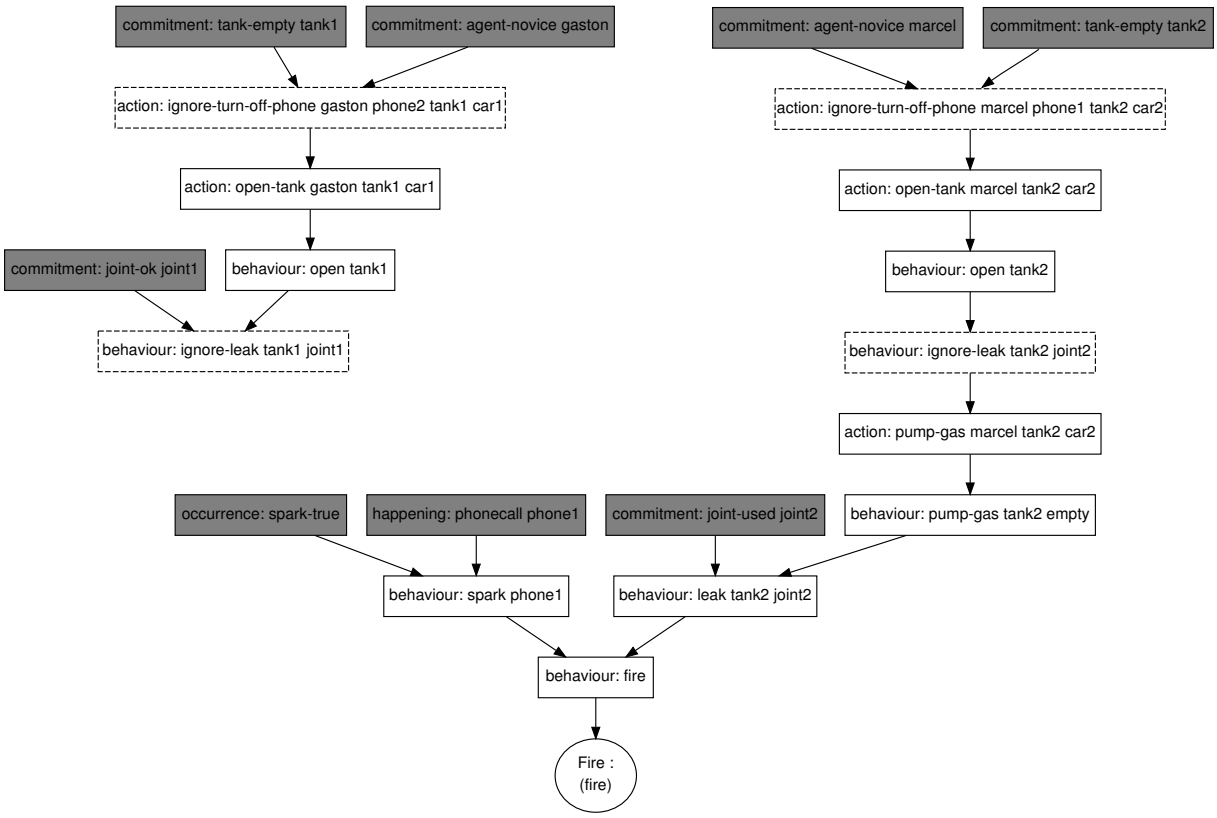


FIGURE 6.12 – Plan 6.11 une fois les opérateurs abstraits supprimés

Le plan “nettoyé” va ainsi contenir des séquences d’actions parallèles, correspondant chacune aux actions d’un agent différent. Dans le cas où les actions d’un agent n’ont pas d’influence sur les actions d’un autre agent et les comportements qu’elles déclenchent, il n’y aura pas de lien de causalité entre les étapes des deux séquences. Si, de plus, les actions de l’agent et les comportements qu’elles déclenchent n’ont pas d’influence sur la situation finale du plan, alors la séquence d’étapes sera complètement indépendante du reste du plan. C’est le cas dans le plan présenté en figure 6.12.
Les séquences d’étapes n’étant pas liées à la situation finale sont alors supprimées du plan. Cependant, les opérations de *commitment* présentes dans la séquence sont gardées en mémoire : ainsi, dans le cas où le moteur de simulation du monde demande à DIRECTOR de raffiner la valeur d’un de ces états, c’est la valeur utilisée dans le plan qui sera renvoyée, puisqu’elle ne met pas en péril le scénario. Pour autant, ces opérations de *commitment* ne sont pas réalisées à l’initiative de DIRECTOR, contrairement à celles présentes dans le scénario final, afin de laisser l’état du monde le plus ouvert possible.
Si la génération du scénario échoue, car aucun plan vers la situation but n’est possible compte tenu de l’état du monde, alors DIRECTOR envoie un message à TAILOR lui demandant de relâcher les contraintes. Il pourra s’agir de choisir une autre situation but ou bien une situation moins précise, par exemple de demander une fuite sans en préciser le type plutôt que de demander un type de fuite particulier. Si la génération du scénario réussit, alors DIRECTOR passe à l’étape de suivi et exécution du scénario.

6.5.2 Suivi et exécution

Une fois le scénario généré, DIRECTOR suit le déroulement des événements dans l’environnement virtuel afin de vérifier que tout se passe comme prévu dans le scénario et de déclencher les ajuste-

ments au moment où ils sont nécessaires.
Pour cela, à chaque réception d'un message indiquant le déclenchement d'un comportement ou d'une action, DIRECTOR met à jour l'état d'avancement du scénario et déclenche les éventuels ajustements nécessaires selon l'algorithme 1.

A la génération du plan, toutes les étapes correspondant à des actions qui ne sont pas précédées par des étapes correspondant à des opérateurs de prédiction sont ajoutées aux étapes suivies, et les ajustements qui les précèdent sont déclenchés. A la réception d'un message informant du déclenchement d'une action ou d'un comportement correspondant à l'une des étapes suivies, DIRECTOR valide l'étape et examine ses étapes suivantes : si toutes les étapes de prédiction précédant une étape donnée sont validées, alors DIRECTOR ajoute cette étape aux étapes suivies, et déclenche les ajustements associés.
Les ajustements ne sont pas déclenchés dès la génération du scénario afin de laisser l'état du monde le plus ouvert possible dans le cas où les objectifs scénaristiques viendraient à changer, ou bien où les actions de l'utilisateur mettraient en péril le scénario prévu et amènerait à devoir générer un scénario différent.
Lorsque le plan comporte des étapes correspondant à des comportements qui ne sont précédées que par des ajustements, comme c'est le cas de l'étape `Spark` dans la figure 6.13, ces ajustements ne sont pas déclenchés dès la génération du plan. Lors de la validation d'une étape, si toutes les étapes de même niveau soit sont déjà validées, soit n'ont pas d'étape de type action dans les chaînes de causalité les précédant, alors les étapes de type comportement en début de ces chaînes sont ajoutées aux étapes suivies et les comportements associés sont déclenchés. Ce sera ici le cas lors de la validation de l'étape `Leak`, qui permettra de déclencher les ajustements `PhoneCall` et `Force-Occurrence Spark`.

Lorsque la prochaine étape à suivre est de type `Ignore`, elle est ajoutée aux étapes suivies, de même que les étapes suivantes de ce type, jusqu'à arriver à une étape correspondant à la réalisation d'une action ou d'un comportement. A la réception d'un message correspondant à cette dernière étape, DIRECTOR valide en même temps cette étape et l'ensemble des étapes `Ignore` qui la précèdent.
Si le moteur de simulation du monde a besoin de raffiner un état particulier, alors DIRECTOR vérifie d'abord si une opération de *commitment* est prévue sur cet état dans le scénario, puis si une telle opération a été enregistrée lors de la phase de génération du scénario. Dans le cas contraire, il ne donne aucune consigne particulière au moteur de simulation, et celui-ci peut choisir une valeur par défaut.
Si le comportement ou l'action activé dans l'environnement virtuel correspond à une étape censée être ignorée, ou que l'action activée est différente de l'action attendue de la part de cet agent, alors le scénario est en échec, et DIRECTOR va générer un nouveau scénario.

6.5.3 Replanification

Lorsque le scénario prévu est en échec, DIRECTOR génère un nouveau scénario à partir de l'état courant du monde, prenant donc en compte les ajustements ayant été effectués jusque là. Afin de pouvoir prédire les actions des agents alors que la simulation est déjà lancée, il est nécessaire de connaître leur avancement dans le modèle d'activité. Pour cela, DIRECTOR interroge le moteur de simulation des personnages virtuels, et le moteur de suivi de l'utilisateur, afin d'obtenir les prédicats `task-state` et `parameters-task` reflétant l'activité des agents. De même, DIRECTOR interroge le moteur de simulation du monde pour obtenir les nouvelles valeurs des états des objets, et des relations entre eux.

Algorithm 1 Algorithme de suivi du scénario et exécution des ajustements

```
function OnScenarioGenerated(Steps, CausalLinks)
    monitoredSteps ← []
    completedSteps ← []
    for all actionStep in GetActionSteps(Steps) do
        inLinks ← GetInLinks(actionStep, CausalLinks)
        if GetSources(inLinks) ∩ GetPredictionSteps(Steps) = [] then
            AddMonitoredStep(actionStep)
function OnActivationMessageReceived(S)
    if S ∈ monitoredSteps then
        ValidateStep(S)
    else if Ignore(S) ∈ monitoredSteps then
        ReplanScenario()
    else
        for all actionStep in monitoredSteps ∩ GetActionSteps(Steps) do
            if actionStep.agent = S.agent and !IsIgnore(actionStep) then
                ReplanScenario()
function ValidateStep(S)
    remove S from monitoredSteps
    RemovePreviousIgnoreSteps(S)
    add S to completedSteps
    for all nextStep in GetTargets(GetOutLinks(S, CausalLinks)) do
        inLinks ← GetInLinks(nextStep, CausalLinks)
        if (GetSources(inLinks) ∩ GetPredictionSteps(Steps)) ⊂ completedSteps then
            AddMonitoredStep(nextStep)
        else
            siblings ← GetSources(inLinks) ∩ GetPredictionSteps(Steps)
            siblings ← siblings\(monitoredSteps ∪ completedSteps)
            for all siblingStep in siblings do
                if GetAncestors(siblingStep) ∩ GetActionSteps(Steps) ≠ [] then
                    continue to next nextStep
            for all siblingStep in siblings do
                for all behStep in GetAncestors(siblingStep) ∩ GetBehaviourSteps(Steps) do
                    behInLinks ← GetInLinks(behStep, CausalLinks)
                    if GetSources(behInLinks) ∩ GetPredictionSteps(Steps) = [] then
                        AddMonitoredStep(behStep)
function AddMonitoredStep(S)
    add S to monitoredSteps
    inLinks ← GetInLinks(S, CausalLinks)
    for all adjustmentStep in (GetSources(inLinks) ∩ GetAdjustmentSteps(Steps)) do
        TriggerAdjustment(adjustmentStep)
    if IsIgnore(S) then
        for all nextStep in GetTargets(GetOutLinks(S, CausalLinks)) do
            AddMonitoredStep(nextStep)
function RemovePreviousIgnoreSteps(S)
    inLinks ← GetInLinks(S, CausalLinks)
    for all previousStep in (GetSources(inLinks) ∩ monitoredSteps) do
        if IsIgnore(previousStep) then
            RemovePreviousIgnoreSteps(previousStep)
            remove previousStep from monitoredSteps
            add previousStep to completedSteps
```

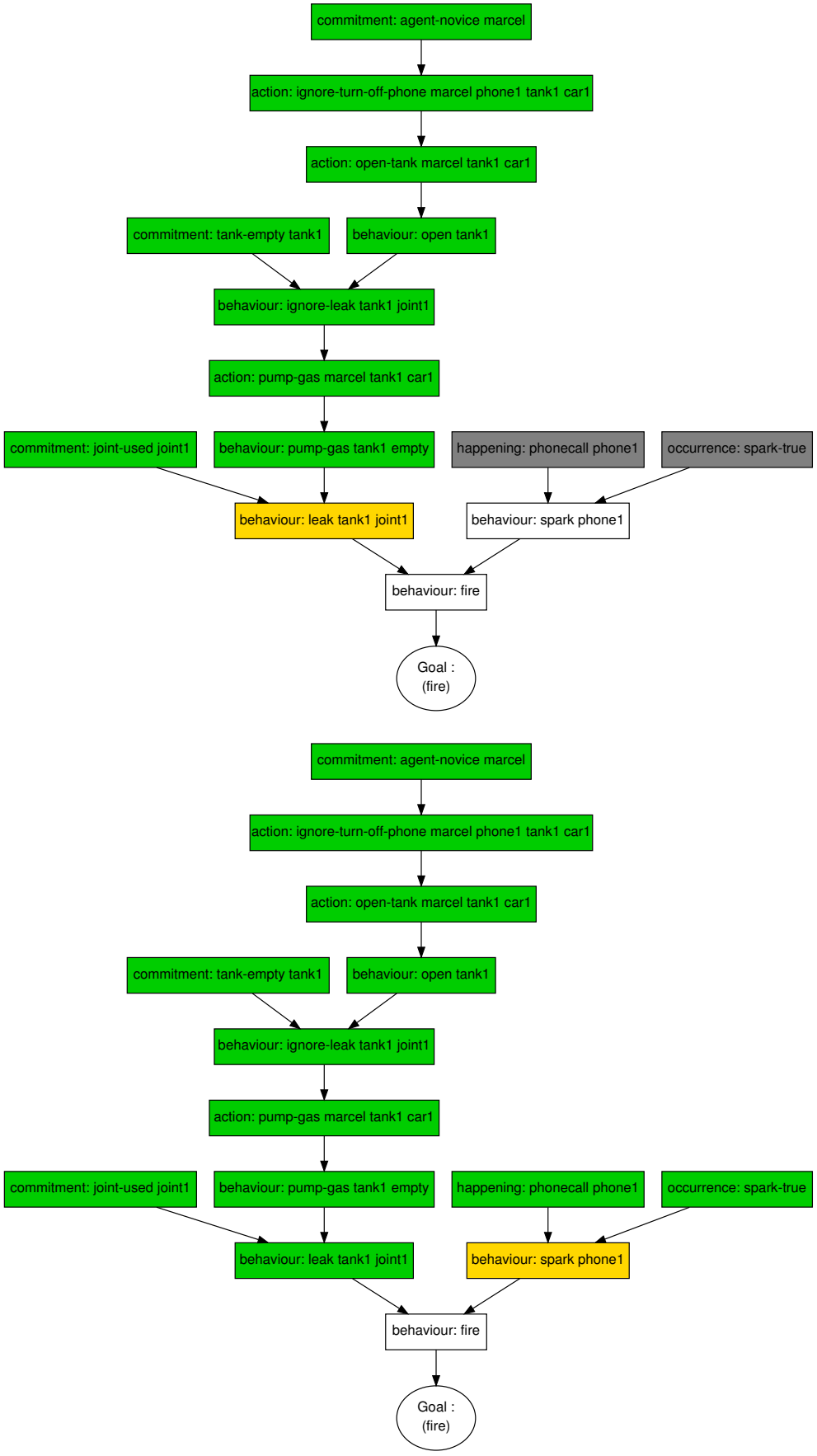


FIGURE 6.13 – Etat d'avancement du scénario 6.3 avant et après la réception d'un message informant de l'activation du comportement correspondant à l'étape `leak tank1 joint1`

6.6 Bilan

Le moteur DIRECTOR permet de générer des scénarios menant à une situation but donnée, en utilisant des opérateurs de prédiction pour planifier une séquence d'étapes basée sur les modèles de domaine et d'activité utilisés par une simulation. Ces scénarios sont réalisés dans l'environnement virtuel par le biais d'ajustements déclenchés au fur et à mesure de l'avancement du scénario via les actions de l'utilisateur ou des personnages virtuels autonomes.
Plutôt que d'exécuter directement le scénario généré en utilisant ses étapes comme des ordres envoyés aux personnages virtuels, DIRECTOR utilise celles-ci pour suivre leurs actions et celles de l'utilisateur. Cela permet d'une part davantage de réactivité face aux perturbations induites par les interactions de ce dernier, puisque les personnages virtuels autonomes sont capables d'adapter leurs actions à une nouvelle situation sans pour autant nécessiter que le scénario entier soit recalculé. Cela permet d'autre part de laisser certains aspects de l'environnement virtuel évoluer de manière libre lorsqu'ils ne sont pas concernés par le scénario ou bien qu'aucune situation but particulière n'a été définie. De plus, l'utilisation d'ajustements pour contrôler des entités autonomes permet d'influencer non seulement les personnages virtuels mais également les utilisateurs, en utilisant ces ajustements pour modifier l'issue des comportements résultant de leurs actions.
La génération du scénario présentée dans ce chapitre ne permet cependant pas de contrôler d'autres propriétés du scénario en dehors de la situation finale. Le chapitre suivant porte donc sur la représentation et l'utilisation par DIRECTOR de différents objectifs scénaristiques, en particulier la représentation d'espaces de scénarios d'intérêt grâce au langage CAUSALITY-DL.

Chapitre 7

Représentation et prise en compte des objectifs scénaristiques

Sommaire

Afin de pouvoir exercer un contrôle plus important sur le scénario de l'environnement virtuel que de se limiter à la spécification de la situation finale, le processus de génération de scénarios de DIRECTOR, présenté dans le chapitre 6 peut être étendu pour prendre en compte un ensemble d'objectifs scénaristiques.
Nous avons distingué dans le chapitre 3 deux types d'objectifs scénaristiques complémentaires : les objectifs de bas niveau, permettant d'opérer un contrôle fin au niveau des situations, et les objectifs scénaristiques de haut niveau, permettant de contrôler des propriétés globales du scénario. Nous avons également distingué les objectifs scénaristiques statiques, qui s'ajoutent au contenu scénaristique pour cadrer un environnement virtuel existant dans le cas d'une scénarisation extrinsèque, des objectifs scénaristiques dynamiques, qui sont générés au cours d'une session d'utilisation de l'environnement virtuel et permettent d'adapter le scénario à l'activité de l'utilisateur. Nous estimons que

tous ces types d'objectifs sont importants et doivent être considérés pour la scénarisation d'un environnement virtuel, particulièrement dans un cadre de formation et d'aide à la décision.
Nous proposons de prendre en compte au niveau de la génération des scénarios par DIRECTOR à la fois des espaces de scénarios d'intérêt statiques, et des objectifs générés de manière dynamique au cours de la simulation par le moteur TAILOR, sous la forme de situations prescrites et proscrites et de contraintes sur les propriétés du scénario.
Dans ce chapitre, nous présenterons tout d'abord CAUSALITY-DL, le langage utilisé pour écrire des modèles de causalité qui définissent des espaces de scénarios d'intérêt, puis nous détaillerons la représentation des trames scénaristiques qui peuvent être extraites de ces modèles. Nous présenterons ensuite les deux types d'objectifs scénaristiques dynamiques pris en compte par DIRECTOR : les situations prescrites et proscrites, et les propriétés des scénarios. Enfin, la dernière section de ce chapitre porte sur la prise en compte de l'ensemble de ces éléments dans le processus de génération de scénario de DIRECTOR.

7.1 CAUSALITY-DL – Représentation d'espaces de scénarios d'intérêt

Le modèle de causalité décrit les événements considérés comme signifiants pour une utilisation donnée de l'environnement virtuel, et les liens de subsomption et de causalité qui les relient, qu'un événement permette de réaliser un autre événement ou inhibe au contraire cette réalisation. Pour le représenter, nous avons proposé le langage CAUSALITY-DL, qui fait suite aux réflexions sur les formalismes de représentation utilisés en analyse de risques réalisées précédemment par notre équipe de recherche, notamment sur les nœuds papillons [Camus, 2010] et leur traduction sous forme de réseaux bayésiens [Amokrane, 2010].

7.1.1 Approche

Besoins

Le langage de représentation des espaces de scénarios d'intérêt a, de même que les langages de représentation du contenu scénaristique, vocation à être utilisé par des experts du domaine non-informaticiens, en l'occurrence par des formateurs ou des experts de l'analyse de risques pour les cas d'application considérés. Il est donc nécessaire que ce langage soit **intelligible** et utilisable par ces experts.
Il est également nécessaire qu'il soit **interprétable**, afin de permettre, d'une part, de raisonner sur les événements du modèle à l'aide du moteur de planification, et d'autre part, d'évaluer l'activation ou la non-activation de chaque événement afin de situer de manière dynamique l'avancement de la simulation par rapport au modèle.
Ce langage doit aussi être **expressif**. Il doit tout d'abord permettre la représentation des relations de causalité entre événements, y compris les conjonctions et disjonctions pouvant exister au niveau de ces relations. Il doit également proposer plusieurs niveau d'abstraction, à la fois via la spécification de relations de subsomption entre événements et par l'utilisation de variables et de concepts plutôt que de références directes à des instances d'objets de l'environnement virtuel. Enfin, il doit permettre de représenter le modèle à différents niveaux de granularité, en spécifiant ou non les événements intermédiaires entre deux événements principaux.
Nous souhaitons également que ce langage soit **modulaire**, afin de permettre d'ajouter ou d'enlever facilement des parties du modèle pour pouvoir adapter l'environnement virtuel à de nouvelles utilisations sans avoir à réécrire un nouveau modèle à chaque fois.
Cette modularité, associée à la possibilité de représenter les modèles avec différents niveau de granularité et d'abstraction, donnent au système de scénarisation utilisant ce langage davantage d'**adaptabilité**, en regard des systèmes basés sur des graphes exhaustifs décrivant l'ensemble des scénarios

possibles. Il n'est pas question ici d'expliciter les séquences d'actions et de comportements à réaliser dans l'environnement virtuel, mais de spécifier un ensemble d'étapes d'intérêt pour une utilisation donnée.

Proposition

Nous avons choisi pour cela de proposer un formalisme sous forme de graphe inspiré de formalismes utilisés en analyse de risques, et en particulier de celui des nœuds papillons [Bernuchon et al., 2006]. Ce formalisme, CAUSALITY-DL, permet ainsi de représenter informatiquement les résultats des analyses de risques réalisées sur le terrain, jusqu'ici représentées à l'aide de formalismes graphiques.
Les figures 7.1 et 7.2 présentent un extrait des parties en amont et en aval de l'événement redouté central d'un nœud papillon réalisé dans le cadre d'une analyse de risques sur le cas d'application des dépôts pétroliers. Elles correspondent respectivement à un arbre de défaillance (causes) et à un arbre d'événements (conséquences) autour de l'événement correspondant au débordement d'une citerne par le dôme. Le rectangle rouge correspond à cet événement redouté central, tandis que les rectangles blancs correspondent aux événements menant à ou découlant de cet événement, et les rectangles noirs correspondent aux barrières permettant d'interrompre la propagation de la causalité d'un événement à un autre.

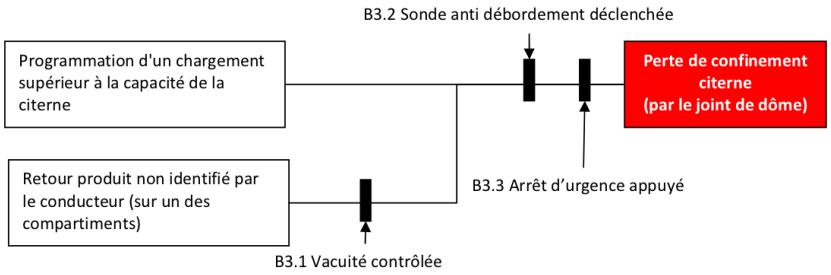


FIGURE 7.1 – Extrait d'un nœud papillon présentant les causes d'un débordement de citerne.

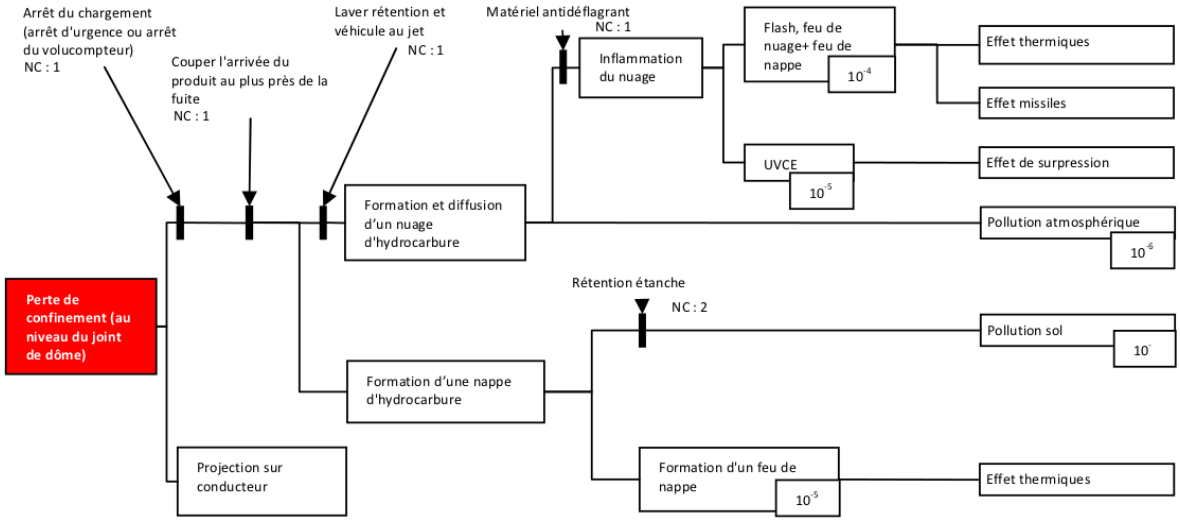


FIGURE 7.2 – Extrait d'un nœud papillon présentant les conséquences d'un débordement de citerne.

Contrairement aux nœuds papillons qui associent deux arbres autour d'un événement redouté central, CAUSALITY-DL repose sur un **graphe orienté acyclique**, afin de permettre la représentation des causes et des conséquences de plusieurs événements à la fois. Les liens de causalité qui lient les événements de ces graphes ont une valeur indicative, et ne correspondent pas à des états spécifiques du monde. Il est ainsi possible d'omettre dans le modèle certains événements des chaînes de causalité, afin de ne représenter que ceux qui sont signifiants par rapport à une utilisation donnée.
Les nœuds papillons sont un formalisme graphique et textuel, qu'il est nécessaire d'opérationnaliser. Pour cela, les nœuds du graphe reprennent les événements définis dans le modèle du domaine écrit en DOMAIN-DL. A l'instar des modèles représentés en DOMAIN-DL, les modèles représentés en CAUSALITY-DL possèdent une partie **statique** et une partie **dynamique** : les événements sont associés à un état d'activation dépendant de l'état courant du monde, et qui permet de suivre l'avancée dans l'environnement virtuel et de déterminer l'activabilité ou non d'un événement en fonction des barrières qui sont levées.

7.1.2 Description du langage

Les modèles de causalité représentés en CAUSALITY-DL sont des **graphes orientés acycliques** dont les nœuds se réfèrent à des **événements** représentés dans le modèle de domaine. Ces événements peuvent être liés par des relations de **subsomption** ou de **causalité**. Les relations de causalité sont également déterminées par des **portes logiques** indiquant la conjonction ou la disjonction, et sont conditionnées par la levée d'un ensemble de **barrières**, elles-aussi correspondant à des événements du modèle du domaine. Les modèles décrits en CAUSALITY-DL s'appuient sur les modèles représentés en DOMAIN-DL pour la spécification des événements et des liens de subsomption, et sont associés à une représentation graphique. Un extrait de modèle de causalité est présenté dans la figure 7.3.

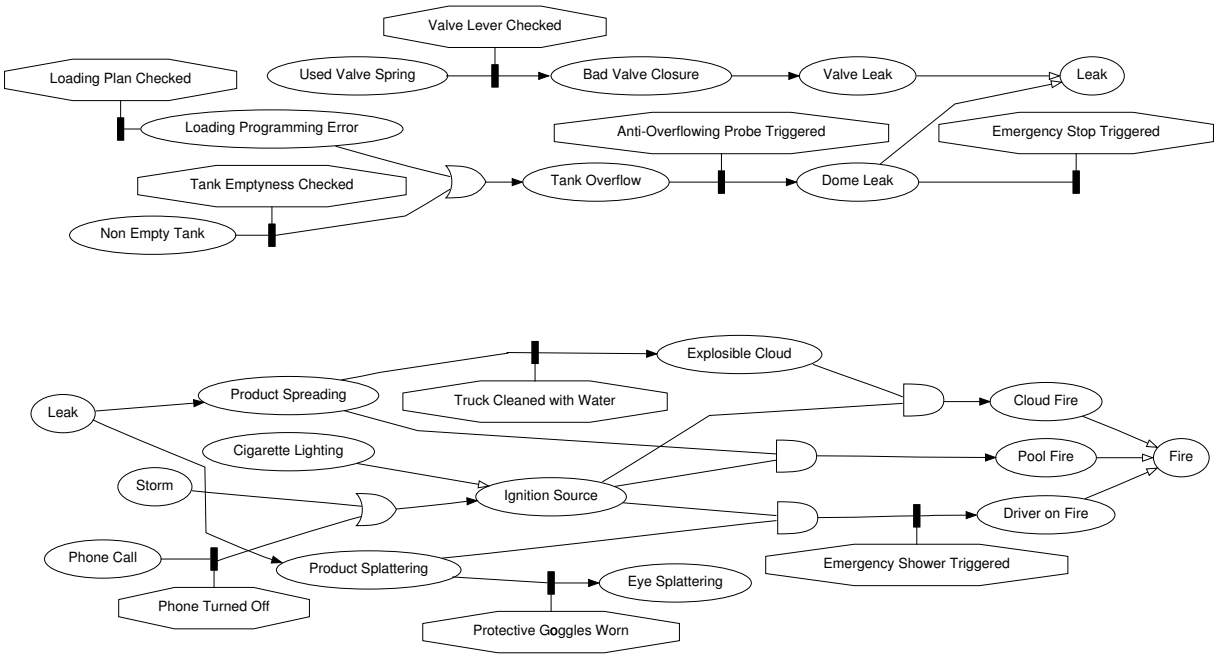


FIGURE 7.3 – Extrait d'un modèle de causalité coupé en deux au niveau de l'événement `Leak`

Les modèles sont décrits selon le méta-modèle présenté dans la figure 7.4.

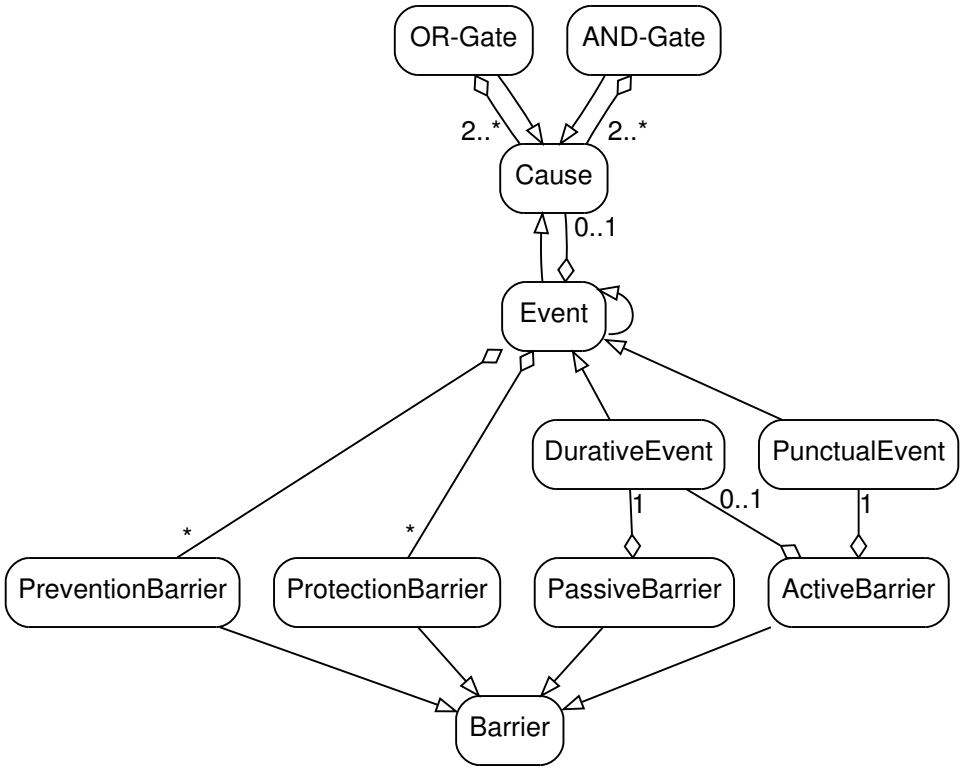


FIGURE 7.4 – Méta-modèle de CAUSALITY-DL

Evénements

Les éléments de base du modèle de causalité sont les événements sélectionnés à partir du modèle de domaine écrit en DOMAIN-DL. Ces événements peuvent donc, comme dans le modèle de domaine, être endogènes ou exogènes, être ponctuels ou duratifs, et correspondre ainsi à une situation donnée ou à un changement de situation.
Chaque événement est lié aux objets qu'il concerne au travers des relations présentes dans le modèle du domaine. Lorsqu'un événement est réalisé dans l'environnement virtuel, une instance du concept correspondant à cet événement est créée et ces relations sont instanciées pour lier cette dernière aux instances d'objets concernés.
Ces instances d'événements sont également dotées d'un état d'activation, permettant de connaître l'état de l'environnement virtuel par rapport au modèle de causalité. Les règles qui régissent la création et l'activation d'un événement dépendent du type d'événement :

- Les instances d'événements exogènes ponctuels sont créées et activées directement de manière externe (il s'agit des *happenings*).
- Les instances d'événements exogènes duratifs sont créées et activées (resp. désactivées) via les règles liées à l'activation des événements ponctuels correspondant à leur début (resp. à leur fin). Par exemple, une instance de l'événement `Rain` sera créée et activée suite au happening `RainStart`, et désactivée suite au happening `RainStop`.
- Les événements endogènes duratifs sont associés à des règles d'activation, comme celles présentées précédemment dans les tableaux 5.3 et 5.4. C'est via ces règles que la création, l'activation et la désactivation des instances de ces événements est réalisée.
- Deux cas de figure se posent pour les événements endogènes ponctuels : les événements ponctuels qui correspondent au début ou à la fin d'un événement duratif, et les événements ponctuels autonomes. Les premiers sont créés et activés via les mêmes règles que les événements duratifs auxquels ils sont associés. Les seconds sont associés à leurs propres règles, qui comparent l'état de l'environnement virtuel au pas de simulation courant et l'état au pas de simulation

précédent pour détecter les changements d'états ponctuels.

Relations de subsomption

Les relations de subsomption entre événements définies au niveau du modèle du domaine sont reprises dans le modèle de causalité. C'est le cas par exemple des relations qui lient les événements `Valve Leak` et `Dome Leak` à l'événement `Leak` dans l'exemple 7.3.
Un événement est ainsi considéré comme actif lorsque l'un des événements qu'il subsume est actif. De même, les relations entre l'événement subsumant et les différents objets concernés subsument des relations entre l'événement subsumé et les objets que celui-ci concerne.

Relations de causalité

Les événements du modèle sont liés par des relations de causalité, permettant d'indiquer les chaines d'événements et les combinaisons d'événements pouvant mener à un événement donné. Les conjonctions ou disjonctions au niveau de ces liens de causalité sont exprimées à l'aide de "portes logiques" `ET` et `OU`, à l'instar des arbres de défaillances.
Cependant, à l'inverse des arbres de défaillance, ces portes n'apparaissent qu'au niveau des liens de causalité, et non au niveau des liens de subsomption. En effet, du fait de la dimension temporelle des systèmes considérés, la présence d'un lien direct entre un événement A et un événement B n'implique pas pour autant qu'il s'agisse du même événement : par exemple, la relation entre un événement correspondant à la programmation d'un sur-remplissage d'une citerne et un événement correspondant au sur-remplissage effectif de celle-ci ne relève pas de la subsomption. Il est donc possible de considérer des chaines de causalité alternatives au niveau d'une porte `OU`. Dans le formalisme des arbres de défaillance, les portes `OU` sont au contraire nécessairement liées à des relations de subsomption : cela nécessite de faire figurer dans le modèle des nœuds correspondant à des événements identiques qui diffèrent uniquement par l'historique des événements y ayant mené. Cela nécessiterait par exemple de considérer les événements "sur-remplissage d'une citerne causé par la programmation d'une quantité supérieure à la capacité de la citerne sur une citerne vide" et "sur-remplissage d'une citerne causé par la programmation d'une quantité égale à la capacité de la citerne sur une citerne déjà pleine" comme étant deux événements différents.
Les relations de causalité sont également associées à un contexte permettant d'indiquer les correspondances entre les objets liés aux différents événements. Un exemple est présenté dans le tableau 7.1.

```
(?ev1 rdf:type :BadValveClosure)
(?ev1 :concerns-valve ?v1)
(?ev2 rdf:type :ValveLeak)
(?ev2 :concerns-valve ?v2)
equal(?v1, ?v2)
```

TABLE 7.1 – Correspondance entre les objets concernés par les événements `Bad Valve Closure` et `Valve Leak` contenue au niveau du lien de causalité entre ces événements.

Barrières

Chaque événement du graphe peut être associé à un ensemble de barrières, qui correspondent aux événements permettant d'interrompre la propagation de la causalité — dans le cas d'analyses de risques, ces événements résultent des systèmes de sécurité. En reprenant la typologie des barrières proposée dans [Miche et al., 2009], on considère que ces barrières peuvent être actives ou passives.

Les barrières passives correspondent à des événements duratifs de type maintien d'une situation, par exemple : "Port des Equipements de Protection Individuels". Les barrières actives correspondent à l'association d'un événement duratif optionnel se rapportant à la mise en place de la barrière, par exemple "Bon fonctionnement de l'arrêt d'urgence", et d'un événement ponctuel se rapportant à son activation, par exemple "Déclenchement de l'extincteur à eau automatique". Dans le cas des barrières humaines, cet événement ponctuel correspond à la réalisation d'une action, par exemple "Appui sur l'arrêt d'urgence".
Une même barrière peut être associée à un ou plusieurs événements. Pour chaque événement, la barrière peut être une barrière de prévention ou une barrière de protection. Les barrières de prévention apparaissent en amont de l'événement dans la représentation graphique du modèle et empêchent la réalisation de l'événement. Les barrières de protection apparaissent en aval dans le modèle et empêchent les conséquences de l'événement de déclencher à leur tour d'autres événements.
Les barrières ne peuvent apparaître qu'au niveau des relations de causalité, et non au niveau des relations de subsomption.

Propriétés

Les événements et les barrières présents dans le modèle de causalité sont également annotés avec un certain nombre de propriétés qualitatives. Celles-ci sont décrites en détail dans la section 7.4.

7.1.3 Bilan

Nous avons proposé le langage **CAUSALITY-DL**, qui permet de représenter des espaces de scénarios d'intérêt au travers de relations de subsomption et de relations de causalité liant des événements d'un modèle de domaine.
CAUSALITY-DL est inspiré du formalisme des nœuds papillons utilisé en analyse de risques, et est associé à une représentation graphique très proche de celle utilisée par les experts des risques, le rendant **intelligible** pour ces derniers. CAUSALITY-DL pourrait ainsi être utilisé directement par ces experts pour spécifier les relations entre événements.
Contrairement aux nœuds papillons, cependant, CAUSALITY-DL n'est pas uniquement graphique, mais est formalisé en termes d'événements liés au modèle du domaine représenté en DOMAIN-DL, et est donc lui aussi **interprétable**. De même, la spécification des correspondances entre variables concernées par les différents événements du modèle permet d'expliciter les connaissances tacites des experts qui conçoivent celui-ci, de sorte à pouvoir les opérationnaliser. Les modèles ainsi décrits sont utilisables à la fois pour du suivi de l'environnement virtuel, au travers des états d'activation des événements, et pour de la génération de scénarios, comme nous le verrons dans la suite de ce chapitre.
La possibilité d'écrire des modèles à différents niveau de granularité permet d'éviter d'avoir à spécifier des séquences exhaustives d'événements de bas niveau. CAUSALITY-DL permet ainsi d'écrire des modèles de manière **modulaire**, en ajoutant, modifiant ou supprimant des parties du graphe sans avoir à modifier les autres parties. Il est par exemple possible d'enlever ou de rajouter une chaine de causalité alternative menant à un événement selon qu'on souhaite ou non qu'elle puisse se produire dans l'environnement virtuel.
Enfin, CAUSALITY-DL est un langage **expressif** qui permet de représenter des chaines de causalité à différents niveaux de granularité et différents niveaux d'abstraction, au travers de relations de subsomption entre événements et de référence à des concepts d'objets du modèle du domaine. Le graphe n'est pas centré sur un événement particulier, et permet donc de représenter des modèles plus complexes que ceux qui peuvent être représentés avec les formalismes classiques utilisés en analyse des risques. A l'instar de ces modèles, l'utilisation de portes logiques au niveau des liens de causalité pour représenter la conjonction et la disjonction, et de barrières pour représenter l'inhibition, permet la

description de relations complexes entre événements.
Les modèles représentés en CAUSALITY-DL sont utilisés par le moteur DIRECTOR pour sélectionner des **trames scénaristiques**, qui correspondent à un parcours particulier dans le modèle de causalité menant à un événement but.

7.2 Trames scénaristiques

Une trame scénaristique (ou *plot*) est un ensemble de points clés (ou *plot points*) partiellement ordonnés, généré par DIRECTOR en vue de guider la création d'un scénario.

Trame scénaristique

Une trame scénaristique est définie comme un tuple $\langle P, L, p_F, C\rangle$ tel que P est un ensemble de points clés, L est un ensemble de liens de causalité entre ces points, tel que tout élément $l \in L$ est un couple $\langle p_1, p_2\rangle$ avec $p_1, p_2 \in P$, p_F le point clé final de la trame scénaristique, tel que $\forall \langle p_1, p_2\rangle \in L, p_1 \neq p_F$, et C le contexte de la trame scénaristique, c'est à dire un ensemble de contraintes de correspondance sur les variables des points clés.

Un point clé correspond à une situation non instanciée, c'est à dire à un ensemble d'assertions sur l'état du monde référençant des variables, et peut être associé à un ensemble de barrières, elles aussi correspondant à des assertions sur le monde.

Point clé

Un point clé est défini comme un tuple $\langle A, V, B\rangle$ tel que A est un ensemble d'assertions, V est un ensemble de variables tel que tout élément $v \in V$ est un couple $\langle id, c\rangle$ où id est un identifiant et c un concept du modèle du domaine, et B est un ensemble de barrières tel que tout élément $b \in B$ est un couple $\langle A_b, V_b\rangle$.

Chaque point clé correspond à un événement du modèle de causalité. Les assertions contenues par un point clé dépendent du type d'événement considéré. Pour un événement endogène, il s'agira des préconditions des règles d'activation de l'événement, ou de désactivation s'il s'agit d'un événement ponctuel marquant la fin d'un événement duratif. Pour un événement exogène, il s'agira des effets du ou des comportements directement déclenchés par l'événement.

La figure 7.5 présente un exemple de trame scénaristique, et le tableau 7.2 détaille un des points clé de cette trame.

De même, les barrières associées aux points clés correspondent à des barrières du modèle de causalité. Les barrières passives, associées à un événement duratif, contiennent ainsi les assertions correspondant aux préconditions de la règle d'activation de l'événement. Les barrières actives, qui sont à la fois associées à un événement duratif et à un événement ponctuel correspondant à l'activation de la barrière, contiennent à la fois des assertions correspondant aux préconditions de la règle d'activation de l'événement duratif, et des assertions témoignant de l'activation préalable du comportement déclenché par l'événement ponctuel — ou par l'action désignée par cet événement — par exemple les assertions `(has-been-activated PushButton)` et `(old-params-PushButton ?eb - EmergencyStopButton)` témoigneront de l'activation antérieure du comportement déclenché par l'action d'appui sur l'arrêt d'urgence, activation représentée pour le moteur de planification par l'assertion `(activate-behaviour PushButton)`.
Les barrières associées à un point clé correspondent aux barrières qu'il est nécessaire de lever pour

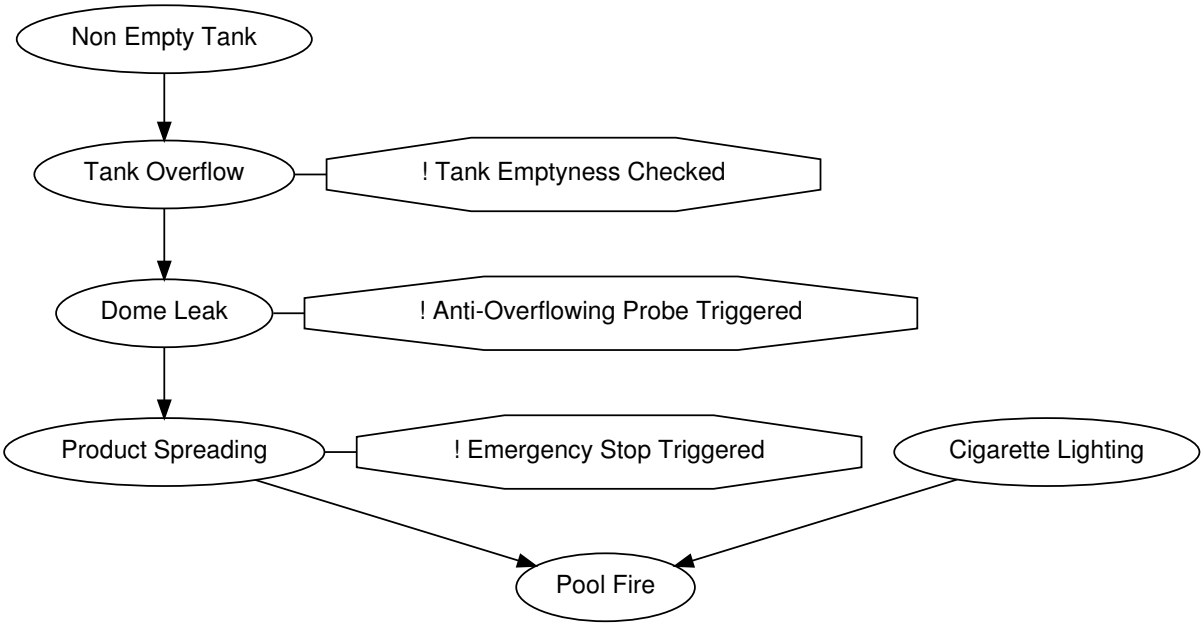


FIGURE 7.5 – Exemple de trame scénaristique générée par DIRECTOR

Plot Point		`Tank Overflow`	
Variables		`?tank - Tank` `?quantity ?capacity - Quantity`	

Assertions		`(fill-state-filling ?tank ?quantity)`
		`(fill-state-capacity ?tank ?capacity)`
		`(superior ?quantity ?capacity)`	

Barriers	`Tank Emptiness Checked`	**Variables**	`?tank - Tank` `?a - Agent`
		Assertions	`(has-been-activated CheckTankEmptiness)` `(old-params-CheckTankEmptiness ?a ?tank)`

Context	`(fill-state-capacity TankOverflow : ?tank TankOverflow : ?capacity)`
	`(superior TankOverflow : ?quantity TankOverflow : ?capacity)`
	`TankOverflow : ?tank = TankEmptinessChecked : ?tank`

TABLE 7.2 – Exemple de point clé correspondant à l'événement endogène duratif `Tank Overflow`.

pouvoir atteindre le point clé en question. De fait, lors de la génération des trames scénaristiques, les barrières de protection associées à un événement sont transformées en barrières de prévention des événements qui le suivent directement, et ainsi associées au point clé correspondant. C'est le cas par exemple de la barrière `Tank Emptiness Checked` dans la figure 7.5. De même, les barrières associées à un événement subsumant un autre événement sont répercutées sur ce dernier.
Le contexte de la trame scénaristique est créé à partir des contraintes de correspondance spécifiées sur les liens de causalité du modèle et des attributs déclarés comme statiques dans le modèle du domaine — dans l'exemple 7.2, il s'agit de la capacité de la cuve et de la relation de supériorité entre les deux quantités.
Pour sélectionner une trame scénaristique à suivre, DIRECTOR se base sur les objectifs scénaristiques dynamiques définis par le moteur TAILOR : les situations prescrites et proscrites, et les contraintes sur les propriétés du scénario.

7.3 Situations prescrites et proscrites

Le premier type d'objectifs dynamiques utilisé pour contrôler la génération des scénarios par DIRECTOR est la spécification de situations prescrites ou proscrites dans le scénario.
Il s'agit d'un contrôle de bas niveau, permettant de contrôler précisément quelles situations seront rencontrées par l'utilisateur ; il peut s'agir de spécifier des situations d'apprentissage mettant en jeu des connaissances ou des compétences particulières, ou encore de tester la confiance dans les barrières d'un système par rapport à un accident particulier.
TAILOR envoie ainsi à DIRECTOR un ensemble de contraintes de situations définies comme suit :

Contrainte de situation

Une contrainte de situation C_{sit} est un couple $\langle e, d \rangle$ tel que e est un événement appartenant au modèle de causalité et $d \in [-1;1]$ est la désirabilité de cet événement.

Cet ensemble contient un élément particulier $\langle e_F, 1 \rangle$ qui correspond à la situation finale désirée pour le scénario. Cette contrainte est une contrainte locale, qui porte sur la fin du scénario. Le reste des contraintes sont des contraintes globales, qui doivent être respectées sur l'ensemble du scénario. Les situations prescrites ont une désirabilité d positive et correspondent aux événements que l'on souhaite voir apparaître dans le scénario. Les situations proscrites ont une désirabilité d négative et correspondent aux événements à éviter.
Le modèle de causalité reprenant les relations de subsomption entre événements définies dans le modèle du domaine, et donc l'ensemble de leur hiérarchie, il est possible de définir des contraintes de haut niveau sur les situations, par exemple en spécifiant une désirabilité sur un événement "Accident" plutôt que de donner un événement accidentel particulier. Cela est particulièrement utile lorsqu'on s'intéresse à une situation d'apprentissage dans l'optique de mettre en jeu des compétences particulières de l'utilisateur, mais que la façon d'arriver à cette situation est négligeable. Par exemple, dans le cas d'application des assistantes de vie, on pourra chercher à amener l'apprenant à effectuer une manœuvre de Heimlich sur un enfant, et donc viser une situation d'étouffement. Selon les cas, la cause de l'étouffement ("Etouffement avec des billes" ou "Etouffement avec des cailloux") peut ne pas avoir d'incidence, et donc il sera possible de spécifier l'objectif à plus haut niveau ("Etouffement") afin de laisser à DIRECTOR davantage de marge de manœuvre dans la génération et la réalisation du scénario.
Le score d'une trame scénaristique par rapport aux contraintes de situation est calculé de la manière suivante : pour une contrainte donnée, s'il existe un point clé P_T de la trame scénaristique qui correspond à l'événement prescrit/proscrit ou à un événement qui est subsumé par celui-ci, alors on ajoute au score total la désirabilité de l'événement, sinon on lui retire.

Calcul du score d'une trame scénaristique pour les contraintes de situations

$$f(e,T) = \begin{cases} 1 & \text{if } \exists p \in P_T : \mathrm{id}(p) = \mathrm{id}(e) \\ 1 & \text{if } \exists p \in P_T, \exists e' \in E : \mathrm{id}(p) = \mathrm{id}(e') \wedge \text{is-a}(e', e) \\ -1 & \text{otherwise} \end{cases}$$

$$S_{sit}(T) = \sum_{\langle e,d \rangle \in C_{sit}} f(e,T) * d$$

7.4 Propriétés des scénarios

En parallèle des contraintes de bas niveau sur les situations, DIRECTOR peut également prendre en compte des contraintes de haut niveau, portant sur des propriétés globales du scénario.
Les éléments du modèle de causalité — événements, et, dans certains cas, barrières — sont annotés avec des valeurs individuelles pour ces propriétés. Plutôt que d'opter pour une approche quantitative, il s'agira de qualifier ces éléments : comme il a été noté lors de l'état de l'art sur les formalismes de représentation utilisés en analyse de risques, il est délicat pour les experts de fournir des chiffres précis pour quantifier la fréquence d'occurrence ou encore la gravité d'événements particuliers. La génération de scénarios dans un but de formation ne nécessitant pas une telle précision, ces valeurs seront déterminées à partir d'un ensemble d'échelles qualitatives, détaillées dans chacune des parties suivantes.
Les connaissances permettant d'évaluer ces propriétés pour l'ensemble d'une trame scénaristique à partir des propriétés individuelles sont indépendantes du domaine, et encodées sous la forme de fonctions d'évaluation.
Les propriétés prises en compte par DIRECTOR sont au nombre de trois : la **complexité** de la trame scénaristique et des événements qu'elle contient, la **gravité** des conséquences des événements de la trame scénaristique, et la **crédibilité** de la trame scénaristique.

7.4.1 Complexité

La complexité en informatique met en jeu de nombreux aspects. Nous ne nous intéressons pas ici à la complexité d'un scénario en termes algorithmiques, c'est à dire au temps et aux ressources nécessaires pour générer et exécuter le scénario, mais à la complexité ressentie par l'utilisateur de la simulation, que l'on assimile d'une part à la charge cognitive nécessaire pour appréhender la trame scénaristique et les liens de causalité entre les différents événements, et d'autre part à l'expertise nécessaire pour réagir face à un événement donné.
Il peut être intéressant, particulièrement dans le cas d'environnements virtuels pour l'apprentissage, de contrôler cette complexité, par exemple en la faisant croître au fur et à mesure des sessions d'utilisation.
Nous définissons ainsi quatre niveaux de complexité qualitatifs, présentés dans le tableau 7.3. Chaque niveau est associé à une valeur entre 0 et 3.

Niveau de complexité	Description	Valeur
élémentaire	Aucune connaissance ou compétence particulière n'est nécessaire pour réagir face à l'événement. Aucun autre facteur n'intervient en parallèle	0
simple	Des connaissances et compétences limitées sont suffisantes pour réagir face à l'événement. Un nombre limité de facteurs interviennent en parallèle.	1
moyen	Certaines connaissances et compétences dans le domaine sont nécessaires pour réagir face à l'événement. Plusieurs facteurs interviennent en parallèle	2
élevé	Des connaissances et compétences expertes dans le domaine sont nécessaires pour réagir face à l'événement De nombreux facteurs interviennent en parallèle.	3

TABLE 7.3 – Niveaux de complexité d'un événement ou d'une trame scénaristique

Chaque événement est associé à un niveau de complexité, qui reflète le niveau d'expertise qui

serait nécessaire à une personne, dans la situation réelle, pour remettre le système dans un mode de fonctionnement nominal, ou de manière générale, pour exécuter la suite de sa procédure à partir de cet événement.
La complexité qui est calculée à partir des complexités des événements individuels correspond à la complexité interactionnelle de la trame scénaristique. La complexité globale correspond à la moyenne de la complexité interactionnelle et de la complexité temporelle.

Contrainte de complexité

Une contrainte de complexité C_{comp} est constituée d'un couple $\langle v_{comp}, d_{comp} \rangle$ tel que $v_{comp} \in [0,3]$ est un niveau de complexité et $d_{comp} \in [0,1]$ est la désirabilité de cette complexité.

Les niveaux de complexité d'une trame scénaristique et le score d'une trame scénaristique par rapport à une contrainte de complexité donnée sont calculés de la manière suivante :
La complexité temporelle $\text{complexity}_{temp}(T)$ est calculée à partir du nombre de branches parallèles et de barrières présentes dans la trame scénaristique, représenté par la fonction $n(T)$.
La complexité interactionnelle $\text{complexity}_{int}(T)$ correspond à la complexité maximale des points clés p de la trame scénaristique.
Le niveau de complexité d'une trame scénaristique $\text{complexity}(T)$ correspond à la moyenne de sa complexité temporelle $\text{complexity}_{temp}(T)$ et de sa complexité interactionnelle $\text{complexity}_{int}(T)$, ramenée à l'entier directement inférieur.
Le score de complexité $S_{comp}(T)$ équivaut à la différence entre le niveau de complexité de la trame scénaristique $\text{complexity}(T)$ et le niveau de complexité désiré v_{comp}, normalisée et multipliée par la désirabilité d_{comp}.

Calcul du score d'une trame scénaristique pour la contrainte de complexité

$$\text{complexity}_{temp}(T) = \begin{cases} 0 & \text{if } n(T) = 0 \\ 1 & \text{if } 1 \leqslant n(T) \leqslant 2 \\ 2 & \text{if } 3 \leqslant n(T) \leqslant 7 \\ 3 & \text{if } n(T) \geqslant 8 \end{cases}$$

$$\text{complexity}_{int}(T) = \max_{p \in P_T}(\text{complexity}(p))$$

$$\text{complexity}(T) = \left\lfloor \frac{\text{complexity}_{int}(T) + \text{complexity}_{temp}(T)}{2} + 0,5 \right\rfloor$$

$$S_{comp}(T) = \frac{|3 - \text{complexity}(T) - v_{comp}|}{3} * d_{comp}$$

7.4.2 Gravité

La gravité d'un événement correspond à une évaluation subjective de ses conséquences négatives sur le système. Il s'agira ici de considérer la gravité selon le point de vue de l'utilisateur de la simulation.
Là encore, il est intéressant de pouvoir contrôler cette propriété, particulièrement dans un cadre d'environnements virtuels pour la formation : on cherchera alors à minimiser la gravité des scénarios pour des apprenants novices, de sorte à ne pas les choquer, mais on pourra augmenter la gravité pour des apprenants experts afin de leur faire prendre conscience des conséquences possibles de leurs erreurs.

Pour définir ces niveaux de gravité, nous reprenons l'échelle proposée dans la norme de sécurité CENELEC EN 50126, reproduite dans [Belmonte, 2008]. Chacun de ces niveaux est associé à une valeur entre 0 et 4. Le tableau 7.4 présente les différents niveaux de gravité considérés.

Niveau de gravité	Description	Valeur
nul	Aucun dommage.	0
insignifiant	Dommages mineurs pour un système ou sous-système. Eventuellement une personne blessée.	1
marginal	Dommages graves pour un ou plusieurs systèmes ou sous-systèmes. Blessures légères et/ou menace pour l'environnement.	2
critique	Un mort et/ou une personne grièvement blessée et/ou des dommages graves pour l'environnement. Perte d'un système ou sous-système important.	3
catastrophique	Des morts et/ou plusieurs personnes gravement blessées et/ou des dommages majeurs pour l'environnement.	4

TABLE 7.4 – Niveaux de gravité d'un événement ou d'une trame scénaristique

La gravité d'une trame scénaristique correspond à la gravité la plus élevée parmi les événements qui la composent. La gravité souhaitée est spécifiée par le biais de contraintes de gravité :

Contrainte de gravité

Une contrainte de gravité C_{grav} est constituée d'un couple $\langle v_{grav}, d_{grav} \rangle$ tel que $v_{grav} \in [0,4]$ est un niveau de gravité et $d_{grav} \in [0,1]$ est la désirabilité de cette gravité.

Le niveau de gravité d'une trame scénaristique et le score d'une trame par rapport à une contrainte de gravité donnée sont calculés de la manière suivante :
Le niveau de gravité d'une trame scénaristique gravity(T) correspond à la gravité maximale des points clés p de la trame scénaristique.
Le score de gravité $S_{grav}(T)$ équivaut à la différence entre le niveau de gravité de la trame scénaristique gravity(T) et le niveau de gravité désiré v_{grav}, normalisée et multipliée par la désirabilité d_{grav}.

Calcul du score d'une trame scénaristique pour la contrainte de gravité

$$\text{gravity}(T) = \max_{p \in P_T}(\text{gravity}(p))$$

$$S_{grav}(T) = \frac{4 - |\text{gravity}(T) - v_{grav}|}{4} * d_{grav}$$

7.4.3 Crédibilité

La crédibilité d'un scénario est fortement liée à sa probabilité d'occurrence : moins un scénario sera probable, moins il sera crédible. Il n'est cependant pas question ici de demander aux experts de quantifier précisément les fréquences d'occurrence des événements et de réaliser des calculs de probabilité, comme c'est le cas en analyse de risques.
Plutôt que d'utiliser des probabilités d'occurrence, nous définissons ici également une échelle à partir

des plages de fréquences d'occurrence subjectives proposées dans la norme de sécurité CENELEC EN 50126 et reproduites dans [Belmonte, 2008]. Ces niveaux peuvent se rapporter à la fréquence d'occurrence d'un événement du modèle de causalité, ou à celle de la levée d'une barrière. Chaque niveau est associé à une valeur allant de 0 à 5. Le tableau 7.5 présente les différents niveaux de fréquence considérés.

Niveau de fréquence	Description	Valeur
fréquent	Susceptible de se produire fréquemment. La situation est continuellement présente. La barrière est continuellement levée.	5
probable	Peut survenir à plusieurs reprises. On peut s'attendre à ce que la situation survienne ou que la barrière soit levée.	4
occasionnel	Peut survenir de temps en temps. On peut s'attendre à ce que la situation survienne ou que la barrière soit levée.	3
rare	Susceptible de se produire à un moment donné du cycle de vie du système. On peut raisonnablement s'attendre à ce que la situation se produise ou que la barrière soit levée.	2
improbable	Peu susceptible de se produire, mais possible. On peut supposer que la situation peut exceptionnellement se produire ou la barrière exceptionnellement être levée.	1
invraisemblable	Extrêmement improbable. On peut supposer que la situation ne se produira pas et que la barrière ne sera jamais levée.	0

TABLE 7.5 – Niveaux de fréquence d'un événement ou de la levée d'une barrière

Contrainte de crédibilité

Une contrainte de crédibilité C_{cred} est constituée d'un couple $\langle v_{cred}, d_{cred} \rangle$ tel que $v_{cred} \in [0,5]$ est un niveau de crédibilité et $d_{cred} \in [0,1]$ est la désirabilité de cette crédibilité.

Le niveau de crédibilité d'une trame scénaristique believability(T) est ainsi évalué comme étant le niveau de crédibilité minimal des points clés qu'il contient (believability(p)) et des levées de barrières nécessaires à la réalisation des événements correspondant à ces points clés (believability(b)). Le score de crédibilité $S_{bel}(T)$ par rapport à une contrainte de crédibilité donnée équivaut à la différence entre le niveau de crédibilité de la trame scénaristique believability(T) et le niveau de crédibilité désiré v_{bel}, normalisée et multipliée par la désirabilité d_{bel}.

Calcul du score d'une trame scénaristique pour la contrainte de crédibilité

$$\text{believability}(T) = \min(\min_{p \in P_T}(\text{believability}(p)), \min_{b \in B_T}(\text{believability}(b)))$$

$$S_{bel}(T) = \frac{5 - |\text{believability}(T) - v_{bel}|}{5} * d_{bel}$$

7.5 Prise en compte des objectifs scénaristiques par DIRECTOR

Pour prendre en compte ces objectifs scénaristiques dans la génération du scénario par DIRECTOR, celle-ci est divisée en trois phases :

- la **sélection** de la trame narrative,
- l'**instanciation** de la trame narrative sélectionnée,
- et la **complétion** du scénario.

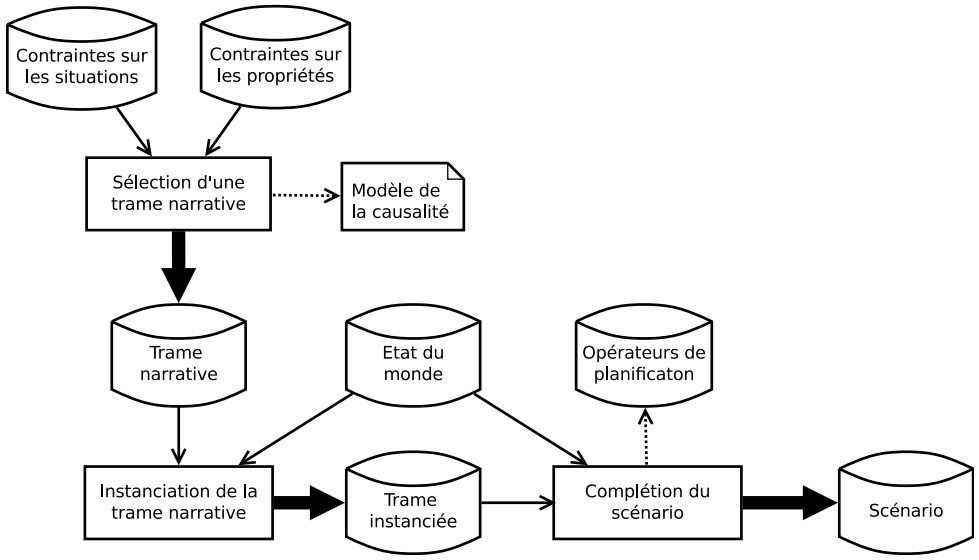


FIGURE 7.6 – Processus de génération des scénarios en trois phases

Ce processus de génération est représenté dans la figure 7.6.

7.5.1 Sélection de la trame scénaristique

Dans un premier temps, DIRECTOR se base sur le modèle de causalité représenté en CAUSALITY-DL pour générer un ensemble de trames scénaristiques. Une trame scénaristique correspond à un parcours particulier dans le modèle de causalité.
Pour générer celles-ci, le graphe de causalité CG est divisé en plusieurs graphes alternatifs au niveau de chaque porte `OU` et de chaque relation de subsomption en partant de la situation but S_G, selon l'algorithme 2, et ce jusqu'à ce que chaque graphe ne contienne plus que des portes `ET`. Une trame scénaristique est ensuite créée à partir de chaque graphe.

Le score de chaque trame scénaristique est évalué par rapport aux contraintes de situation et aux contraintes sur les propriétés selon la formule suivante :

Calcul du score total d'une trame scénaristique

$$S_{total}(T) = S_{sit}(T) + S_{comp}(T) + S_{grav}(T) + S_{bel}(T)$$

Une trame scénaristique est ensuite sélectionnée au hasard parmi les trames ayant obtenu le plus haut score.

Algorithm 2 Algorithme de génération des trames scénaristiques

```
function GENERATEPLOTS(CG, S_G)
    delete nodes after S_G in CG
    L ← {CG}
    while L contains decomposable graph G do
        if G contains "OR" nodes or "is-a" links then
            repeat N ← breadth-first search of G from S_G
            until N="OR" or (l ∈ in-links(N) and l ="is-a")
            if N ="OR" then
                for all l ∈ in-links(N) do
                    G' ← G
                    for all l' ∈ out-links(N) do
                        add link⟨source(l), target(l')⟩ to G'
                    delete in-links(N), out-links(N) and Nfrom G'
                    delete nodes not connected to S_G in G'
                    L ← L ∪ {G'}
            else
                for all l ∈ in-links(N) do
                    G' ← G
                    delete in-links(N)except l from G'
                    delete nodes not connected to S_G in G'
                    if l ="is-a" then merge N and source(l) in G'
                    L ← L ∪ {G'}
```

7.5.2 Instanciation de la trame scénaristique

La trame scénaristique sélectionnée est ensuite instanciée à partir des instances d'objets présentes dans l'état du monde. Pour cela, chacune des variables présentes dans les situations et les barrières est associée à une instance du concept correspondant. A chaque association, les contraintes de correspondance spécifiées dans le contexte de la trame sont appliquées pour associer chacune des variables correspondantes à la même instance, et pour appliquer les liens statiques définis entre les objets dans l'état initial du monde.
Un exemple de trame scénaristique instanciée est présentée dans la figure 7.7.

7.5.3 Complétion du scénario

La trame scénaristique instanciée est ensuite complétée à l'aide d'un moteur de planification. L'algorithme utilisé (voir algorithme 3) étend l'algorithme **Decomposition Search Control** proposé par [Porteous and Cavazza, 2009], en ajoutant la prise en compte des situations proscrites, ici les barrières liées aux situations. L'algorithme DSC est inspiré du principe des *landmarks*. Il repose sur l'utilisation de buts disjonctifs permettant au moteur de planification de trouver de lui-même quel point clé valider en premier parmi ceux qui se trouvent en amont d'un arbre de contraintes.
Notre algorithme prend en entrée une trame scénaristique T, un ensemble d'opérateurs de planification A, un ensemble d'objets O, et une situation initiale S_I. Un but disjonctif S_G est créé à partir des situations feuilles de la trame scénaristique — les points clés ne possédant pas de lien entrant —, puis on ajoute à ce but la non réalisation de l'ensemble des barrières de cette trame. Ce but est donné en entrée à un moteur de planification avec la situation de départ, l'ensemble des opérateurs et l'ensemble des objets. Si un plan est trouvé, alors le point clé ainsi validé est enlevé de la trame scénaristique, de même que les barrières qui lui sont associées, et le plan est ajouté au scénario global. L'algorithme itère ainsi jusqu'à ce que tous les points clés de la trame scénaristique aient été validés.

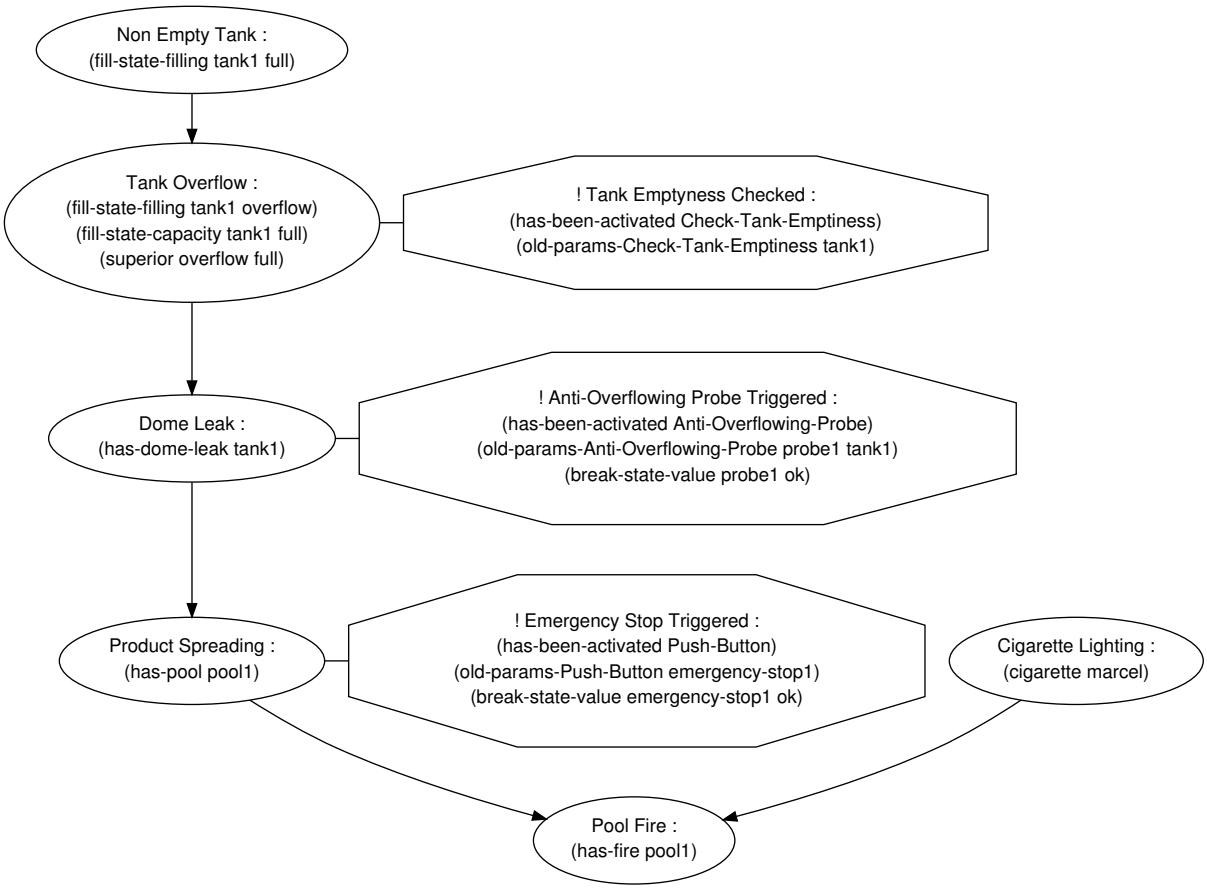


FIGURE 7.7 – Exemple de trame scénaristique instanciée

Algorithm 3 Algorithme de planification étendant le *Decomposition Search Control* [Porteous and Cavazza, 2009]

function PLANSCENARIO(T, A, O, S_I)
  $S \leftarrow S_I$
  while $T \neq \emptyset$ **do**
    for all L in leaf nodes of T **do**
      $S_G \leftarrow S_G \vee L$
    for all B in barriers of T **do**
      $S_G \leftarrow S_G \wedge \neg B$
    $P' \leftarrow$PLANNER(S_G, A, O, S)
    if not P' **then** fail
    else
      $P \leftarrow P \circ P'$
      $S \leftarrow$result of executing P' in S
      for all L in leaf nodes of T **do**
        if L is reached by P **then**
          remove L and every B of L from T

A la différence de [Porteous and Cavazza, 2009], les situations buts ne sont pas ici des faits uniques, mais des ensembles de faits qui doivent être vérifiés simultanément. Il n'est donc pas question de retirer certains faits des situations partiellement vérifiées par le plan, puisque ces faits pourraient être

annulés par la suite. De plus, la trame scénaristique peut ici être un graphe plutôt qu'un simple arbre. Un exemple de scénario ainsi complété est présenté dans la figure 7.8.

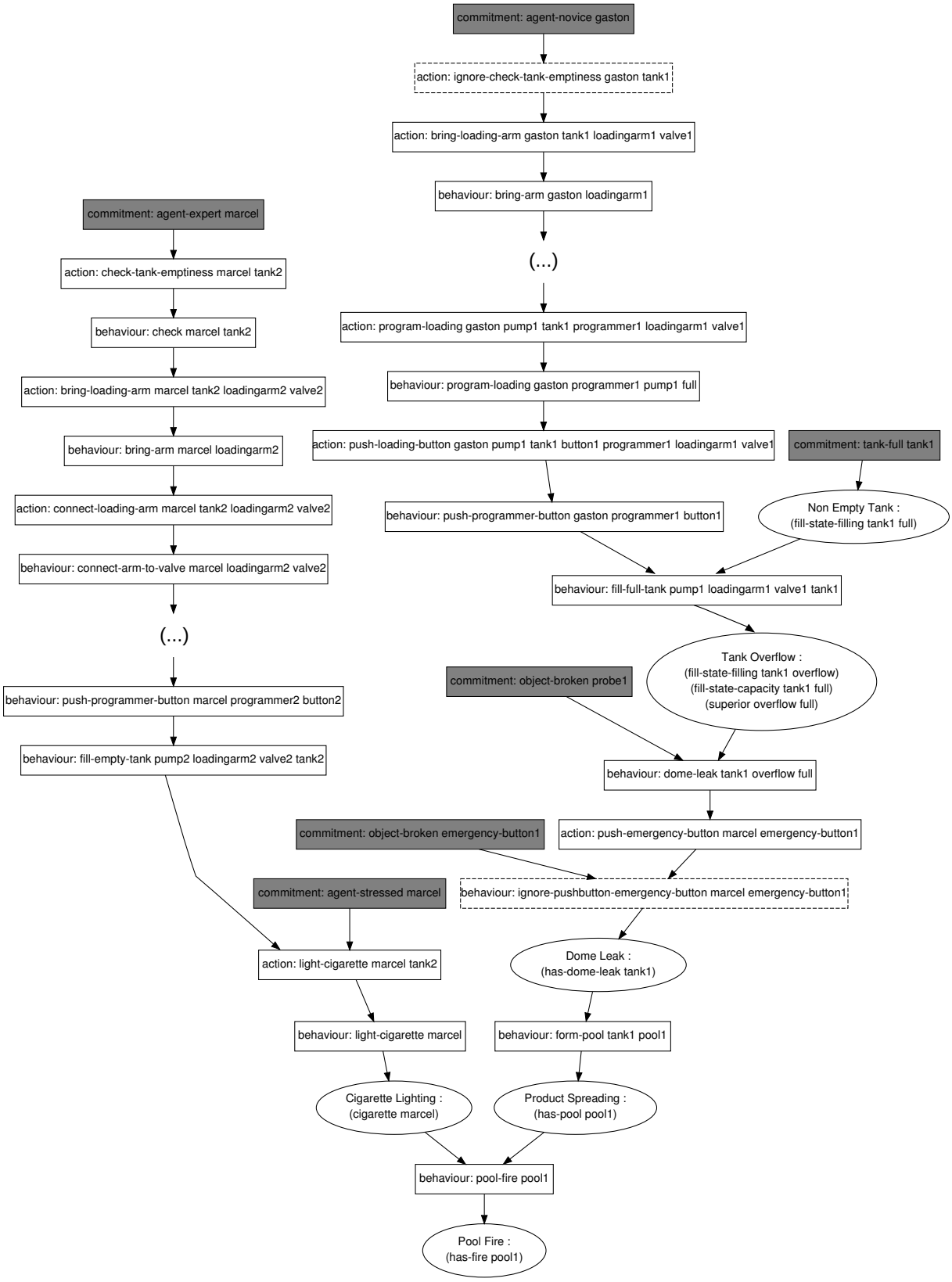


FIGURE 7.8 – Exemple de scénario généré à partir d'une trame scénaristique

7.6 Bilan

Nous avons présenté dans ce chapitre les objectifs scénaristiques pouvant être pris en compte par DIRECTOR durant le processus de génération des scénarios, au travers de la génération et la sélection de trames scénaristiques qui sont ensuite utilisées comme des points clés par le moteur de planification.
Le langage CAUSALITY-DL permet de représenter des objectifs scénaristiques statiques, sous la forme d'espaces de scénarios d'intérêt. Il tire ses origines de formalismes utilisés en analyses de risques, mais les connaissances qu'il permet de représenter sont suffisamment abstraites — relations de causalité, héritage, ou encore inhibition — pour qu'il puisse être étendu à d'autres domaines, comme la représentation de structures narratives classiques.
En plus des connaissances statiques sur le domaine contenues dans le modèle de causalité, DIRECTOR prend en compte deux sortes d'objectifs scénaristiques dynamiques, qui permettent d'adapter les scénarios générés en fonction des besoins des différentes sessions d'utilisation de l'environnement virtuel. Le premier type d'objectifs dynamiques est la spécification de situations prescrites ou proscrites, qui peuvent induire un contrôle précis des événements pouvant apparaitre dans le scénario, mais qui peuvent aussi mener à un contrôle de plus haut niveau via la désignation de situations abstraites présentes dans le modèle de causalité et le modèle du domaine grâce aux relations de subsomption. Le deuxième type d'objectifs dynamiques concerne des propriétés de haut niveau et indépendantes du domaine, calculées à l'aide de fonctions d'évaluation qui se basent sur des annotations des événements individuels du modèle de causalité. Ces contraintes sur les propriétés permettent d'opérer un contrôle plus souple, notamment lorsqu'elles sont associées à des contraintes de haut niveau sur les situations. On peut ainsi déclarer comme objectif la génération d'un scénario menant vers un Accident, et associé à une forte gravité.

Troisième partie

Implémentation et résultats

Chapitre 8

Implémentation

Sommaire

Le moteur DIRECTOR a fait l'objet d'une implémentation, qui a rencontré des problématiques particulières du fait de l'utilisation d'un moteur de planification. Nous présentons dans ce chapitre les choix liés à la planification, ainsi que l'architecture logicielle de DIRECTOR.

8.1 Choix liés à la planification

Le processus de création d'un scénario par DIRECTOR repose sur la génération d'un domaine de planification à partir des modèles de domaine et d'activité, sur la génération d'un problème de planification à partir des objectifs scénaristiques, et sur l'utilisation d'un moteur de planification afin de générer les plans qui correspondront aux étapes du scénario. Pour implémenter ce processus, nous nous sommes basés sur l'utilisation du langage PDDL et d'un moteur de planification capable d'interpréter ce langage.

8.1.1 Représentation en PDDL

Les connaissances utilisées pour la planification des scénarios sont représentées en PDDL. PDDL (Planning Domain Definition Language) est un langage de représentation de tâches de planification basé sur le Lisp. Ce langage a été proposé dans le cadre de la compétition de planification IPC (International Planning Competition) dans le but de permettre l'échange des domaines et la comparaison des résultats des différents moteurs.
Il en existe à ce jour différentes versions :

– **PDDL1.2** [McDermott et al., 1998] , la première version de PDDL, permet de représenter des problèmes de planification classiques, de manière très proche du STRIPS [Fikes and Nilsson, 1971] ;

- **PDDL2.1** [Fox and Long, 2003] étend cette première version avec la représentation des problèmes de planification numérique et temporelle ;
- **PDDL2.2** [Edelkamp and Hoffmann, 2004] introduit les axiomes, ou prédicats dérivés, qui permettent d'inférer directement certains faits à partir d'autres faits ;
- **PDDL3.0** [Gerevini and Long, 2005] y ajoute la représentation de préférences, et de contraintes sur l'occurrence ou l'ordonnancement de certaines situations ;
- **PDDL3.1** [Kovacs, 2011], la dernière version en date, permet de considérer des fonctions pouvant renvoyer n'importe quel type d'objet, et non plus seulement des booléens ou des nombres.

En PDDL, une tâche de planification est composée d'un **domaine** et d'un **problème**.
Le domaine est constitué de :

- la définition des **types d'objets** considérés,
- l'ensemble des **constantes**,
- la définition des **prédicats**,
- l'ensemble des **opérateurs de planification** (aussi appelés actions PDDL). Dans notre cas, il s'agit des cinq types d'opérateurs : actions, comportements, commitments, happenings et contraintes d'occurrence.

Le problème est constitué de :

- l'ensemble des **objets**,
- les faits constituant l'**état initial**,
- la spécification du **but**.

Le langage PDDL permet de représenter de nombreux types de problèmes, mais tous les types de problèmes ne peuvent pas être traités par tous les moteurs de planification. PDDL définit donc un certain nombre de prérequis, qui doivent être déclarés dans le domaine pour s'assurer que le moteur utilisé a la capacité de traiter la tâche de planification.
Les tâches de planification créées par DIRECTOR possèdent les prérequis suivants :

- `strips` : les opérateurs de planification doivent pouvoir ajouter et supprimer des faits à l'état du monde ;
- `typing` : les objets doivent pouvoir être typés, de manière à reproduire la hiérarchie de concepts définie dans le modèle du domaine, et de permettre ainsi d'appliquer les comportements sur les objets du type associé ;
- `disjunctive-preconditions` : le moteur de planification doit pouvoir prendre en compte des buts disjonctifs pour l'algorithme de complétion des trames scénaristiques à partir des points clés ;
- `quantified preconditions` : le nombre d'instances d'un type d'objet donné n'est pas fixé dans le domaine, et donc n'est pas connu lors de la génération des opérateurs. Il est donc nécessaire de pouvoir exprimer à la fois la quantification universelle ($\forall$) et la quantification existentielle ($\exists$) pour représenter certains comportements complexes, à la fois au niveau des préconditions (par exemple, un bras de chargement ne peut être verrouillé que s'il existe une vanne à laquelle il est branché) et des effets (par exemple, l'arrêt d'un programmateur entraine l'arrêt de toutes les pompes auxquelles il est relié).

Ces prérequis correspondent à des tâches de planification classiques, et se retrouvent dans les tâches de type **ADL** [Pednault, 1989], une extension de STRIPS incluant l'ensemble de ces prérequis, ainsi que la définition d'effets conditionnels dans les opérateurs. Les actions ADL peuvent exprimer des effets à l'aide du prédicat `not` pour signaler le retrait d'un fait, plutôt que d'avoir deux listes d'ajout et de retrait séparées comme c'est le cas dans STRIPS. Le choix du moteur de planification intégré dans l'implémentation de DIRECTOR est guidé par ces prérequis.

8.1.2 Choix du moteur de planification

A partir de ces prérequis, nous avons étudié un ensemble de moteurs de planification parmi les plus populaires, en les sélectionnant d'une part pour la disponibilité de leur code source, et d'autre

part pour leur succès dans les différentes compétitions de planification.
Le tableau 8.1 présente les différents moteurs de planification envisagés. Nous avons considéré ici uniquement des moteurs de planification classique, et nous avons notamment laissé de côté les moteurs comme TLPlan [Bacchus and Kabanza, 1998] ou SHOP [Nau et al., 1999] qui utilisent des informations dépendantes du domaine pour guider la recherche. De plus, nous n'avons pas étudié les moteurs trop anciens pour supporter le PDDL, comme Prodigy [Veloso et al., 1995] ou Graphplan [Blum and Furst, 1997]. Cependant, étant donné le grand nombre de moteurs de planification existants, cette liste est tout de même loin d'être exhaustive.

Moteur		**Approche**	**Forme de plan**	**Code**	**ADL**			
					`not`	**Types**	**But** $\wedge$	$\forall/\exists$
SGP [1]	1.0	graphe de plans	POP	Lisp	+	+	+	+
IPP [2]	4.1	graphe de plans	POP	exe	+	$\approx$	+	+
VHPOP [3]	2.2.1	graphe de plans	POP	C++	+	+	+	+
HSP [4]	1.12	recherche heuristique avant	séquentiel	C	+	-	-	-
	2.0	recherche heuristique av./ar.	séquentiel	C	+	+	+	+
Fast-Forward [5]	2.3	recherche heuristique avant + graphe de plans	séquentiel	C	+	+	+	+
Fast Downward [6]	2013-12-12	recherche heuristique avant	séquentiel	C++	+	+	+	+

TABLE 8.1 – Tableau comparatif de différents moteurs de planification

Parmi ces moteurs, nous avons écarté IPP, qui gère les types prédéfinis en PDDL mais ne prend pas en compte les sous-types, et la version originale d'HSP qui ne prend pas en compte tous les prérequis ADL. Les approches utilisées par HSP et FF (Fast-Forward) sont en réalité très proches, mais FF a régulièrement réalisé de meilleures performances que HSP dans les compétitions de planification (voir [Hoffmann and Nebel, 2001b]).
Trois moteurs nous semblent sortir du lot :

- **Fast-Forward**, qui est réputé pour être particulièrement utilisable, de par la qualité des explications fournies lors de la construction des plans, et qui a été utilisé avec succès pour générer des scénarios dans le domaine de la narration interactive [Porteous and Cavazza, 2009].
- **Fast Downward**, dont les différentes variantes (notamment LAMA) ont remporté les premières places dans les catégories "sequential satisfaction" et "sequential optimisation" à IPC 2011 [Garcia-Olaya et al., 2011], et la première place dans la catégorie "sequentiel satisfaction" à IPC 2008 [Helmert et al., 2008].
- **VHPOP**, qui est capable de générer des plans partiellement ordonnés (POP).

1. Sensory Graphplan (SGP) [Anderson et al., 1998] est téléchargeable sur :
`http://www.cs.washington.edu/ai/sgp.html` (accédé le 05/01/2014)
2. Interference Progression Planner (IPP) [Koehler, 2000] est téléchargeable sur :
`http://www.isi.edu/~blythe/cs541/2000/planners/ipp.html` (accédé le 05/01/2014)
3. Versatile Heuristic Partial Order Planner (VHPOP) [Younes and Simmons, 2003] est téléchargeable sur :
`http://www.tempastic.org/vhpop/` (accédé le 05/01/2014)
4. Heuristic Search Planner (HSP) [Bonet and Geffner, 2001] est téléchargeable sur :
`http://code.google.com/p/hsp-planners/` (accédé le 05/01/2014)
5. Fast-Forward (FF) [Hoffmann and Nebel, 2001a] est téléchargeable sur :
`http://fai.cs.uni-saarland.de/hoffmann/ff.html` (accédé le 05/01/2014)
6. Fast Downward [Helmert, 2006] est téléchargeable sur :
`http://www.fast-downward.org` (accédé le 05/01/2014)

L'avantage de l'utilisation de PDDL pour représenter les tâches de planification est que les différents moteurs sont conçus pour être compatibles avec ces domaines, dans la mesure où ils possèdent les bons prérequis. Il est donc particulièrement aisé de passer d'un moteur de planification à un autre. Ainsi, plutôt que de choisir un seul moteur, nous avons choisi d'intégrer ces trois moteurs dans DIRECTOR, afin de pouvoir comparer leurs performances pour la génération des scénarios : ces trois moteurs reflétant des approches différentes, cela permet ainsi de tirer des conclusions sur la génération de la tâche de planification plutôt que sur les performances d'une technique de planification particulière.
De plus, ces trois moteurs étant implémentés en C ou C++, cela donne la possibilité de les utiliser comme une librairie dans une architecture globale en C++, comme celle de DIRECTOR qui est présentée dans la section suivante.

8.2 Architecture

Le processus global de génération, suivi et exécution des scénarios de DIRECTOR a été implémenté dans un module développé en C++. Ce module utilise les librairies BGL (Boost Graph Library) [7] et Qt [8]. Il intègre les trois moteurs de planification évoqués précédemment : FF-v2.3, Fast Downward et VHPOP.
Un effort particulier a été fourni lors de l'implémentation pour mettre en place une architecture et des interfaces claires, afin que DIRECTOR puisse être étendu et réutilisé par la suite. Le diagramme UML des principales classes de l'implémentation est présenté dans la figure 8.1. Les diagrammes concernant les types de graphes manipulés et les concepts liés au PDDL sont présentés dans l'annexe C.

8.2.1 Génération du domaine PDDL

La génération du domaine PDDL est réalisée offline par un module complémentaire développé en Python.
Tout d'abord, l'ontologie du modèle de domaine est interrogée pour récupérer la hiérarchie des types d'objets, ainsi que les noms des Comportements, qui sont déclarés comme des constantes dans le domaine de planification.
Le fichier XML correspondant au modèle d'activité, créé au préalable via un éditeur graphique, est ensuite parsé et utilisé pour générer un ensemble d'opérateurs de prédiction d'après les algorithmes décrits dans la section 6.4.
A ce jour, seule la génération des opérateurs de prédiction d'actions à partir du modèle d'activité a été implémentée. Les opérateurs de prédiction des comportements ont fait l'objet d'une traduction manuelle à partir du modèle du domaine.

8.2.2 Génération du scénario

La génération d'un scénario à partir d'une trame scénaristique est implémentée selon l'algorithme présenté dans la section 7.5.3. A chaque itération, un nouveau problème PDDL est généré à partir du but disjonctif, et donné en entrée au moteur de planification sélectionné.
Dans le cas de FF et Fast Downward, le plan ainsi obtenu est séquentiel, ce qui ne correspond pas au scénario désiré pour le suivi et l'exécution dans l'environnement virtuel. Dans le cas de VHPOP, le plan est partiellement ordonné, cependant certains liens de causalité, tels ceux basés sur les prédicats `(has-turn ?a)`, servent à s'assurer de l'alternance des actions des différents agents lors de la génération du plan mais n'ont pas lieu de figurer dans le scénario final.

7. `http://www.boost.org`
8. `http://qt-project.org`

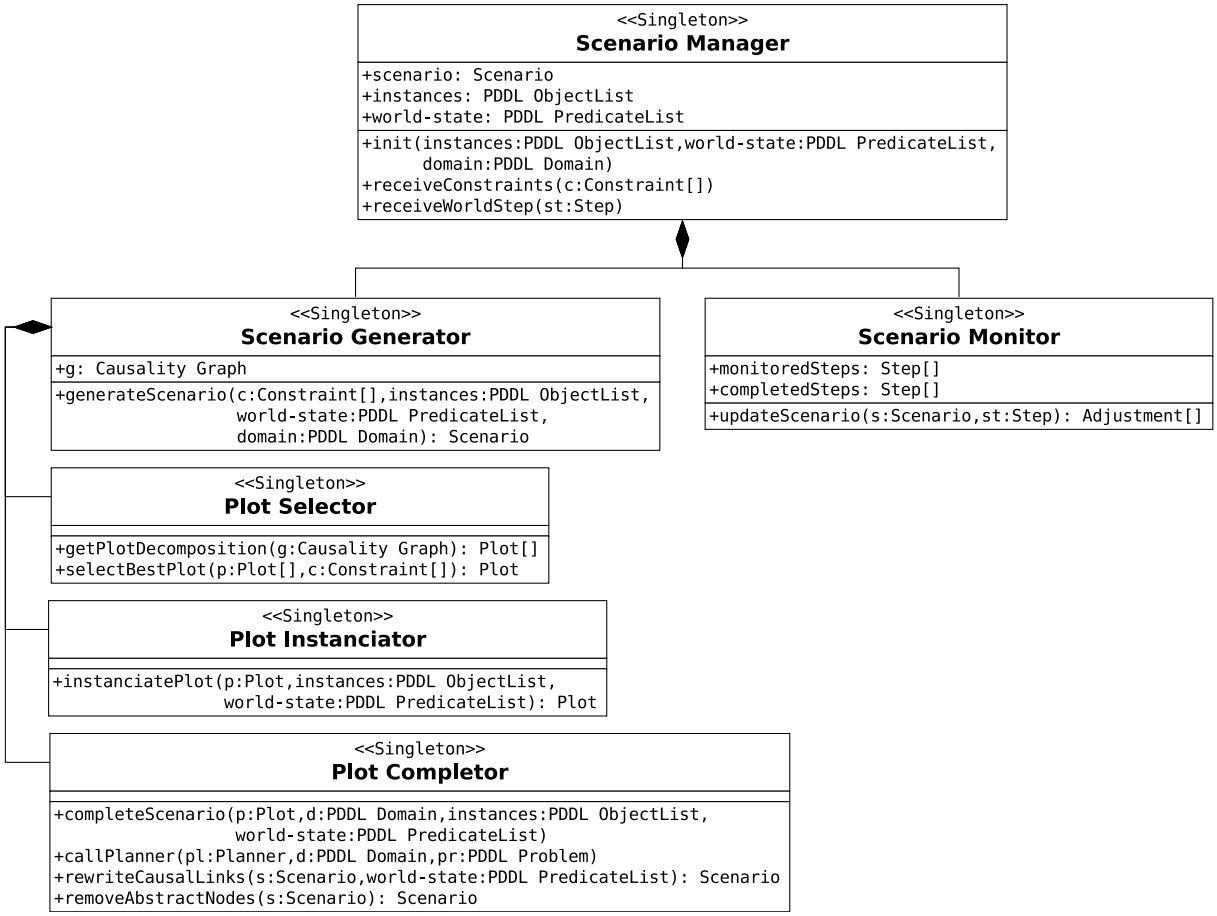


FIGURE 8.1 – Diagramme UML de l'architecture de DIRECTOR

Le scénario résultant de la phase de complétion est donc traité de la manière suivante : dans un premier temps, les plans séquentiels sont transformés en plans partiellement ordonnés, en liant chaque étape à un ensemble d'étapes précédentes — situation initiale comprise — qui permette de vérifier ses préconditions, en raisonnant à partir des préconditions et des effets de chaque étape. Puis ces plans sont nettoyés en supprimant les liens redondants : par exemple si il existe déjà des liens de précédence entre les étapes S_1 et S_2 et entre S_2 et S_3, alors le lien entre S_1 et S_3 sera supprimé. Puis dans un deuxième temps, les opérateurs abstraits sont supprimés de ces plans partiellement ordonnés, en conservant les liens de précédence entre les étapes qui les précèdent et les suivent directement, à l'exception des opérateurs `Abstract-EndTurn` où le lien n'est pas recréé entre les actions des deux agents concernés, mais entre l'action du deuxième agent et le dernier comportement déclenché à la suite de l'action précédente de ce même agent. Cela permet ainsi d'obtenir des séquences d'actions parallèles pour chaque agent dans le cas où les actions de l'un n'influencent pas celles de l'autre, et réciproquement.

8.2.3 Interfaces

DIRECTOR est doté d'une interface graphique, qui utilise le logiciel GraphViz [9] pour générer automatiquement des graphes permettant de visualiser les scénarios générés et de suivre l'avancement de ces scénarios par rapport aux événements de l'environnement virtuel. Une capture d'écran de cette interface est présentée dans la figure 8.2.

9. `http://www.graphviz.org`

DIRECTOR possède également une interface d'envoi et de réception de messages en JSON [10], afin de pouvoir communiquer avec les modules de simulation du monde, de génération des comportements des personnages virtuels, et de suivi de l'utilisateur. Il peut ainsi être intégré au sein de la plateforme HUMANS — décrite dans l'annexe B — mais pourrait également être interfacé avec d'autres modules sous réserve qu'ils utilisent le même format de messages.

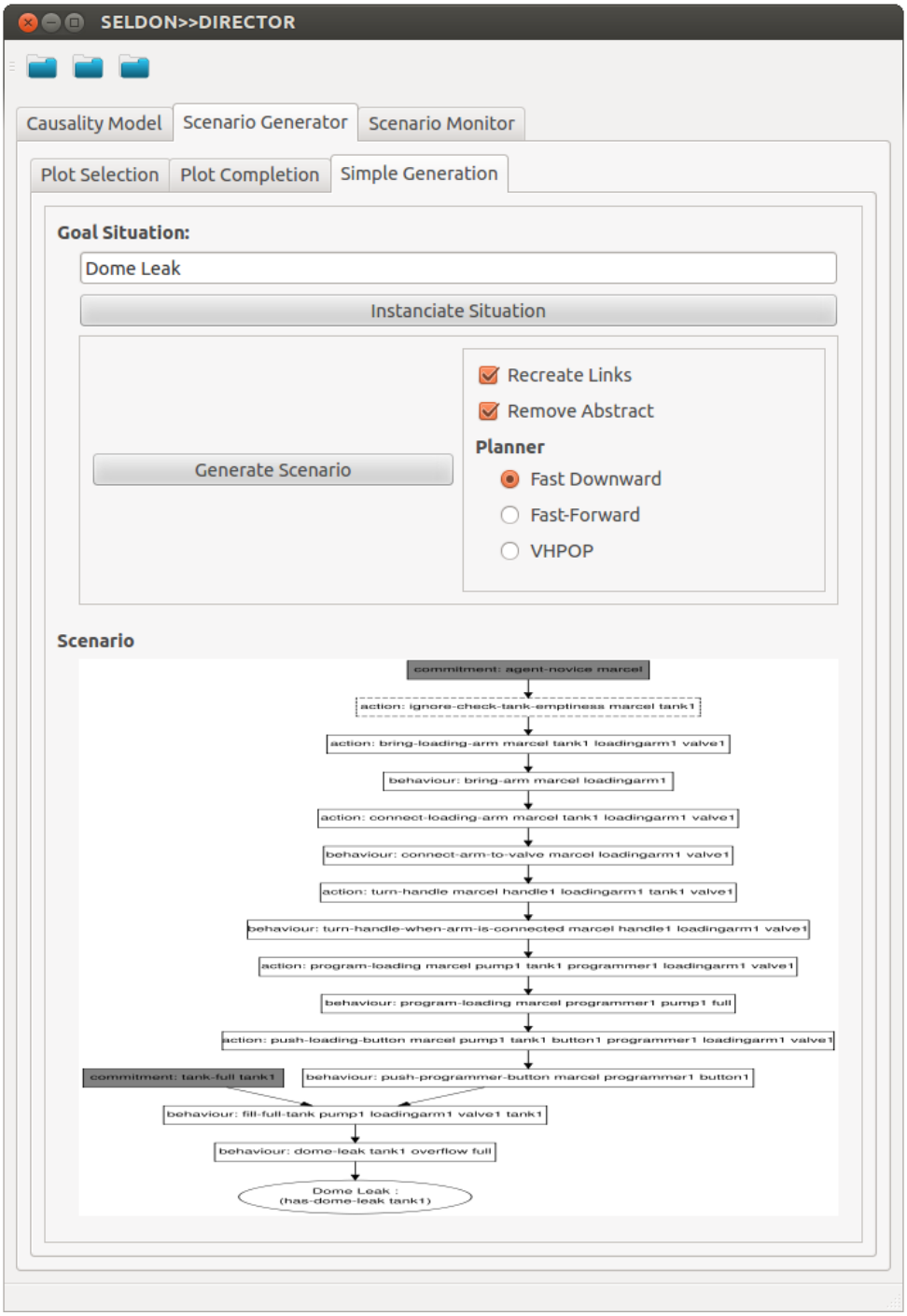


FIGURE 8.2 – Capture d'écran de l'interface utilisateur de DIRECTOR

10. http://www.json.org/

8.2.4 Plateforme HUMANS

L'implémentation de DIRECTOR a été réalisée au sein de la plateforme de création d'environnements virtuels HUMANS, décrite en détail dans l'annexe B. La part de mes contributions à cette plateforme ainsi que l'état de l'implémentation des différents modules sont présentés dans la figure 8.3.

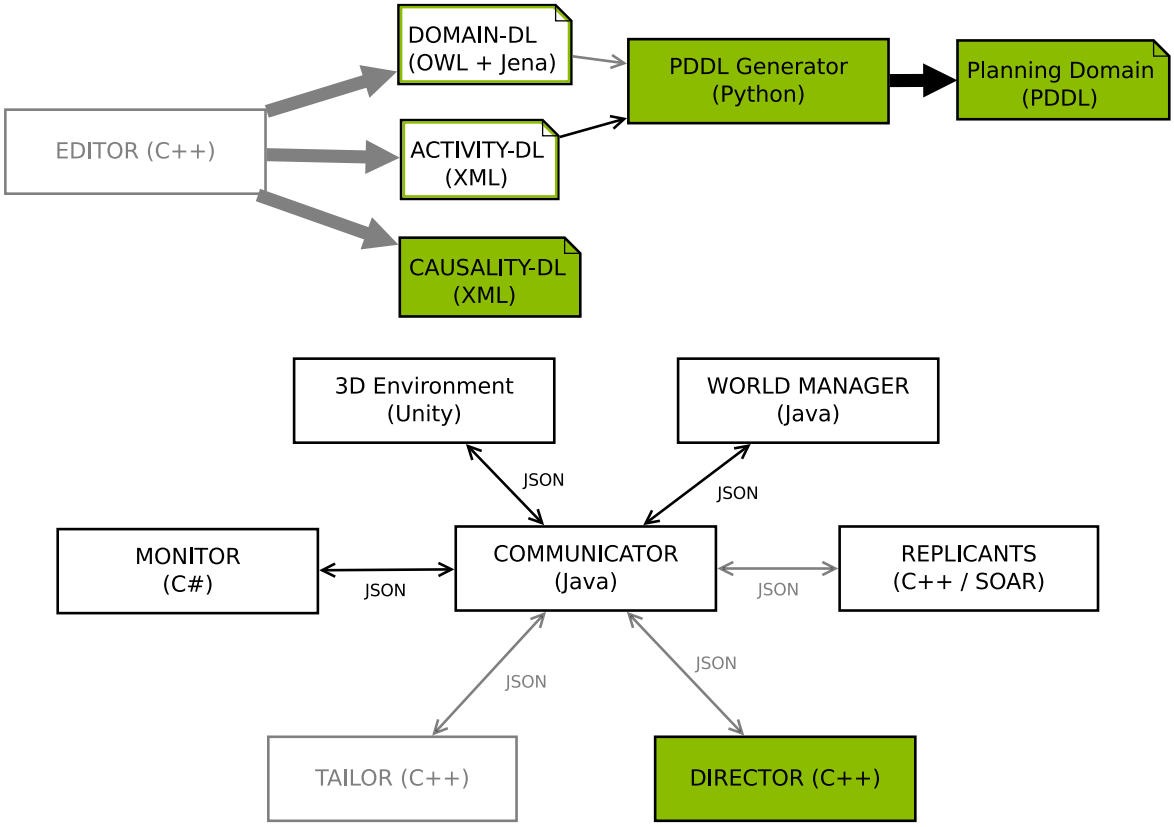


FIGURE 8.3 – Schéma des différents modules de la plateforme HUMANS.
Les éléments colorés en vert ont été proposés dans le cadre de cette thèse.
Les éléments entourés en vert ont été proposés par l'ensemble de l'équipe HUMANS et enrichis dans le cadre de ce travail (*événements* pour DOMAIN-DL, *constructeurs* et *préconditions* pour ACTIVITY-DL).
Les éléments grisés n'ont pas encore été implémentés à l'heure actuelle.

Cette implémentation a servi de support à une évaluation informatique de DIRECTOR, présentée dans le chapitre suivant.

Chapitre 9

Validation

Sommaire

Les systèmes de scénarisation ou de génération d'histoires sont réputés pour être difficiles à valider. Jusqu'ici, peu d'entre eux ont fait l'objet d'une évaluation, qu'il s'agisse d'évaluations individuelles ou de comparaisons entre systèmes [Magerko, 2007].
L'une des raisons principales de ce constat est que chaque système est conçu avec un ensemble de scénarios particuliers en tête, et a donc ses propres objectifs en termes de scénarios générés. Le contrôle sur le scénario offert par un système comme SELDON, qui accorde une grande importance à mener le scénario vers un événement but précis, n'est ainsi pas comparable à celui offert par exemple par Thespian [Si, 2010] qui ne peut prévoir les événements qu'à un horizon très limité, mais qui s'attache à satisfaire des contraintes temporelles sur la réalisation de différents événements.
De même, il existe une forte inter-dépendance entre les moteurs de scénarisation et les formalismes de représentation du contenu qu'ils utilisent. Il ne serait pas possible de modéliser le même contenu pour deux systèmes différents ; par exemple, les formalismes utilisés par un moteur visant des scénarios fortement axés sur des simulations de systèmes technico-humains, comme SELDON, et par un moteur visant des scénarios centrés sur des dialogues en langage naturel, comme par exemple Façade [Mateas and Stern, 2005], ne permettront pas de représenter les mêmes connaissances.
Contrairement à d'autres domaines, comme celui de la planification, il n'existe pas de "benchmark" pour la scénarisation ou la narration interactive. On peut cependant souligner une tentative dans cette direction ayant été réalisée lors d'un atelier de la conférence TIDSE 2006 (Technologies for

Interactive Digital Storytelling and Entertainment), autour du conte bien connu du Petit Chaperon Rouge [Spierling and Iurgel, 2006]. Dans cet atelier, qui portait sur l'*authoring*, c'est à dire sur la création du contenu et des objectifs scénaristiques, les participants étaient encouragés à modifier leurs systèmes de scénarisation pour permettre la génération d'histoires sur le modèle du Petit Chaperon Rouge [1]. Cependant, le problème de l'adéquation entre domaine et systèmes de scénarisation se pose ici à nouveau. Comme noté par [Riedl et al., 2008b], ce conte est particulièrement difficile à transposer en narration interactive : le personnage principal est une victime et se fait dévorer, et il y a dans l'ensemble très peu de personnages et d'interactions. Pour SELDON, qui repose sur des modèles de systèmes techniques complexes et représente l'activité comme un ensemble de procédures et de déviations de ces procédures, l'exemple du Petit Chaperon Rouge serait particulièrement délicat à adapter.
Une raison supplémentaire à cette difficulté d'évaluation est que, les scénarios générés dépendant fortement du contenu scénaristique représenté, il est également délicat de distinguer l'évaluation du système de scénarisation lui-même de l'évaluation de la qualité des modèles de domaine ou d'activité utilisés. Un problème additionnel se pose dans notre cas : les cas d'application considérés étant des cas métier fortement techniques, des connaissances métier adéquates sont nécessaires pour pouvoir juger de la cohérence des comportements des personnages virtuels et du système à partir du seul résultat de la simulation. Un panel de sujets suffisamment expérimentés dans le domaine serait nécessaire à l'expérimentation. Mais même alors, une telle évaluation ne permettrait pas de distinguer entre l'explicabilité du scénario final par rapport aux modèles de la simulation, et la validité écologique de ces modèles eux-mêmes.
Nous avons donc opté dans un premier temps pour une évaluation informatique, portant notamment sur les performances des différents moteurs de planification sur les tâches de planification générées par DIRECTOR. Cette évaluation est associée à des retours qualitatifs sur la conception des environnements virtuels et des différents modèles associés. Nous proposons enfin un ensemble de pistes d'évaluation se rapportant aux différents objectifs fixés pour notre système de scénarisation, qui, s'ils elles n'ont pas pu être concrétisées dans le cadre de ce travail de thèse, nous paraissent toutefois devoir être explorées par la suite .

9.1 Evaluation informatique

Pour évaluer la phase de génération des scénarios par DIRECTOR, nous avons comparé les performances de trois moteurs de planification sur plusieurs domaines. Nous présentons ici tout d'abord les domaines et les moteurs de planification utilisés pour cette comparaison, puis les résultats sur le plan qualitatif — influence du moteur sur le contenu du scénario et validité des plans générés — et sur le plan quantitatif — temps de génération. Nous discutons ensuite brièvement ces résultats.

9.1.1 Eléments comparés

Nous avons utilisé pour cette comparaison deux domaines distincts, dans plusieurs configurations, ainsi que trois moteurs de planification, là encore avec différents ensembles de paramètres.

Domaines

La génération de scénarios a été réalisée sur deux exemples : **Gas Station**, un exemple très simplifié portant sur une opération de remplissage d'une voiture dans une station service, et **Arakis**, un exemple plus complet correspondant au chargement d'un camion citerne sur un dépôt pétrolier.

1. Les résultats de cet atelier peuvent être consultés sur : `http://redcap.interactive-storytelling.de/` (accédé le 05/01/2014)

Chaque exemple est associé à un modèle d'activité en ACTIVITY-DL, qui est utilisé pour générer un domaine de planification PDDL, auquel sont ensuite rajoutés les opérateurs de prédiction des comportements, traduits manuellement depuis le modèle du domaine en DOMAIN-DL.
Chacun des deux exemples a été testé avec deux situations buts différentes : **End**, correspondant à la réalisation de la tâche principale, et **Fire**, qui correspond au déclenchement d'un feu. La première demande un minimum d'ajustements, tandis que la seconde en nécessite *a priori* davantage. Dans le cas du feu, la génération a été faite avec et sans sélection préalable d'une trame scénaristique.
De plus l'exemple **Arakis**, de par sa taille, a été testé avec deux modèles de domaine différents : une version **Arakis-Basic**, où seuls sont présents les opérateurs de prédiction des comportements correspondant au bon déroulement de la procédure (c'est à dire que les comportements "accidentels" — feu, fuite, etc. — ont été enlevés), et une version **Arakis-Full** qui comporte l'ensemble des opérateurs.
Enfin, chaque exemple a été testé avec une situation initiale correspondant à la présence d'**un seul agent**, ou bien de **deux agents**.
Un troisième domaine de planification est présenté à titre de comparaison : **Fake**, qui contient des actions écrites à la main et non générées à partir d'un modèle d'activité, et qui ne garantit donc pas la cohérence des actions des agents. Il s'agit du domaine utilisé pour générer le plan présenté précédemment dans la figure 6.4. Le tableau 9.1 récapitule les caractéristiques de chaque condition.

Domaine	**Fake**	**Gas Station**			**Arakis**			
					Basic	**Full**		
Tâches mères	-	1			4	5		
Tâches feuilles	-	4			8	11		
Comportements	-	10			14	34		
Ajustements	-	9			12	18		
Opérateurs	13	31			52	88		
Prédicats	18	24			38	56		
Constantes	17	26			45	59		
Objets	5	5 / 10			10 / 20	14 / 28		
But	Fire	End	Fire		End	End	Fire	
Trame	-	-	-	+	-	-	-	+
Identifiant	F	G_1^E / G_2^E	G_1^F / G_2^F	G_1^{F*} / G_2^{F*}	Ab_1^E / Ab_2^E	A_1^E / A_2^E	A_1^F / A_2^F	A_1^{F*} / A_2^{F*}

TABLE 9.1 – Tableau récapitulatif des domaines ayant été utilisés pour les évaluations

Le modèle d'activité écrit pour l'exemple `Gas Station`, ainsi que le domaine PDDL généré à partir de celui-ci, et les deux problèmes PDDL générés par DIRECTOR pour chacun des buts — dans la configuration à un seul agent sans sélection de trame scénaristique — sont retranscrits dans l'annexe E.

Moteurs de planification

La génération des scénarios a été réalisée avec les trois moteurs de planification présentés précédemment : **FF**, **Fast Downward** et **VHPOP**.
FF-v2.3 a été utilisé sans modifications, ce qui correspond au moteur utilisé comme point de comparaison lors de la compétition IPC-6 pour la catégorie "satisfaction" [Helmert et al., 2008].
Fast Downward a été testé dans deux configurations différentes : d'une part, avec une recherche A* sans heuristique (A* blind), ce qui correspond au moteur utilisé comme point de comparaison pour la catégorie "optimisation" lors de la même compétition, et d'autre part dans une configuration "lazy greedy best-first" avec deux heuristiques : FF et "context-enhanced additive". On les notera respectivement FD_{A*} et FD_{LG}.

VHPOP a été également utilisé dans deux configurations : d'un côté, la configuration utilisée pour la compétition IPC-3, et de l'autre, une configuration reprenant les heuristiques et stratégies de sélection de failles du moteur UCPOP sur lequel il est basé. On notera respectivement V_{IPC} et V_{UCP}.

9.1.2 Influence du moteur sur les plans générés

Le premier constat réalisé est que le choix du moteur de planification influence fortement le contenu du plan généré, et donc du scénario, et ce même sur des domaines très simples comme `Gas Station` où il n'existe qu'une seule combinaison d'ajustements permettant d'aboutir à l'événement `Fire`.
La figure 9.1 présente une comparaison visuelle des plans générés à partir des conditions G_F^1 par trois moteurs différents : le plan (a) a été généré avec FF, le plan (b) a été généré avec FD_{A*} et le plan (c) a été généré par V_{IPC}/V_{UCP} (FD_{LG} génère, comme FF, le plan (a)). Les plans (a) et (b) ont fait l'objet d'une phase de réécriture des liens de causalité pour obtenir un plan partiellement ordonné, et les opérateurs abstraits ont ensuite été supprimés des trois plans. Les graphes correspondant sont retranscrits dans l'annexe E.
On peut ainsi remarquer que le plan (b) contient le minimum d'ajustements nécessaires, tandis que le plan (a), généré par des moteurs visant un plan satisfaisant et non optimal, contient des ajustements inutiles qui mènent à un comportement de type `Ignore`.
Le plan (c), dont les étapes ont été directement ordonnées par le moteur VHPOP, n'est pas optimal non plus, dans le sens où certains ajustements sont placés en début de plan alors qu'ils ne sont nécessaires que pour des étapes ultérieures. Ceci peut poser problème dans le cas où l'on doive générer un nouveau plan en cours d'exécution : ces ajustements auront alors été déclenchés inutilement, ce qui aurait pu être évité en attendant le dernier moment pour les lancer.

9.1.3 Validité des plans générés

Les plans générés ont été validés à l'aide du validateur PDDL INVAL [2], créé par Patrick Haslum.
D'une part, nous nous sommes assurés de la validité des plans directement générés par les moteurs de planification : plans séquentiels pour FF et FD, et plans partiellement ordonnés pour $VHPOP$.
D'autre part, nous avons cherché à valider les plans générés par DIRECTOR à partir de ces moteurs en réalisant les étapes suivantes :

- complétion du graphe de points clés avec les plans intermédiaires générés par les moteurs (dans le cas où le scénario résulte de la sélection préalable d'une trame scénaristique) ;
- réécriture des liens de causalité entre les étapes ;
- traduction de ce graphe de scénario vers une représentation d'un plan partiellement ordonné en PDDL, en parcourant le graphe à partir de la situation finale et en associant à chaque étape un marqueur temporel, en suivant l'algorithme 5 décrit dans l'annexe D ;
- validation de ces plans à partir d'un problème PDDL composé de la situation initiale et de la situation but du scénario.

Un problème intéressant se pose pour la validation des plans générés par DIRECTOR à partir de domaines comportant un seul agent et plus d'un niveau de tâches mères. On se retrouve alors dans le cas où les étapes `Abstract-EndTurn` possèdent comme effet à la fois le retrait et l'ajout du fait `(has-turn agent)`, et que ce fait est présent dans les préconditions des étapes `Abstract-End-<Task>`, qui sont considérées comme parallèles par DIRECTOR. Elles peuvent en effet être appliquées soit avant soit après ces dernières, selon que l'on considère que l'agent évalue les conditions de satisfaction de ses tâches en cours à la fin de son tour, ou au début de son tour suivant — ce qui n'a pas d'incidence sur le scénario une fois les tâches abstraites retirées. Il semblerait que INVAL considère ce cas comme un conflit (une précondition d'une étape étant rendue vraie par une étape parallèle), même si dans notre

2. INVAL est disponible sur : `http://users.cecs.anu.edu.au/~patrik/` (accédé le 05/01/2014)

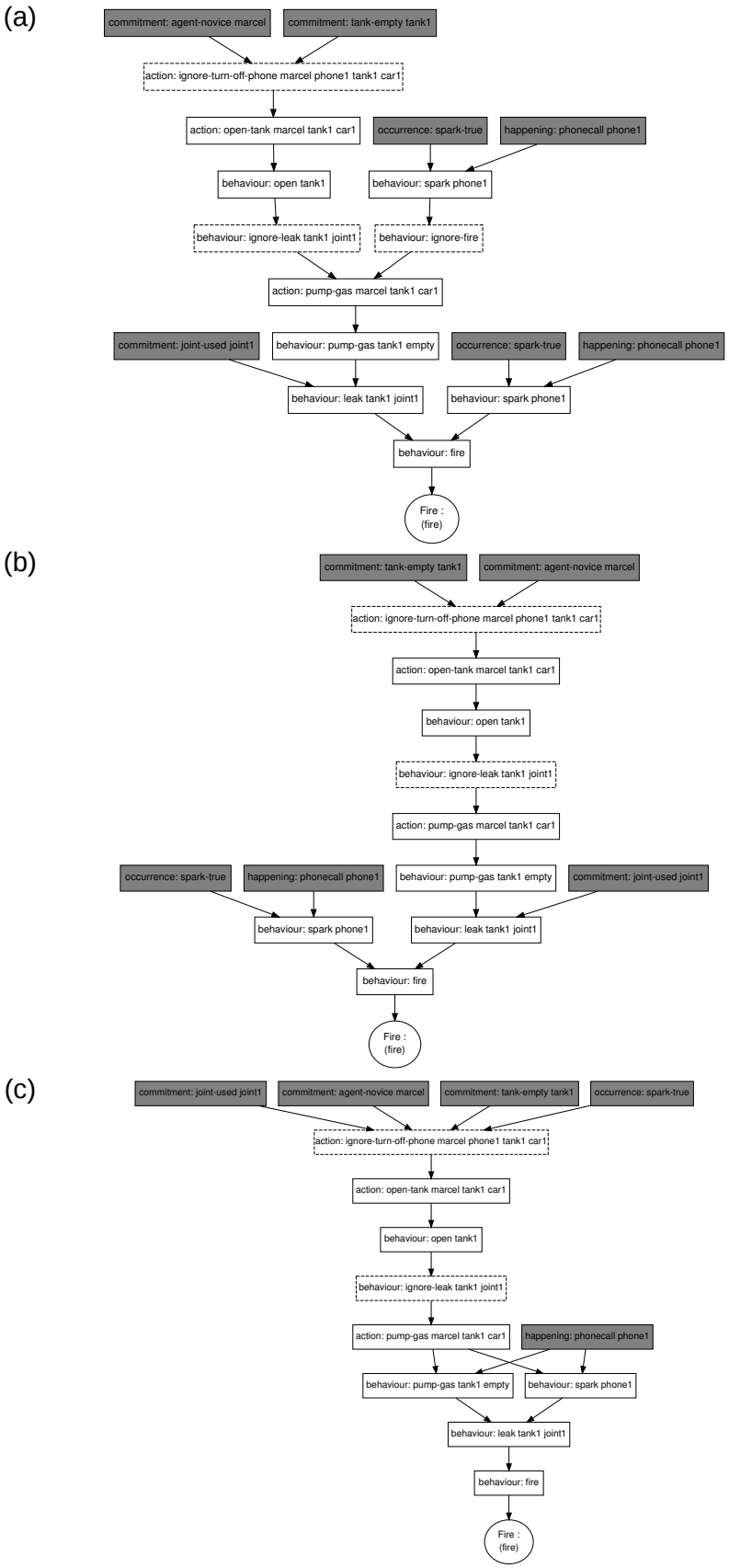


FIGURE 9.1 – Scénarios générés dans la configuration G_1^F avec les trois moteurs de planification : (a) FF, (b) Fast Downward (A* blind), (c) VHPOP

cas l'opérateur `Abstract-EndTurn` a un impact nul sur la base de faits. Ces plans sont cependant considérés comme valides par INVAL à partir du moment où les étapes sont ordonnées manuellement, dans un ordre comme dans l'autre, nous considérons donc que ces plans constituent des plans partiellement ordonnées PDDL valides. Il est également à noter que VHPOP génère les plans correspondants comme des plans séquentiels et ne considère pas lui non plus ces étapes comme pouvant être parallèles. Une fois ce cas particulier résolu, cependant, tous les plans générés par DIRECTOR ont été considérés comme valides par INVAL.
Une meilleure mesure de la validité d'un scénario serait toutefois de valider ceux-ci après la suppression des étapes abstraites : pour cela il serait nécessaire de modifier le domaine de planification en supprimant toutes les préconditions et effets des opérateurs qui ont trait à l'ordonnancement des tâches et des actions (`has-turn`, `task-state`, etc.).

9.1.4 Temps de génération des plans

Les temps obtenus pour la génération des plans dans chaque condition et par chaque moteur de configuration sont présentés dans les tableaux 9.2 et 9.3. Chaque chiffre correspond à la moyenne des temps obtenus sur trois itérations de la génération de plan.
Le tableau 9.2 présente les temps obtenus dans le cas de la génération d'un scénario sans sélection de trame scénaristique préalable, chiffres qui correspondent donc au temps mis par le moteur de planification pour générer un plan, auquel doivent être ajoutés le temps de transformation du plan séquentiel en plan partiellement ordonné pour FF et Fast Downward, et le temps de suppression des opérateurs abstraits dans tous les cas, soit entre $0.010s$ et $0.040s$ pour chacune de ces deux étapes (temps obtenus sur les générations présentées ici).
Le tableau 9.3 présente les temps obtenus pour la génération de scénario avec sélection préalable d'une trame scénaristique, ici choisie arbitrairement. La trame scénaristique utilisée pour l'exemple *G* comportait 3 points clés (`Source d'Ignition`, `Fuite` et `Feu`), tandis que la trame utilisée pour l'exemple *A* en comportait 7 (il s'agit de celle présentée dans la figure 7.5). La première ligne de chaque exemple correspond au temps total de complétion de la trame scénaristique (temps de création des liens de causalité et de suppression des opérateurs abstraits compris), et les lignes suivantes correspondent aux temps de planification obtenus pour chaque itération sur les points clés.
Une durée de $120s$ a été fixée comme limite haute pour la génération d'un plan, durée à partir de laquelle la génération a été arrêtée manuellement ; ces cas sont indiqués dans le tableau par le symbole ∞. Le symbole `x` représente les cas où le moteur de planification a subi un crash lors des trois tentatives de génération de plan. Le symbole − indique que la génération n'a pas pu être évaluée dans ces conditions : soit parce que la génération a été arrêtée lors d'une étape précédente pour la complétion d'une trame scénaristique, soit dans le cas de VHPOP et du domaine *A* pour une raison de prérequis. En effet, si VHPOP est censé prendre en compte les quantifications universelles dans les préconditions et effets des opérateurs, les heuristiques correspondantes ne sont implémentées que lorsque les quantifications portent sur des constantes, et non sur des objets. Le domaine `Arakis Full`, qui contient de telles préconditions, n'a donc pas pu être évalué avec VHPOP.

9.1.5 Discussion

Des trois moteurs utilisés, seul Fast Downward s'est avéré suffisamment stable et complet pour la génération de plans sur les domaines créés par DIRECTOR. FF, qui a obtenu des résultats prometteurs sur le domaine *G*, n'a pas été capable de traiter le domaine *A*, d'une taille plus conséquente. VHPOP, quant à lui, s'est heurté sur ce dernier à un problème de prise en compte des quantifications universelles.
La question se pose à présent de savoir si les temps obtenus par Fast Downward sont acceptables pour les besoins de DIRECTOR en termes de génération de scénarios. D'après [Porteous et al., 2010],

	FF	FD_{A*}	FD_{LG}	V_{IPC}	V_{UCP}
F	0.009	0.093	0.086	0.010	0.009
G_1^E	0.012	0.141	0.129	0.379	6.925
G_2^E	0.014	14.739	0.173	∞	x
G_1^F	0.012	0.167	0.132	17.889	0.370
G_2^F	0.017	10.654	0.181	∞	x
Ab_1^E	1.659	0.205	0.197	4.955	∞
Ab_2^E	x	0.615	0.325	∞	x
A_1^E	x	18.322	0.247	–	–
A_2^E	x	∞	∞	–	–
A_1^F	x	12.860	0.241	–	–
A_2^F	x	∞	∞	–	–

TABLE 9.2 – Temps de génération en secondes par condition et par moteur (sans sélection de trame scénaristique préalable)
x : crash | – : impossible de générer | ∞ : génération stoppée après $t = 120s$

		FF	FD_{A*}	FD_{LG}	V_{IPC}	V_{UCP}
G_1^{F*}		**0.121**	**0.507**	**0.495**	∞	**1.082**
	i_1	0.012	0.131	0.132	0.010	0.012
	i_2	0.010	0.158	0.139	∞	0.500
	i_3	0.007	0.112	0.120	–	0.440
G_2^{F*}		**0.147**	**7.393**	**0.597**	∞	x
	i_1	0.011	0.157	0.185	0.016	0.014
	i_2	0.006	6.963	0.165	∞	x
	i_3	0.014	0.151	0.151	–	–
A_1^{F*}		x	**4.702**	**2.118**	–	–
	i_1	x	0.237	0.241	–	–
	i_2	–	0.236	0.238	–	–
	i_3	–	2.856	0.235	–	–
	i_4	–	0.213	0.247	–	–
	i_5	–	0.231	0.228	–	–
	i_6	–	0.248	0.238	–	–
	i_7	–	0.216	0.206	–	–
A_2^{F*}		x	∞	∞	–	–
	i_1	x	0.369	0.383	–	–
	i_2	–	0.379	0.372	–	–
	i_3	–	∞	∞	–	–
	...	–	–	–	–	–

TABLE 9.3 – Temps de génération en secondes par condition et par moteur pour la complétion d'une trame scénaristique
x : crash | – : impossible de générer | ∞ : génération stoppée après $t = 120s$

un système de narration interactive devrait avoir un temps de réponse inférieur à 1.5s pour permettre de réagir de manière satisfaisante aux interactions des utilisateurs. Dans le cas de DIRECTOR, cette notion de temps interactif est à mettre en regard de la façon dont le scénario est exécuté : les personnages virtuels et la simulation évoluant de manière autonome, la phase de génération du plan n'est pas bloquante pour l'environnement virtuel. Cependant, pour certains scénarios comportant des ajustements dès les premières étapes, un trop grand délai dans la génération mettrait aussitôt ces scénarios en échec.
Ces chiffres sont également à considérer dans le contexte des systèmes de génération d'histoires textuelles basés sur de la planification, où les temps constatés pour les systèmes qui respectent les motivations des personnages écartent souvent la possibilité d'utilisation pour des applications en temps interactif : la génération d'une histoire avec l'algorithme IPOCL demandant par exemple un temps de l'ordre de 12.3 heures [Riedl and Young, 2010]. Il s'agit cependant d'un domaine qui évolue très rapidement : la transposition de cet algorithme vers des algorithmes de planification classique proposée par [Haslum, 2012] a permis de réduire le temps de génération à 45s sur ce même domaine, tandis que l'approche multiagent de [Teutenberg and Porteous, 2013] obtient des temps allant de 1s à 25s environ.
Si l'utilisateur d'un moteur de planification générant des plans parallèles plutôt que séquentiels semblait au départ être une meilleure solution que d'avoir à recréer les liens de causalité à partir d'un plan séquentiel, cette idée est remise en question au vu des résultats de VHPOP. D'une part, l'ordonnancement des étapes des plans générés par celui-ci n'est pas optimal, dans le sens où l'on souhaite que les ajustements soit déclenchés le plus tard possible dans le scénario. D'autre part, la différence de performance entre VHPOP et FF ou Fast Downward va bien au delà des temps constatés pour la création des liens de causalité nécessaire pour ces derniers.
Un compromis est également à trouver entre l'optimalité du scénario généré — nombre d'ajustements et d'étapes de prédiction nécessaires avant d'arriver à la situation but — et le temps de planification. Ainsi, l'utilisation de l'algorithme A* sans heuristique (FD_{A*}) permet d'obtenir des plans de meilleure qualité que la recherche avec heuristique (FD_{LG}) mais est beaucoup plus coûteuse, en mémoire comme en temps de calcul. Cette solution correspond en fait à l'approche triviale ayant été écartée dans la section 6.3, qui consistait à calculer une prédiction de scénario pour chaque combinaison d'ajustements.
On peut cependant constater que dans ce cas de figure, la sélection préalable d'une trame scénaristique permet d'améliorer fortement les performances à partir d'une certaine taille de problème. C'est le cas pour G_2^F, où l'on passe d'un temps moyen de 10.654s à 7.393s, et pour A_1^F, où l'on passe de 12.860s à 4.702s.
Par ailleurs, les performances obtenues pour la complétion d'une trame scénaristique pourraient être grandement améliorées. Le temps de planification par les moteurs comprend en effet, en plus du temps de recherche, le temps nécessaire à l'instanciation des différents opérateurs avec les objets et les constantes correspondant aux types des paramètres de ces opérateurs. Dans le cas de FF, par exemple, l'instanciation sur le domaine Ab_1 prend en moyenne 1.51s sur 1.66s de temps total de planification. Cette phase d'instanciation pourrait ainsi être mise en commun pour l'ensemble des itérations sur les points clés, en instanciant les opérateurs au préalable comme c'est le cas par exemple dans [Porteous and Cavazza, 2009].
Cette instanciation préalable permettrait également de réduire le nombre d'opérateurs considérés. En effet, certaines relations entre objets peuvent être déclarées comme statiques dans le modèle du domaine : `(has-tank ?truck ?tank)`, ou `(has-spring ?valve ?spring)` par exemple. Toutes les combinaisons de ces objets seront instanciées au niveau des opérateurs, pourtant les préconditions de ces derniers ne pourront être vérifiées que dans le cas où ces relations statiques sont présentes dans l'état initial du monde. L'instanciation des opérateurs pourrait donc être limitée aux combinaisons d'objets qui permettent de vérifier ces relations.

9.2 Retours d'usage sur la conception des modèles

Le projet NIKITA, décrit dans la section 1.7.2, a donné lieu à la création d'un environnement virtuel utilisant la plateforme HUMANS. Cet environnement vise à servir de support à la formation des assembleurs-monteurs en aéronautique, sur la tâche d'assemblage de la barque d'un avion. Des ergonomes du LATI de l'Université Paris Descartes ont réalisé les analyses terrain, à partir desquelles ils ont écrit les modèles de domaine et d'activité.

9.2.1 DOMAIN-DL

Pour l'écriture du modèle de domaine, les ergonomes ont utilisé la version gratuite de l'éditeur TopBraid Composer[3]. Il leur a également été fourni le méta-modèle ontologique de DOMAIN-DL, ainsi que plusieurs exemples de règles pour les comportements.
Les ergonomes ont ainsi défini 22 composants, 28 types d'objets, 33 actions et 34 comportements. Si la définition des concepts et des relations dans l'ontologie n'a demandé de notre part que des interventions limitées, le formalisme des règles s'est avéré au contraire inutilisable pour les ergonomes.
Cette difficulté pourrait être dûe en grande partie à la syntaxe Jena des règles, qui se trouve être assez lourde. Un outil auteur est en cours de développement pour tenter d'explorer davantage les possibilités d'utilisations de DOMAIN-DL par des non-informaticiens.

9.2.2 ACTIVITY-DL

Les ergonomes ont également écrit un modèle d'activité associé à ce modèle du domaine, portant sur une tâche d'assemblage d'un support à la barque de l'avion, tâche qui demande notamment des opérations de perçage de trous et de pose d'agrafes.
Du fait du manque d'éditeur dédié à ACTIVITY-DL, la structure de base du modèle a été créée dans l'éditeur Visual-HAWAI, correspondant à l'ancienne version du langage [Amokrane, 2010], et complétée ensuite en XML.
Le modèle d'activité ainsi créé contient 11 tâches mères et 37 tâches feuilles, pour une durée réelle de la procédure d'une quinzaine de minutes. La structure arborescente du modèle d'activité ainsi que le lien explicite avec les objets et actions définis dans le modèle du domaine ont été particulièrement appréciés, car permettant de structurer le modèle et d'éliminer certaines ambiguïtés.
Le problème majeur qui s'est posé lors de la modélisation a été celui de la granularité des actions à modéliser, de par la distinction des tâches feuilles (liées aux actions du modèle du domaine) et des tâches mères ; or, ce qui constitue une tâche élémentaire dans cette procédure dans certains contexte (lors de la phase de production par exemple) sera au contraire décomposé en un ensemble de tâches élémentaires dans un autre (par exemple, en début de formation).
De plus, du fait de la structure arborescente du modèle, les ergonomes ont été forcés de répéter à l'identique certains sous-arbres de tâches dans des branches différentes. Cela pose notamment un problème au niveau de la maintenabilité, puisque si une partie de cet arbre est modifié, cette modification doit être reportée manuellement à tous les endroits où il est présent. Une piste d'amélioration serait la proposition de "templates" de tâches, qui prendraient en paramètres un ensemble de variables, à la manière du contexte des tâches, et correspondraient à des sous-arbres de tâches qui pourraient être référencés à plusieurs endroits du modèle d'activité.
Comme pour le modèle du domaine, un éditeur dédié est en cours de réalisation, permettant d'extraire directement les concepts et les relations définies dans le modèle de domaine, afin de pouvoir spécifier les actions, les objets et les prédicats concernés par les tâches, et réciproquement d'ajouter au modèle de domaine les éléments manquants par rapport au modèle d'activité.

3. TopBraid Composer est disponible sur : `http://www.topquadrant.com/` (accédé le 05/01/2014)

9.3 Pistes d'évaluation

Nous proposons ici un ensemble de pistes pour l'évaluation de nos propositions, en regard des différents objectifs qui ont été définis pour notre système de scénarisation. Si ces pistes n'ont pas pu être concrétisées dans le cadre de ce travail de thèse, il nous semble essentiel de veiller à la validation de ces différents points avant d'envisager l'utilisation du système SELDON en conditions réelles.

9.3.1 Efficacité de la prédiction

Tout d'abord, il serait nécessaire d'évaluer la prédiction des actions des personnages virtuels et des comportements des systèmes techniques simulés, en validant les opérateurs de prédiction générés à partir des modèles de domaine et d'activité.
Pour cela, il serait possible de lancer la simulation à partir d'un état initial donné, d'explorer l'espace d'états à partir de l'état courant pour chaque message de changement d'état transmis par la simulation, et de vérifier, d'une part, qu'un seul opérateur d'action peut être appliqué par personnage, et d'autre part, que cet opérateur correspond bien à la prochaine action qui sera choisie par celui-ci.
Il serait également possible, à partir de chacune des situations permises par la simulation, de lancer cette dernière jusqu'à ce qu'elle arrive à une seconde situation donnée, considérée arbitrairement comme la fin du scénario (fin de la procédure d'un personnage, accident, etc.), puis d'utiliser un moteur de planification pour calculer, en utilisant des opérateurs de prédiction uniquement, un plan allant de l'une à l'autre de ces situations, et de comparer le plan généré avec la trace de la simulation.
La prédiction des actions des utilisateurs est quant à elle plus délicate à évaluer, puisqu'elle nécessite de faire la distinction entre ce qui relève de la transposition du modèle d'activité vers des opérateurs de planification, et ce qui relève directement du modèle d'activité lui-même.

9.3.2 Efficacité du contrôle

De même, l'efficacité du contrôle offert par les ajustements devrait être validée. Il serait possible, de manière similaire, de comparer systématiquement le scénario planifié et le scénario effectif, pour un ensemble de combinaisons de situations initiales et d'objectifs scénaristiques. Il faudrait alors limiter les agents aux personnages virtuels ou, à défaut, à un utilisateur se comportant de manière conforme au modèle d'activité.

9.3.3 Résilience du système

En plus de valider le contrôle du scénario dans le cas où les agents se comportent de manière conforme aux prédictions, il faudrait également évaluer la résilience du système face aux actions de l'utilisateur. En particulier, il s'agirait de quantifier le nombre de tentatives possibles pour atteindre un objectif donné avant que l'ensemble des opérations de *commitment* possibles ne devienne trop limité pour permettre l'ajustement du scénario.
Une évaluation similaire à la précédente pourrait être menée, mais, plutôt que de simuler des agents se comportant de manière conforme au modèle d'activité, il s'agirait d'utiliser un utilisateur simulé sur le modèle de celui utilisé pour les évaluations de [Si, 2010], en introduisant par exemple des actions aléatoires.

9.3.4 Variabilité des scénarios

La variabilité des scénarios étant l'un des objectifs de notre système de scénarisation, il serait intéressant de quantifier celle qu'offre DIRECTOR. Il serait par exemple possible de générer l'ensemble

des trames scénaristiques possibles à partir d'une situation initiale et d'un objectif scénaristique donnés, puis de générer un scénario pour chacune d'entre elles. Une mesure de la variabilité pourrait être calculée à partir du nombre de scénarios différents et de la distance entre chaque paire de scénarios (voir par exemple l'utilisation de la distance de Levenshtein par [Porteous et al., 2013]), et rapportée au nombre d'objets et d'opérateurs présents dans le domaine. Cependant, il est ici encore difficile de faire la part des choses entre ce qui relève de DIRECTOR et ce qui relève du modèle de causalité lui-même. S'il était possible de quantifier la variabilité offerte par DIRECTOR, il serait alors intéressant de comparer celui-ci avec un autre système de scénarisation, notamment sur la taille des modèles nécessaires pour atteindre une certaine variabilité dans les scénarios. En particulier, la question suivante se pose : *"A partir de quelle taille de domaine devient-il rentable d'utiliser un système comme DIRECTOR plutôt que de définir explicitement et exhaustivement l'ensemble des scénarios ?"*

9.3.5 Cohérence des scénarios

Il serait également nécessaire d'évaluer la cohérence des scénarios générés par DIRECTOR, et des scénarios effectifs qui en résultent dans la simulation. Par cohérence des scénarios nous entendons ici la cohérence des comportements des personnages virtuels et des systèmes techniques, compte tenu de leurs motivations et de leurs règles de fonctionnement apparentes.
Pour [Riedl and Young, 2005], cette cohérence peut être évaluée au travers de mesures de la compréhension par des lecteurs ou spectateurs des liens de causalité qui relient les événements d'une histoire et les différentes motivations exprimées par les personnages. Ils proposent pour cela un protocole d'évaluation basé sur le modèle QUEST, décrit dans la section 3.3.
Il serait ainsi possible de comparer la cohérence d'un scénario généré à partir d'opérateurs transposant simplement les actions du modèle de domaine (comme le domaine `Fake` décrit précédemment) et d'un scénario généré à partir d'opérateurs de prédiction, ou encore de comparer un scénario généré à partir d'opérateurs d'ajustement classiques, et un scénario généré à partir d'opérateurs d'ajustement "incohérents", c'est à dire qui changent directement des états de la simulation. Ces scénarios pourraient être présentés aux sujets sous la forme de textes, ou encore de vidéos de la simulation.
Le problème se pose toujours quant à la distinction entre ce qui a trait à l'explicabilité qui est assurée par DIRECTOR, et ce qui relève de la validité écologique des modèles utilisés. De plus, il est nécessaire que les sujets de l'expérience aient suffisamment de connaissances dans le domaine pour pouvoir inférer les motivations des personnages virtuels à partir de leurs actions, et les règles de fonctionnement des systèmes techniques à partir de leurs réactions.

Quatrième partie

Discussion et conclusion

Chapitre 10

Discussion

Sommaire

La problématique de nos travaux était de concevoir un système de scénarisation qui permette de concilier le contrôle du scénario avec la liberté d'action de l'utilisateur, la cohérence des comportements présentés, et l'adaptabilité du système.
Nous avons proposé le modèle SELDON, un modèle pour la scénarisation qui offre à la fois les avantages des approches émergentes centrées sur les personnages et des approches centrées sur le scénario. SELDON permet d'adapter dynamiquement le déroulement des événements d'un environnement virtuel en influençant de manière indirecte le comportement d'un ensemble d'entités autonomes par le biais d'ajustements.
Ces ajustements sont calculés par le moteur DIRECTOR, qui utilise des techniques de planification pour générer des scénarios prédictifs à partir des modèles qui sous-tendent la simulation.
Dans ce chapitre, nous présentons tout d'abord les points forts de nos propositions, puis nous discutons de leurs limites. Enfin, nous proposons un ensemble de perspectives pour la suite de ce travail.

10.1 Points forts

L'originalité de notre proposition réside dans l'association de la planification de scénarios avec un environnement virtuel évoluant de manière autonome. Cette autonomie encourage la réactivité de l'environnement face aux actions de l'utilisateur et la variabilité des scénarios, tandis que la planification permet d'assurer leur pertinence.
La génération d'opérateurs de prédiction à partir des modèles de la simulation garantit la cohérence des comportements des personnages virtuels et des systèmes techniques au sein du scénario. Il ne s'agit cependant pas simplement de *planification intentionnelle*, telle que proposée par [Riedl and Young, 2010], [Haslum, 2012] ou [Teutenberg and Porteous, 2013] : la prédiction considère les actions de tous les agents — à travers le système de tour par tour — et l'ensemble du système technique — *via* la vérification systématique des comportements activables — et permet ainsi de s'assurer que les événements qui vont se dérouler en parallèle de ceux nécessaires au scénario ne mettront pas ce dernier en échec.
Nous considérons que les points forts de nos contributions résident en particulier dans les aspects suivants :

- la cohérence des scénarios,
- la liberté et la capacité d'action de l'utilisateur,
- les différents degrés de contrôle des scénarios,
- l'équilibre entre variabilité et pertinence des scénarios,
- l'adaptabilité du système.

10.1.1 Cohérence des scénarios

Les personnages virtuels qui peuplent l'environnement évoluent de manière autonome à partir des modèles d'activité et de domaine, de même que les systèmes techniques simulés. Les ajustements réalisés par le module de scénarisation sont conçus de sorte à ne pas aller à l'encontre de ces modèles : la gestion du *late commitment* assure que les états indéterminés soient précisés dès lors qu'ils donnent lieu à une alternative possible entre plusieurs comportements ou actions, les *happenings* sont choisis pour pouvoir être déclenchés sans qu'il y ait besoin de les justifier, et les contraintes d'occurrence ne peuvent être spécifiées que dans la mesure où l'événement probabiliste auquel elles se rattachent est possible. Les scénarios qui en résultent sont donc explicables en regard des modèles de domaine et d'activité.
De plus, l'utilisation du langage ACTIVITY-DL, issu de langages directement utilisés par les ergonomes, permet de créer des modèles d'activité qui reflètent la complexité des comportements de terrain, notamment dans la prise en compte des contextes de réalisation des tâches et des déviations des procédures prescrites. De même, le langage DOMAIN-DL permet de représenter des systèmes techniques complexes. L'expressivité offerte par ces langages contribue ainsi à la validité écologique des modèles qui les utilisent, et donc à celle de l'environnement virtuel.

10.1.2 Liberté et capacité d'action de l'utilisateur

Etant donné que la simulation évolue de manière autonome à partir des modèles, il est possible pour l'utilisateur de réaliser n'importe quelle action qui soit permise par ces modèles, et d'obtenir une réaction cohérente de la part des personnages virtuels et des systèmes techniques simulés.
De plus, cette liberté d'action ne met pas nécessairement en péril le scénario prévu par DIRECTOR. En effet, les scénarios prédictifs ne concernent qu'un sous-ensemble d'agents et d'objets de l'environnement virtuel, dont DIRECTOR suit l'évolution pour évaluer le bon déroulement du plan ; le reste des agents et des objets peuvent évoluer comme bon leur semble tant que cela ne modifie pas les actions et comportements attendus pour le scénario. Ainsi, si le scénario souhaité ne concerne qu'un personnage virtuel donné, l'utilisateur pourra interagir avec les autres personnages sans que

cela ne force DIRECTOR à calculer un nouveau scénario : il pourra ainsi discuter avec un conducteur de camion sur un poste de chargement pendant que s'annonce une fuite sur le poste voisin, ou jouer avec un enfant pendant qu'un autre choisit la bille avec laquelle il s'étouffera.

10.1.3 Degrés de contrôle des scénarios

Les objectifs scénaristiques pris en compte par DIRECTOR pour le contrôle des scénarios peuvent être définis à plus ou moins haut niveau.
D'une part, la représentation ontologique des entités avec DOMAIN-DL et CAUSALITY-DL permet, de par les liens de subsomption entre événements, de spécifier des situations prescrites ou proscrites allant d'événements très concrets (par exemple "Chute d'un bébé de la table à langer") à plus abstraits ("Chute d'un enfant", voire "Accident domestique"). La modélisation des espaces de scénarios d'intérêt à différents niveaux de granularité dans CAUSALITY-DL permet également de préciser les relations de causalité qui sont signifiantes pour une application donnée, et, associées à la spécification de situations prescrites, d'opérer un contrôle sur les événements apparaissant tout au long du scénario, en prenant en compte un nombre plus ou moins important de points clés dans les trames scénaristiques.
D'autre part, les contraintes sur les propriétés globales du scénario permettent de spécifier des objectifs de haut niveau qui ne soient pas directement liés aux situations. Pour l'instant, des fonctions d'évaluation ont été implémentées pour trois propriétés : la complexité, la gravité et la crédibilité d'un scénario. Il serait possible d'en définir de nouvelles, que ces propriétés soient indépendantes du domaine et uniquement fonction de la structure de la trame scénaristique (par exemple, le nombre de barrières à lever, qui pourrait être une mesure du conflit), ou qu'elles reposent sur des annotations des éléments du modèle de causalité.

10.1.4 Variabilité et pertinence des scénarios

C'est également cette possibilité de définir les espaces de scénarios d'intérêt à différents niveaux de granularité qui permet d'atteindre un équilibre entre la variabilité des scénarios et leur pertinence. L'émergence des scénarios à partir des modèles de domaine et d'activité et de l'évolution autonome de la simulation permet en effet d'obtenir des scénarios variés pour un effort réduit, en comparaison des systèmes qui se basent sur des graphes multi-linéaires de scénarios. Une simulation purement émergente n'offre cependant pas de garanties sur la pertinence des situations qui surviennent suite aux interactions de l'utilisateur avec l'environnement virtuel.
Il est donc possible de spécifier à l'aide de CAUSALITY-DL un espace de scénarios d'intérêt. Plus cet espace sera défini de manière fine et plus l'on se rapprochera du niveau de contrôle offert par les systèmes basés sur des graphes multi-linéaires, tout en gardant la flexibilité offerte par le langage et la façon dont il est interprété par DIRECTOR : d'une part car il est possible de spécifier dans un même modèle des événements ou des sous-graphes d'événements qui ne soient pas reliés entre eux, et d'autre part car les trames scénaristiques qui en résultent peuvent être instanciées et complétées différemment en fonction des objets et des agents qui sont présents dans l'environnement.

10.1.5 Adaptabilité du système

De par le choix d'une scénarisation extrinsèque, le système de scénarisation proposé n'est pas spécifique aux modules de simulation du monde et de simulation des personnages autonomes utilisés pour l'environnement virtuel, et pourrait ainsi être intégré à d'autres plateformes.
De plus, DIRECTOR, tout comme les modules de simulation du monde et des personnages virtuels, est générique et opère à partir d'un ensemble de modèles du contenu scénaristique. Il peut donc, via

un changement de modèles, servir à d'autres applications que celles présentées dans ce mémoire. En particulier, les modèles de causalité utilisés ici en tant qu'objectifs scénaristiques statiques étaient issus de modèles réalisés lors d'analyses de risques, mais CAUSALITY-DL pourrait être utilisé pour exprimer des espaces de scénarios qui ne soient pas nécessairement liés à des accidents. Il pourrait par exemple être utilisé pour représenter des structures narratives classiques, en assimilant les victoires du protagoniste à des événements, et les antagonistes à des barrières.
Les modèles eux-mêmes peuvent être réutilisés en totalité ou en partie : l'exemple `Arakis-Full` utilisé pour l'évaluation informatique dans la section 9.1 étend le domaine `Arakis-Basic` par l'ajout de tâches dans le modèle d'activité, et l'ajout d'un ensemble de comportements accidentels liés aux objets dans le modèle du domaine.
Cette adaptabilité pourrait être poussée encore plus loin, puisque notre proposition permettrait de modifier les modèles de la simulation de manière dynamique, durant les sessions d'utilisation de l'environnement virtuel. En effet, si la génération des opérateurs de prédiction est actuellement réalisée à l'initialisation, rien n'empêche de mettre à jour les modèles et de générer ces opérateurs à nouveau avant de planifier un nouveau scénario. Les modifications dynamiques des règles de comportement des objets sont d'ailleurs déjà permises par le moteur de simulation du monde de la plateforme HUMANS, présenté dans l'annexe B.

10.2 Limites

Nous constatons cependant un certain nombre de limites à ces propositions, particulièrement en ce qui concerne l'utilisation de la planification, comme il a été constaté dans le chapitre 9. Ces limites portent sur le temps nécessaire à la planification, la replanification et la prise en compte des objectifs scénaristique durant ces phases. Elles portent également sur la génération des opérateurs de prédiction, sur la dépendance de DIRECTOR par rapport au contenu des modèles, et sur la gestion des cas où il est impossible de planifier un scénario.

10.2.1 Temps de planification

Les temps de génération constatés lors de l'évaluation présentée dans le chapitre 9 sont suffisamment élevés pour remettre en question l'utilisation de techniques de planification pour la génération dynamique de scénarios. Ces temps sont d'autant plus problématiques que les domaines sur lesquels cette génération a été évaluée restent relativement simplifiés par rapport aux domaines modélisés pour les cas d'application réels, notamment en termes de nombre de tâches ou d'ajustements possibles (voir par exemple dans la section 9.2).
Le passage à l'échelle est donc très limité, ce qui est une difficulté notoire du domaine de la planification où les moteurs sont souvent évalués et comparés sur des problèmes "de laboratoire", comme *blocks world*, et peu sur des problèmes liés à des cas d'application réels.
Cependant, les spécificités de notre approche offrent quelques pistes pour réduire le temps nécessaire à la planification des scénarios, qui pourraient permettre de les générer, si ce n'est en temps interactif, tout du moins dans un temps acceptable pour nos besoins de scénarisation dynamique.
Nous avons déjà évoqué dans la section 9.1.5 la possibilité de procéder à l'instanciation des opérateurs en amont de la planification, et de limiter cette instanciation aux combinaisons d'objets respectant les relations statiques définies dans le modèle du domaine. Cela pourrait faire gagner un temps considérable, particulièrement lorsqu'il s'agit d'effectuer plusieurs tentatives de planification successives pour compléter une trame scénaristique.
Une autre spécificité de DIRECTOR qu'il serait possible d'exploiter est le fait que, dans certains cas, seule une partie des agents et des objets de l'environnement virtuel est effectivement impliquée dans la réalisation du plan, et que dans ce cas les actions des autres agents et les comportements des autres

objets, qui ne sont pas reliés à la situation finale, sont retirés du scénario durant la phase de suppression des opérateurs abstraits ; c'était le cas pour les plans présentés dans les figures 6.5 et 6.6. Or dans certains cas il devrait être possible de proposer des heuristiques dépendantes du domaine permettant de déterminer à l'avance, à partir d'une trame scénaristique donnée, quels sont les agents ou objets qui ne seront pas présents dans le scénario. Par exemple, on pourrait considérer que les actions d'un conducteur situé à un bout du dépôt pétrolier n'auront pas d'incidence sur celles d'un second conducteur situé à l'autre bout. Il serait alors possible d'utiliser de telles heuristiques pour réduire l'ampleur de la tâche de planification, sans pour autant modifier le scénario qui en résulte.
Nous pourrions également nous inspirer des travaux de [Teutenberg and Porteous, 2013] en couplant le moteur de planification à un système multi-agent et en chargeant chaque agent de calculer de lui-même l'opérateur de prédiction qui serait applicable pour l'agent lui correspondant dans l'environnement virtuel.

10.2.2 Replanification et instanciation des trames scénaristiques

En parallèle de la réduction du temps de planification, la replanification serait également à améliorer. Dans le cadre de DIRECTOR, la replanification d'un scénario est nécessaire dans deux cas : soit car l'utilisateur met en péril le scénario existant, soit car de nouveaux objectifs scénaristiques sont reçus.
Or la phase d'instanciation des trames scénaristique est actuellement réalisée de manière très simpliste, à partir des correspondances entre variables spécifiées au niveau des liens entre les points clés, sans prendre en compte l'état courant du monde ou le déroulement précédent des événements. Il serait toutefois possible d'instancier cette trame à partir des événements déjà activés, plutôt que de repartir de zéro, voire de considérer la progression dans les différentes trames scénaristiques comme un critère de choix lors de la phase de sélection.
De même, plutôt que de répéter toute la phase de génération d'un scénario (sélection, instanciation et complétion d'une trame scénaristique), il serait plus pertinent de mettre en place des mécanismes de réparation de scénario, en cherchant tout d'abord à réparer le plan entre les points clés en échec, et en ne sélectionnant une nouvelle trame scénaristique que si cette réparation est impossible.

10.2.3 Prise en compte des objectifs lors de la planification

La séparation des phases de sélection, instanciation et complétion des trames scénaristiques a également pour effet de limiter la prise en compte des objectifs scénaristiques. En effet, pour l'instant les contraintes sur les situations et sur les propriétés du scénario ne sont considérées que durant la phase de sélection de la trame.
Les situations prescrites et proscrites sont donc limitées aux événements présents dans le modèle de causalité, ce qui s'explique par le fait que ce modèle est censé contenir l'ensemble des événements d'intérêt pour une application donnée : une situation suffisamment signifiante pour être prescrite ou proscrite devrait donc figurer dans ce modèle. De même, le calcul des propriétés du scénario (complexité, gravité et crédibilité) est effectué à partir de la trame scénaristique, ce qui peut être justifié par les mêmes raisons.
Cependant, ces contraintes ne sont pas prises en compte durant la phase de planification pour la complétion de la trame scénaristique. En particulier, il est possible que des situations proscrites apparaissent dans le scénario final, alors même qu'elles avaient été écartées de la trame scénaristique au préalable.
Il serait possible de prendre en compte ces situations proscrites comme des barrières dans l'algorithme de complétion entre les points clés, en ajoutant systématiquement leur négation au but transmis au moteur de planification. Cependant cela reviendrait à leur affecter systématiquement une désirabilité de -1, ce qui n'est pas nécessairement le cas au départ.

Il semblerait plus pertinent de prendre en compte ces contraintes directement au niveau de la planification, en les représentant comme des contraintes faibles ou fortes de PDDL3.0 [Gerevini and Long, 2005]. De même, le calcul de la crédibilité devrait prendre en compte les choix faits au niveau des contraintes d'occurrence par rapport aux probabilités originelles des événements qu'elles contraignent. Cela pourrait être pris en compte au niveau de la planification en définissant une métrique de crédibilité liée à ces opérateurs, et en cherchant à la minimiser ou la maximiser selon les cas.
De manière générale, le processus de génération de scénario gagnerait à être rendu itératif plutôt que linéaire, sur le plan de la prise en compte des objectifs scénaristiques tout du moins. Cela permettrait ainsi de remettre en question le choix d'une trame scénaristique si le scénario, une fois complété, s'avère ne pas répondre de manière satisfaisante aux objectifs.

10.2.4 Génération des opérateurs de prédiction

La génération des opérateurs de prédiction à partir des modèles du domaine et de l'activité pourrait elle aussi être améliorée sur un certain nombre de points.
Tout d'abord, les algorithmes de génération des opérateurs d'action considèrent pour l'instant le modèle d'activité de manière séquentielle. Or la représentation hiérarchique des tâches permet de modéliser des comportements plus riches, notamment le suivi de plusieurs tâches mères en parallèle, ou de plusieurs arbres d'activité indépendants, permettant aux personnages virtuels de changer de tâche principale si l'état du monde évolue. Par exemple, si un feu se déclenche, la tâche principale de l'agent ne sera plus "Effectuer le chargement du camion" mais deviendra "Eteindre le feu". Cela nécessiterait l'ajout de tâches abstraites permettant de continuer ou d'interrompre une tâche mère en cours — toutefois cela aurait pour effet d'augmenter encore le nombre d'opérateurs nécessaires pour prédire l'activité à partir d'un même domaine. De même, ces algorithmes ne considèrent pas pour l'instant qu'une tâche est susceptible d'échouer, ce qu'il serait possible de prendre en compte avec un mécanisme similaire.
Le fait de considérer les actions en "tour par tour" au niveau du plan permet d'assurer que les actions de tous les agents seront considérées, cependant les actions se déroulant dans l'environnement virtuel n'ont pas toutes la même durée, et il serait possible pour un agent d'effectuer plusieurs actions pendant qu'un autre n'en fait qu'une, ce qui rendrait la prédiction erronée. De même, les effets des comportements duratifs des comportements du domaine sont simplifiés en effets instantanés au niveau des opérateurs.
Certains moteurs de planification, compatibles avec des versions de PDDL ultérieures à la 1.1, permettent toutefois de considérer la temporalité dans la planification. D'autres fonctionnalités pourraient s'avérer pertinentes pour la génération des opérateurs de prédiction : l'utilisation d'axiomes (ou "prédicats dérivés") pour traduire les liens de subsomption entre événements, ou d'événements PDDL pour appliquer systématiquement les comportements activables sans devoir passer par des opérateurs. La représentation des scénarios pourrait être enrichie de manière similaire ; nous en discutons dans la section 10.3.1.

10.2.5 Dépendance des modèles

Les performances de DIRECTOR dépendent directement de la qualité des modèles de domaine, d'activité et de causalité qu'il utilise.
Ces modèles posent un certain nombre de difficultés lors de leur conception. Certaines sont liées directement aux langages : par exemple, la représentation des règles de comportement dans DOMAIN-DL ne permet pas, en l'état, leur définition par les experts sans l'appui d'un informaticien. D'autres sont liées aux méta-modèles : par exemple, les constructeurs d'ACTIVITY-DL étant formalisés à l'aide de couples de relations d'Allen, il est nécessaire que l'ensemble des couples définis pour un constructeur personnalisé soit calculable.

Du fait de l'ambiguïté de ces langages sur certains aspects, une certaine subjectivité est présente lors de la phase de conception, notamment pour ce qui concerne le modèle de causalité : par exemple, certains éléments pourront être représentés comme des barrières dans un cas ("Déclenchement de la sonde"), et comme des événements dans d'autres cas ("Non fonctionnement de la sonde").
Le facteur limitant reste cependant la cohérence des modèles entre eux : le modèle d'activité doit être réalisable d'après les règles de comportement du modèle de domaine ; les relations de subsomption entre événements doivent être cohérentes avec les règles d'activation de ces événements ; les relations de causalité entre événements doivent correspondre à un ensemble de comportements du modèle du domaine.
Une problématique particulière de DIRECTOR est qu'il nécessite de plus que le modèle de causalité soit complet : si certaines barrières ne sont pas indiquées dans le modèle, alors elles ne seront pas considérées lors de la complétion de la trame scénaristique, et la planification échouera. Par exemple, la réalisation de l'événement "Eclaboussure dans les yeux d'un agent" est soumise à la levée de la barrière "Port des lunettes de protection". Si cette barrière n'est pas indiquée dans le modèle, sa négation ne sera pas ajoutée au but considéré pour la phase de planification vers le premier point clé de la trame menant à l'événement, et rien ne sera fait pour empêcher l'agent de mettre ses lunettes de protection, ce qui rendra impossible la génération ultérieure d'un plan vers le dernier point clé.
Au final, les premières versions des modèles sont rarement complètes, et nécessitent un certain nombre d'itérations avant qu'il soit possible d'exécuter la simulation et de générer des scénarios. Une piste de solution serait d'associer aux outils auteurs permettant d'éditer les modèles un ensemble de mécanismes de vérification de ces derniers (voir par exemple [Rempulski, 2013].

10.2.6 Cas où la planification est impossible

Enfin, l'une des principales limites de DIRECTOR se situe dans les cas où il est impossible de générer un plan permettant d'atteindre les objectifs scénaristiques donnés par TAILOR. Ce problème est intrinsèquement lié à l'utilisation du *late commitment* : plus la simulation avance, et moins DIRECTOR aura d'ajustements possibles sur le scénario.
Il est alors essentiel de chercher à limiter le nombre d'ajustements effectués pour réaliser un scénario donné. On retrouve alors la question du compromis entre optimalité et performances de la planification, déjà évoquée dans la section 9.1.5. Le choix d'une planification non-optimale amène en effet à l'exécution d'un ensemble d'ajustements qui ne soient pas nécessaires pour la réalisation du scénario.
Sur le plan narratif, la présence d'ajustements "inutiles" n'est toutefois pas inintéressante. Prenons l'exemple du plan généré par le moteur FF sur l'exemple de la station service, présenté dans la figure E.1 de l'annexe E. Dans ce scénario, un appel téléphonique est déclenché au départ, créant une étincelle qui reste sans conséquence puisqu'il n'y a pas encore de fuite d'essence à ce moment. On peut considérer ici que ce premier appel a pour effet de créer du suspense en préfigurant ce qui risque d'arriver, un peu à la manière de la série de films *Destination Finale* où le spectateur assiste à la mise en place graduelle de situations qui vont, par effet domino, aboutir à la mort accidentelle des personnages.
Cependant, si ces ajustements "inutiles" ne sont pas problématiques dans le cas de *happenings* ou de contraintes d'occurrence, le raffinement superflu de certains états ferme inutilement des portes pour la suite de la simulation. Il faudrait dès lors chercher à minimiser le nombre d'opérateurs de type *late commitment* dans les plans, par exemple en définissant des coûts pour les opérateurs de planification qui permettraient de minimiser le nombre d'ajustements. Ce mécanisme pourrait être utilisé alternativement pour tenter de minimiser la longueur du scénario en assignant des coûts aux actions concrètes.
Il serait également possible d'étendre les possibilités d'ajustement, en permettant l'ajout de nouveaux objets, soit via les mécanismes de *late commitment* (par exemple, l'ajout d'un outil dans un casier n'ayant pas encore été ouvert), soit via les mécanismes de *happenings* (par exemple, en faisant

arriver un nouvel agent dans la scène). Les contraintes d'occurrence pourraient également être étendues, en particulier pour moduler les actions erronées des personnages, qui pour l'instant sont considérées comme systématiques.
Dans le cas où DIRECTOR ne parvienne toutefois pas à trouver un plan satisfaisant, un mécanisme de négociation devrait être mis en place avec TAILOR, afin de permettre le relâchement des contraintes.

10.3 Perspectives

En parallèle des améliorations visant à résoudre les problèmes venant d'être évoqués, nous envisageons un certain nombre de pistes pour étendre nos propositions. Ces pistes concernent l'amélioration de la représentation des scénarios, l'ajout d'un modèle de la cognition et d'un modèle de l'utilisateur, la prise en compte de différents niveaux de présentation du scénario, et enfin l'évaluation de l'intérêt narratif des scénarios en complément des propriétés déjà considérées.

10.3.1 Représentation des scénarios

Une première piste serait d'améliorer l'**expressivité des formalismes de représentation des scénarios**, en particulier la **représentation hiérarchique du scénario** et la prise en compte de la **temporalité**, à la fois au niveau des espaces de scénarios modélisés en CAUSALITY-DL et du scénario généré lui-même.
En effet, si CAUSALITY-DL permet d'exprimer des liens de subsomption entre événements (par exemple, une "Chute dans l'escalier" et une "Chute de la table à langer" sont deux types de "Chute", qui est un type d'"Evénement Accidentel"), la représentation des scénarios sous forme de plans partiellement ordonnés ne permet pas d'**exprimer au niveau des opérateurs** :

- ni la **subsomption**, par exemple `(avaler manon bille)` et `(avaler manon caillou)` héritent de `(avaler manon petit-objet)`;
- ni la **décomposition**, par exemple `(aller-chercher-biberon anita)` se décompose en :
  - `(aller anita frigo)`,
  - `(ouvrir anita frigo)`,
  - `(prendre anita biberon frigo)`,
  - `(fermer anita frigo)`.

Utiliser un algorithme de planification hiérarchique, comme DPOCL [Young and Moore, 1994], pourrait permettre de prendre en compte ces relations, et de manipuler ainsi le scénario à plus haut niveau. Cela permettrait notamment d'identifier des phases dans le scénario, par le biais d'**opérateurs abstraits** correspondant chacun à un ensemble d'actions des agents et de comportements du système technique. Ce niveau d'abstraction supplémentaire simplifierait grandement la **replanification** du scénario en permettant de ne modifier que certaines phases. Il permettrait également d'avoir une plus grande marge de manœuvre par rapport aux actions de l'utilisateur, en intégrant dans le plan des actions abstraites pouvant être réalisées de différentes manières par celui-ci sans mettre en péril le scénario global.
De plus, la **représentation de la temporalité** dans CAUSALITY-DL permettrait d'augmenter la couverture des scénarios pouvant être considérés. Certains scénarios accidentels, identifiés lors d'analyses de risques servant à définir l'espace de scénarios d'intérêt, reposent en effet sur une **synchronisation** très précise entre plusieurs événements. Un exemple, mentionné lors du recueil d'expertise sur le cas d'application des dépôts pétroliers, est le cas où un conducteur ayant fini son chargement démarre son camion au moment même où un second conducteur déclenche accidentellement une fuite sur le poste de chargement voisin, ne laissant pas à ce dernier le temps de sécuriser la zone.
La prise en compte de la temporalité en CAUSALITY-DL devrait ainsi permettre de décrire la **synchronisation** entre :

- **deux événements liés par une porte ET**, qui peuvent être :
  - **synchrones**, par exemple “Présence d’une source d’ignition” et “Epandage de produit inflammable” menant à “Formation d’un feu de nappe”,
  - **asynchrones**, par exemple “Retour d’une citerne non-vide” et “Arrivée d’un nouveau chauffeur” menant à “Citerne pleine non identifiée”,
- **une barrière et l’événement qu’elle prévient**, pouvant également être :
  - **synchrones**, par exemple “Déclenchement de la sonde anti-débordement” par rapport à “Non-arrêt du système de comptage”,
  - **asynchrones**, par exemple “Contrôle de la vacuité” par rapport à “Citerne pleine non identifiée”.

En parallèle, l’utilisation de **planification temporelle** pour la génération du scénario permettrait de considérer les durées des différentes actions afin de prendre en compte ces différents cas de figure, et notamment de déclencher les événements exogènes au moment précis où ils sont nécessaires — et non avec une approche “tour par tour” comme c’est actuellement le cas.

10.3.2 Modèle de la cognition

Une amélioration nécessaire pour rendre compte de scénarios plus complexes sur le plan humain serait la proposition et l’intégration dans notre architecture d’un **modèle de la cognition**.
En effet, si le modèle de l’activité utilisé décrit un ensemble de connaissances sur l’activité des personnages virtuels, il n’explicite pas pour autant leurs **processus cognitifs**. Notamment, les connaissances modélisées à l’heure actuelle ne permettent pas de savoir comment les personnages virtuels interprètent ce modèle d’activité : comment ils évaluent l’utilité de chaque action du modèle et pondèrent ces utilités en fonction de leurs buts, à quel horizon ils planifient leurs actions, etc. Selon l’architecture cognitive utilisée, les décisions prises par un personnage virtuel, à partir du même modèle d’activité, pourront être différentes. Ces connaissances sur l’interprétation du modèle d’activité sont actuellement **encodées de manière implicite** dans l’algorithme qui traduit le modèle représenté en ACTIVITY-DL en opérateurs de prédiction PDDL.
Abstraire les processus de prise de décision implémentés par **différentes architectures cognitives** nous permettrait ainsi d’interfacer notre système de scénarisation avec des moteurs de personnages virtuels basés sur des approches différentes : le module utilisé actuellement, REPLICANTS [Lhommet, 2012], utilise Soar [Laird et al., 1987], mais on pourrait imaginer utiliser une autre architecture, comme par exemple PsychSim [Pynadath and Marsella, 2004] qui est basée sur des problèmes de Markov partiellement observables (POMDPs).
Or, différentes architectures cognitives prennent en compte **différentes composantes “humaines”** : émotions, personnalité, relations sociales, adoption et révision dynamique des buts, etc. Des travaux précédents menés par [Edward, 2011] sur l’architecture MASVERP ont notamment porté sur la prise en compte du modèle de contrôle contextuel **COCOM** [Hollnagel, 1994]. Ce modèle définit quatre **modes de contrôle**, qui représentent l’état dans lequel peut se trouver un agent en fonction de la pression temporelle qu’il subit, et vont influer sur ses capacités de planification et prise de décision. Un agent qui sera dans un mode “stratégique” va par exemple anticiper les préconditions des tâches du modèle d’activité en allant chercher tous les outils nécessaires avant de se déplacer pour réaliser la séquence de tâches, tandis qu’un agent en mode “tactique” ira récupérer ces outils au moment où il en aura besoin. L’intégration d’un modèle de la cognition permettrait la prédiction et la prise en compte dans le scénario de tels comportements.

10.3.3 Modélisation de l’utilisateur

Le système serait également grandement amélioré par l’intégration d’un meilleur **modèle de l’utilisateur**. Dans la proposition actuelle, l’utilisateur est considéré lors de la construction du scénario

comme un personnage virtuel dotés de caractéristiques “moyennes”. Il est donc fréquent que son comportement réel s’éloigne de ce comportement standard qui est attendu de lui. Pour les scénarios qui reposent fortement sur les actions de l’utilisateur et non sur celles des personnages virtuels, il est ainsi nécessaire de replanifier fréquemment le scénario.
Une meilleure modélisation du comportement de l’utilisateur permettrait d’adopter une approche **proactive**, en anticipant davantage les déviations possibles du scénario souhaité pour éviter d’avoir à le recalculer systématiquement. Cela permettrait également la construction de scénarios plus intéressants, car en prédisant le type d’erreurs que peut commettre l’utilisateur, il est alors possible d’amener ce dernier à des **situations propices à ces erreurs**, permettant ainsi de lui montrer les conséquences de ses propres erreurs plutôt que de se reposer sur celles des personnages virtuels.
Nous envisageons trois pistes principales pour améliorer le modèle de l’utilisateur : la prise en compte des **traces d’activité**, l’intégration de **motivations propres à l’utilisateur**, et la modélisation de ses **croyances**.
Le modèle SELDON prend en entrée un ensemble de **traces d’activité** de l’utilisateur. Ces traces sont analysées et interprétées par TAILOR, le module de génération de situations d’apprentissage, pour déterminer des objectifs scénaristiques adaptés au profil et à l’historique de l’utilisateur. Cependant, ces traces d’activité ne sont pas utilisées par DIRECTOR. Certains éléments du profil de l’utilisateur sont certes utilisés pour ajuster le modèle de l’agent qui lui est associé dans le scénario, mais cette paramétrisation reste très basique : il s’agit principalement de classifier celui-ci comme étant un novice ou un expert. Des liens plus étroits avec le **module de suivi de l’apprenant** permettraient à DIRECTOR d’améliorer la prédiction des actions de l’utilisateur. En effet, de même que les personnages virtuels peuvent ne pas suivre à la lettre le modèle d’activité selon le modèle de la cognition qui est utilisé, l’utilisateur ne respecte pas nécessairement les préconditions des tâches qui y sont spécifiées. Un utilisateur peut ainsi ignorer une précondition contextuelle, ou ne pas respecter les préconditions favorables lors d’une alternative entre deux tâches. La prédiction de son comportement faite par DIRECTOR étant basée sur des opérateurs de planification déterministes construits à partir du modèle d’activité, elle ne permet pas de prévoir ces cas. Le module de suivi de l’apprenant MONITOR (décrit dans l’annexe B.4) intègre par contre un mécanisme de reconnaissance de plans qui permet un suivi et une prédiction plus fine.
Une autre piste d’amélioration serait la prise en compte des **spécificités de l’utilisateur** de l’environnement virtuel, en cela qu’il est différent des personnages virtuels autonomes mais également du même utilisateur en situation réelle, en particulier au niveau de ses **motivations**. Le système actuel n’attribue en effet qu’une seule motivation à l’utilisateur : la réalisation de la procédure (par exemple “surveillance des enfants”). Or, bien que l’on recherche la validité écologique de l’environnement pour tendre vers un apprentissage situé, l’utilisateur se comporte différemment dans la situation simulée et dans la situation réelle. Il serait ainsi pertinent de lui attribuer des motivations propres non pas au rôle qu’il joue dans la simulation, mais à l’utilisation de la simulation même — explorer l’environnement, discuter avec les différents personnages, etc. —, comme c’est le cas dans le système Thespian [Si, 2010].
Enfin, il serait possible de modéliser explicitement les croyances de l’utilisateur, non pas au niveau pédagogique en termes de connaissances ou de compétences, mais en termes de **croyances sur les états de l’environnement**. Il serait pour cela nécessaire d’intégrer un modèle de la perception, afin d’inférer à la fois les croyances de l’utilisateur sur les états des objets en fonction des comportements observés, et les croyances sur les motivations et autres états internes des personnages virtuels en fonction des actions perçues. Cela permettrait notamment de laisser davantage de marge pour le *late commitment*. La modélisation des croyances de l’utilisateur permet de recentrer la scénarisation sur ce dernier, qui est son principal destinataire. Dans le domaine de la narration interactive, on parlera alors d’*experience management*, plutôt que de *drama management*, c’est à dire que l’on s’intéresse aux propriétés de l’expérience de l’utilisateur et non plus aux propriétés de l’histoire elle-même [Riedl et al., 2008b]. Dans le système IN-TALE, les objectifs ne sont ainsi plus définis en termes d’états de l’environnement mais de croyances de l’utilisateur, par exemple `(knows player (has Mohammed`

```
bomb1)).
```

10.3.4 Discours et récit

Jusqu'ici, nos travaux sur la scénarisation se sont focalisés — pour reprendre le vocabulaire cinématographique — sur la génération et la gestion du **screenplay** (*"ce qu'on filme"*), qui décrit les événements et les actions ayant lieu dans l'environnement virtuel. Nous n'avons pas abordé la question du **shooting script** (*"comment on le filme"*), qui définit la façon dont ces éléments sont mis en scène : décors, caméras, lumières, musique, placement des personnages, narrateur...
Or l'utilisation d'un environnement virtuel offre de nombreuses possibilités pour la restitution d'une séquence d'événements. En narratologie, notamment suite aux travaux de [Propp, 1928], on distingue généralement **trois composantes de la narration** [Cheong, 2007] :

- la **fabula** : la séquence d'événements prenant place dans le monde de l'histoire, dans un ordre chronologique ;
- le **sjužet** : une séquence d'événement extraite de la *fabula*, d'après un point de vue particulier et dans un ordre particulier ;
- le **discours** : le résultat de la présentation du *sjužet* au travers d'un media particulier.

On peut traduire *fabula* et *sjužet* respectivement par **histoire** et **récit** [Swartjes, 2010].
L'approche choisie pour nos environnements virtuels, qui est de laisser l'utilisateur **naviguer librement** — notamment dans le cas d'environnements immersifs à l'échelle 1 où le point de vue est calqué sur le point de vue réel de l'utilisateur — et de se focaliser sur une scénarisation intra-diégétique, limite le contrôle à celui de l'**histoire**, en restreignant les possibilités pour le système de maîtriser quels événements sont perçus et comment ils sont perçus.
Certains éléments du **discours** peuvent toutefois être manipulés de manière **intra-diégétique**, c'est à dire sans sortir l'utilisateur du monde virtuel dans lequel il est immergé. Il serait ainsi possible de travailler sur une **mise en scène adaptative**, en jouant sur l'ambiance de l'environnement virtuel pour modifier l'expérience de l'utilisateur. Le système pourrait notamment intégrer un modèle d'éclairage interactif [El-Nasr and Horswill, 2003], ou utiliser le *late commitment* pour gérer le placement des personnages et des objets dans l'environnement.
D'autres environnements virtuels pourraient permettre de manipuler davantage le discours, via la **manipulation du point de vue** : dans un environnement avec vue à la troisième personne, le point de vue de l'utilisateur et celui de son avatar (son incarnation dans l'environnement virtuel) ne coïncident plus. L'utilisateur peut alors voir le personnage qu'il contrôle dans l'environnement et est toujours capable de se déplacer librement dans l'environnement, mais le système contrôle les positions et les angles de caméra utilisés pour restituer la scène. Selon l'effet voulu, le système pourra alors choisir d'utiliser un plan large ou un plan serré, en plongée ou en contre-plongée, etc. Une telle manipulation du discours serait par exemple réalisable via l'intégration, comme pour le système Mimesis [Young et al., 2004], d'un *discourse planner* chargé de contrôler les mouvements de caméra et la musique durant l'exécution du scénario dans l'environnement virtuel. Un certain nombre de travaux ont ainsi été réalisés sur le placement dynamique de caméras pour des systèmes de narration interactive : [Courty et al., 2003], [Riedl et al., 2008a], [Jhala and Young, 2010], [Lino et al., 2010].
En restreignant encore davantage la navigation, il serait possible de travailler sur le **récit**, en déterminant quels événements doivent être présentés à l'utilisateur et dans quel ordre ils doivent l'être. S'écarter d'un ordre chronologique permettrait d'introduire des structures narratives comme le **suspense**, avec l'utilisation de *flashbacks* et de *flashforwards* [Cheong, 2007]. Cela pourrait être notamment pertinent pour des applications de formation ou de prise de décision liées à la maîtrise des risques, en présentant tout d'abord un événement accidentel, puis en remontant à la chaîne d'événements qui y a aboutie. Il serait également possible de travailler sur le **point de vue subjectif** avec lequel est présenté le scénario, en se focalisant sur un personnage ou un autre [Porteous et al., 2010].

10.3.5 Calcul de l'intérêt narratif

Il existe de nombreuses manière d'évaluer la qualité d'un scénario, que ce soit en termes de pertinence pédagogique ou d'intérêt narratif. Nous avons centré notre proposition sur le premier cas de figure, avec la prise en compte de trois propriétés des scénarios : leur complexité, leur gravité, et leur crédibilité. Ces propriétés sont intéressantes pour répondre à des demandes en termes de situations pédagogiques, mais ne garantissent pas à elles seules que le scénario sera jugé intéressant par l'utilisateur.
On trouve dans la littérature d'autres critères permettant de juger de l'**intérêt narratif** d'un scénario, et donc de sa capacité à maintenir l'attention et la motivation de l'utilisateur. L'architecture Moe utilise ainsi une mesure de l'**intensité**, assimilée à la progression des connaissances de l'utilisateur sur le monde [Weyhrauch, 1997]. Le système IDTension cherche à faire apparaître des situations de **conflit**, en favorisant les situations où les personnages sont contraints à réaliser des actions contraires à leurs valeurs ou à leurs motivations [Szilas, 2003]. D'autres systèmes, comme [Porteous et al., 2011] ou [Delmas, 2009], se reposent sur des théories narratives classiques en assimilant l'intérêt narratif d'un scénario à son adéquation à une **structure narrative** donnée.
Il serait ainsi pertinent de prendre en compte de telles propriétés dans le choix des trames narratives, afin de favoriser non seulement l'intérêt pédagogique du scénario mais également sa qualité sur le plan narratif.

Chapitre 11

Conclusion

Les travaux présentés dans ce mémoire adressaient la problématique de la scénarisation d'environnements virtuels au travers de quatre objectifs : la liberté d'action de l'utilisateur et sa capacité d'action sur le scénario ; l'ampleur, le caractère dynamique et l'efficacité du contrôle exercé par les concepteurs et/ou formateurs sur ce même scénario au travers du système de scénarisation ; la cohérence des comportements présentés, à la fois au niveau de leur explicabilité au sein de la simulation et de leur validité par rapport à la situation réelle simulée ; enfin, l'adaptabilité du système de scénarisation, nécessaire pour proposer une variabilité des scénarios pour un effort modéré, en veillant à la maintenabilité, à la réutilisabilité et au passage à l'échelle de ces scénarios.
Pour répondre à cette problématique, nous avons proposé le modèle SELDON, un modèle pour la scénarisation d'environnements virtuels basé sur une approche émergente et modulaire. SELDON assure la scénarisation extrinsèque d'un environnement peuplé de personnages virtuels autonomes, où l'utilisateur est libre du choix de ses actions. Partant d'une simulation interactive, il réalise cette scénarisation en deux étapes : dans un premier temps, des objectifs scénaristiques sont déterminés à partir du profil et des traces d'activité de l'utilisateur ; puis un scénario est généré de sorte à atteindre ces objectifs, et exécuté au travers d'ajustements indirects réalisés sur la simulation.
Nous avons réifié ce modèle au travers du moteur DIRECTOR. Plutôt que de contrôler directement l'état du monde et de donner des ordres aux personnages virtuels, DIRECTOR utilise les modèles qui sous-tendent la simulation pour prédire son évolution, et guide cette évolution au travers d'un ensemble d'ajustements qui influencent les réactions des systèmes techniques et les prises de décision des personnages virtuels de manière indirecte, sans nuire à leur explicabilité. DIRECTOR utilise un moteur de planification pour générer des scénarios composés à la fois d'opérateurs de prédiction — dont il va suivre l'instanciation dans la simulation — et d'opérateurs d'ajustement — dont il va déclencher les effets.
Nous avons également proposé un ensemble de langages de représentation du contenu et des objectifs scénaristiques : DOMAIN-DL, ACTIVITY-DL et CAUSALITY-DL. Ces langages sont issus de formalismes utilisés par les experts chargés de concevoir les simulations à partir d'analyses terrain, et permettent de représenter les connaissances de manière modulaire, à différents niveaux d'abstraction et de granularité.
Ces contributions ont été implémentées et ont fait l'objet d'une évaluation informatique. Si elles possèdent un certain nombre de limites, notamment en ce qui concerne la planification en temps interactif des scénarios, nous estimons toutefois que notre approche consistant à associer planification centralisée et simulation émergente permet de s'affranchir de nombreuses limitations classiques des systèmes de scénarisation en termes de liberté d'action, d'adaptabilité ou de cohérence, et qu'elle mérite d'être explorée davantage, en particulier au travers des pistes de solutions proposées dans la discussion.
De manière plus générale, l'une des principales faiblesses de nos travaux est l'absence d'une validation formelle de notre système vis à vis des objectifs fixés quant aux possibilités de contrôle qu'il offre

et à la cohérence des scénarios. Il ne s'agit cependant pas là d'une spécificité liée à nos propositions, mais d'un problème auquel se heurte l'ensemble du domaine de la narration interactive, et, par extension, des travaux liés à la scénarisation d'environnements virtuels.
C'est ici que se situe pour nous la prochaine étape pour ce domaine de recherche : dans la proposition de métriques et de protocoles d'évaluation permettant de valider les systèmes de scénarisation, et surtout dans la proposition d'un formalisme standard de représentation des scénarios, du contenu et des objectifs scénaristiques, qui, tel le PDDL pour le domaine de la planification, permettrait l'échange et la comparaison des résultats, tout en gardant à l'esprit que la spécificité de la scénarisation d'environnements virtuels est tout d'abord la multiplicité des usages auquel elle fait face, et qu'elle ne saurait être évaluée sur des critères purement techniques.

Cinquième partie

Annexes

Annexe A

Glossaire

action Une action est un processus physique ou cognitif, pouvant être mis en œuvre par un agent à l'issue d'une décision. Une action peut être ponctuelle ou durative.

agent Un agent est un objet capable d'interagir de manière autonome avec son environnement, c'est à dire à la fois de le percevoir et d'effectuer des actions dessus. Un agent peut être un utilisateur humain ou un personnage virtuel autonome.

comportement Un comportement correspond à une manière d'évoluer de l'objet ou du système qui en est doté. Un comportement peut être ponctuel ou duratif.

diégèse La diégèse est l'univers d'une œuvre, elle correspond au monde simulé.

discours Le discours est le résultat de la présentation du récit au travers d'un media particulier.

endogène Un événement endogène est la résultante de comportements du domaine, que ceux-ci relèvent d'actions de l'utilisateur, d'actions des personnages virtuels ou de comportements propres au système lui-même.

état Un état est une condition instantanée dans laquelle se trouve un objet ou un système.

événement Un événement est un fait marquant, qui survient à un moment donné. Un événement correspond à une agrégation de changements d'états liés de manière causale, signifiante du point de vue de l'observateur. Un événement peut être ponctuel ou duratif.

exogène Un événement exogène est un événement déclenché de manière externe au système.

extradiégétique Un événement extradiégétique est un événement qui n'appartient pas au domaine simulé, se situant donc en dehors de la diégèse.

formalisme Un formalisme est une symbolisation matérielle (textuelle, graphique, etc.) arbitraire permettant de rendre manipulables des concepts.

histoire L'histoire, ou *fabula*, est la séquence d'événements prenant place dans le monde, dans un ordre chronologique.

intradiégétique Un événement intradiégétique est un événement qui appartient au domaine simulé et se situe donc dans la diégèse.

langage Un langage est un système de signes doté d'une sémantique et d'une syntaxe.

langage formel Un langage formel est un langage défini de sorte qu'il soit dénué d'ambiguïté, par opposition aux langages naturels. Un langage formel peut être spécifié par une grammaire formelle, une expression régulière, un automate, etc. Un langage formel repose sur un ou plusieurs formalismes.

méta-modèle Un méta-modèle est un modèle qui a pour particularité de modéliser d'autres modèles.

modèle Un modèle est un objet permettant de décrire la complexité inhérente à un phénomène, un système, une situation, un domaine, etc., dans un but particulier. Un modèle est écrit dans un langage de modélisation. Un langage de modélisation se concentre sur certains aspects de ce qu'il modélise.

objet Un objet est une entité, concrète ou abstraite, ayant une existence propre, c'est-à-dire pouvant être décrite ou manipulée sans qu'il soit nécessaire de connaître d'autres objets.

objet géométrique Un objet géométrique est un objet possédant une représentation géométrique dans l'environnement virtuel.

récit Le récit, ou *sjužet*, est une séquence d'événement extraite de l'histoire, d'après un point de vue particulier et dans un ordre particulier.

scénario Un scénario est un ensemble d'événements particuliers, partiellement ordonnés et instanciés dans un environnement virtuel.

scénarisation La scénarisation est un processus comprenant à la fois la spécification du ou des déroulements possibles ou souhaitables de la simulation, et le contrôle (exécution et/ou suivi et correction) du déroulement des événements en temps-réel.

scénarisation extrinsèque La scénarisation est extrinsèque si elle est vue comme une surcouche, une étape supplémentaire de cadrage ou de personnalisation d'un environnement virtuel existant.

scénarisation intrinsèque La scénarisation est intrinsèque lorsqu'elle est partie intégrante du processus de conception de l'environnement virtuel.

situation Une situation correspond à une agrégation d'états.

système de scénarisation Un système de scénarisation est composé d'un ou plusieurs langages de scénarisation, permettant de modéliser le contenu scénaristique et/ou les objectifs scénaristiques, et d'un moteur de scénarisation permettant de gérer de manière dynamique la réalisation du scénario.

trame scénaristique Une trame scénaristique, ou *plot*, est un ensemble de points clés partiellement ordonnés et non instanciés.

Annexe B

Description des modules de la plateforme HUMANS

Sommaire

HUMANS, pour HUman Models based Artificial eNvironments Software platform, est une plateforme de création d'environnements virtuels complexes prenant en compte les facteurs humains. Elle est composée d'un ensemble de moteurs logiciels, qui raisonnent sur un ensemble de modèles communs. Grâce à cette approche modulaire, elle peut fonctionner avec différentes configurations, en ajoutant ou en enlevant des modules selon les besoins.

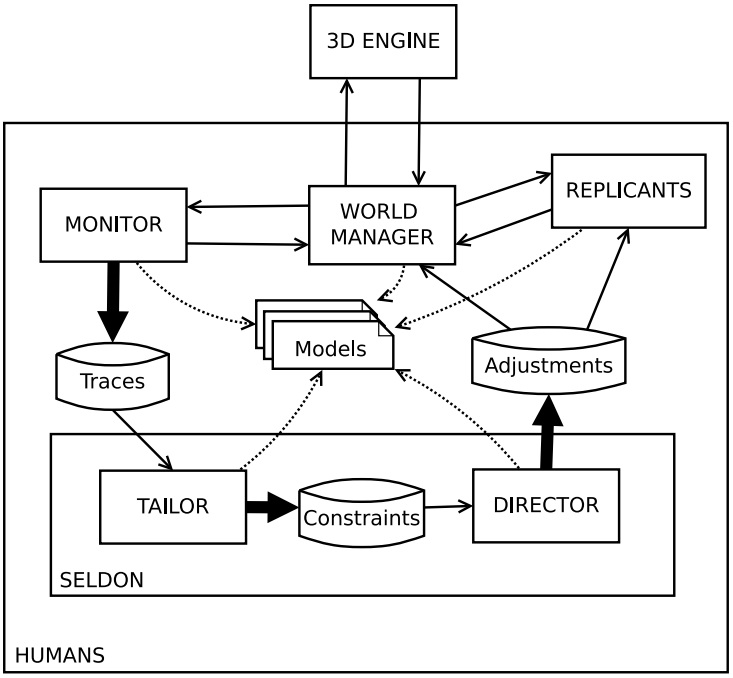


FIGURE B.1 – La plateforme HUMANS.

Cette plateforme se place dans le cadre théorique de la didactique professionnelle et vise à encourager un apprentissage par la pratique, à partir de situations écologiquement valides qui reproduisent des situations de terrain. Afin de rendre compte de comportements observés, les modèles qui servent de base à la simulation sont créés à partir d'analyses terrain (analyses ergonomiques et analyses de risque). De plus, ces modèles ne contiennent pas seulement le déroulement idéal des procédures, mais décrivent également l'activité observée telle qu'elle apparait dans les conditions réelles, tendant vers une description de l'activité finalisée. Ainsi les erreurs et les violations conscientes sont prises en compte dans l'activité, que ce soit du côté de l'utilisateur ou de celui des personnages virtuels qui peuplent l'environnement. En effet, afin de rendre compte de la complexité des situations en termes de facteurs humains, la plateforme HUMANS permet de simuler des personnages virtuels riches, capables de comportements variés et adaptables.
Les scénarios émergent des comportements individuels des personnages virtuels, des systèmes techniques, et du ou des utilisateurs, afin de permettre de rendre compte d'une variété de situations sans pour autant devoir les expliciter *a priori*. Ces scénarios sont représentés *a posteriori* sous la forme de traces de la simulation, qui peuvent être consultées par les utilisateurs ou utilisées comme entrée pour la scénarisation. Les scénarios peuvent en effet être adaptés dynamiquement grâce au modèle SELDON, afin d'offrir des situations d'apprentissage pertinentes en regard du profil et de l'activité de l'apprenant.
La plateforme HUMANS est composée de quatre modules :

- WORLD MANAGER, pour la gestion de l'état du monde ;
- REPLICANTS, pour la génération du comportement des personnages virtuels ;
- MONITOR, pour le suivi de l'apprenant et la génération de traces ;
- SELDON, pour la scénarisation adaptative.

Le module SELDON est lui-même divisé en deux sous-modules :

- TAILOR, pour la génération de situations d'apprentissage ;
- DIRECTOR, pour la génération et la mise en place de scénarios correspondant à ces situations d'apprentissage.

Ces modules utilisent un ensemble de modèles communs, représentés à l'aide de trois langages de description réunis dans le formalisme unifié HUMANS-DL.

B.1 HUMANS-DL - Représentation des connaissances

Les différents modules de la plateforme HUMANS raisonnent sur un ensemble commun de modèles. Trois langages de description ont ainsi été formalisés :

- DOMAIN-DL, qui permet de représenter le fonctionnement du monde ;
- ACTIVITY-DL, qui permet de représenter l'activité des utilisateurs et des personnages virtuels ;
- CAUSALITY-DL, qui permet de représenter les chaînes de causalité.

Les connaissances représentées dans ces modèles sont issues d'analyses terrain, et sont spécifiques au cas d'application addressé. Il est cependant possible de réutiliser tout ou partie de ces modèles pour différents cas d'application : des formations traitant de deux sites industriels similaires pourront par exemple utiliser un même modèle de domaine pour représenter des systèmes techniques identiques, mais différents modèles d'activité pour représenter les différences de pratiques qui se manifestent d'un site à l'autre.
Ces langages ont fait l'objet d'un travail de formalisation réalisé en commun avec l'ensemble de l'équipe HUMANS (D. Lourdeaux, K. Amokrane, K. Carpentier, V. Lanquepin et M. Lhommet). Ils sont présentés en détail dans les parties 5 et 7.1 de ce mémoire.

B.2 WORLD MANAGER - Gestion de l'état du monde

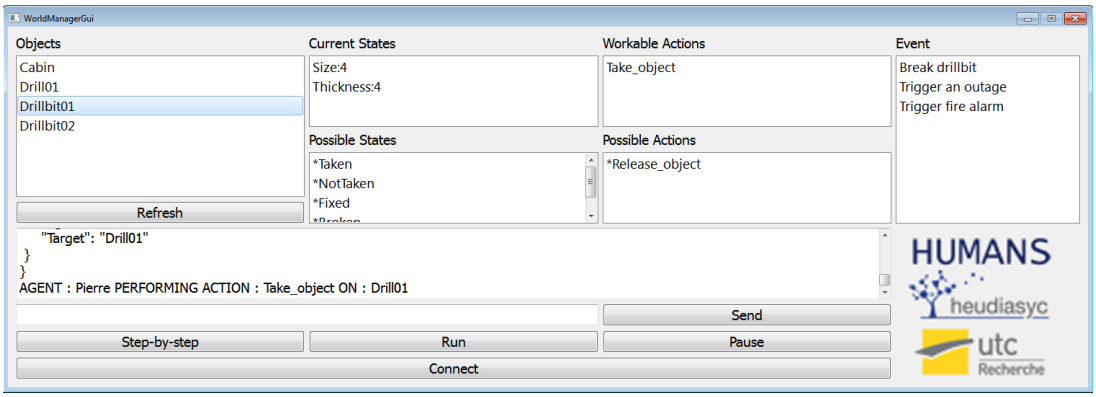


FIGURE B.2 – Interface utilisateur de WORLD MANAGER.

WORLD MANAGER est le module en charge de la gestion des états du monde, qui a été développé par K. Carpentier dans le cadre de sa thèse de doctorat.
L'implémentation de WORLD MANAGER est étroitement liée à la représentation du modèle du monde en DOMAIN-DL. Il est doté une base de connaissances qui contient l'ensemble des instances d'objets et leurs états courants.
WORLD MANAGER traite les demandes d'actions qui proviennent de l'utilisateur, via le moteur 3D, ou des personnages virtuels. Il est doté d'un moteur de règles qui déclenche ou non les comportements associés à ces actions. Ces comportements peuvent être ponctuels ou duratifs (changement continu d'un état sur une certaine durée par exemple).
Il gère également le déclenchement des événements, qu'ils soient déclenchés par un message provenant d'un autre module (événements exogènes) ou par des comportements en interne (événements endogènes).
WORLD MANAGER est implémenté sous la forme d'un programme Java, et utilise la bibliothèque Jena pour interroger l'ontologie et la base de connaissances.

B.3 REPLICANTS - Personnages virtuels

REPLICANTS est le module qui gère le comportement des personnages virtuels qui peuplent l'environnement. Il se place au niveau des processus cognitifs et de la prise de décision, l'objectif étant d'obtenir des personnages à la fois adaptables et explicables afin de rendre compte de comportements de terrain variés.
REPLICANTS faisait l'objet des travaux de doctorat de Margaux Lhommet [Lhommet, 2012]. Les personnages virtuels de REPLICANTS sont modélisés à plusieurs niveaux : celui de la personnalité (avec le modèle OCEAN [Costa and MacCrae, 1992] [Goldberg, 1992]), celui des émotions (avec les modèles Lazarus [Lazarus, 1966] et OCC [Ortony et al., 1988]) et celui des relations sociales [Ochs et al., 2009].
REPLICANTS reprend en partie le modèle proposé par [Edward, 2011] pour la génération de comportements non-idéaux. Les personnages sont capables de faire des déviations et des erreurs par rapport aux procédures qu'ils doivent réaliser. Les erreurs sont générées en fonction de la classification CREAM [Edward, 2011], et les violations conscientes sont modélisées par la notion d'Activités Limites tolérées par l'Usage [Garza and Fadier, 2007] inclue dans le formalisme ACTIVITY-DL, et réalisées ou non en fonction du mode de contrôle du personnage, d'après le modèle COCOM [Hollnagel, 1994].
REPLICANTS est implémenté avec l'architecture cognitive Soar [Laird et al., 1987], avec la sur-couche Soar-NGS pour la génération des buts des personnages virtuels.

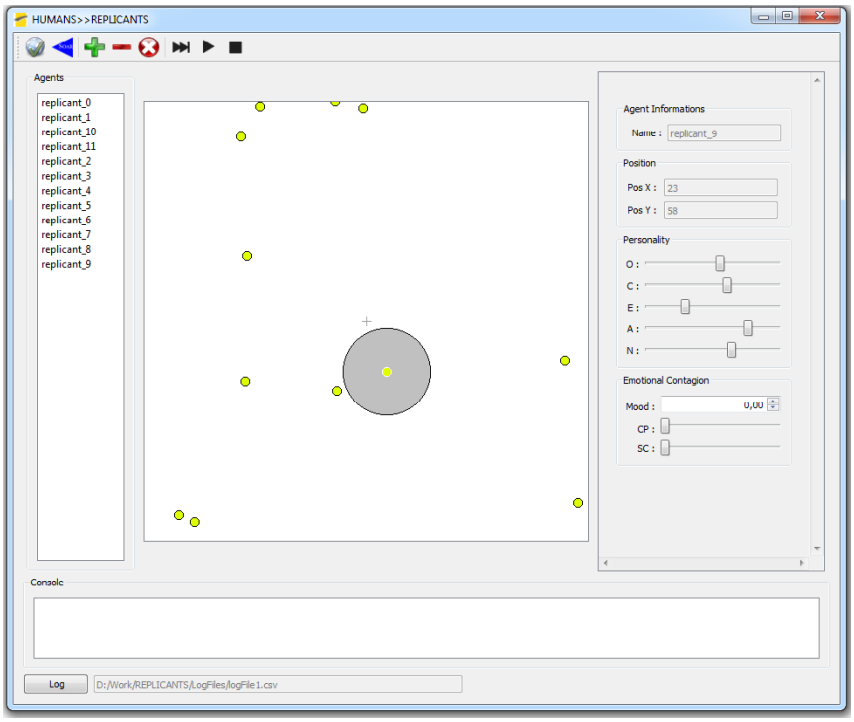


FIGURE B.3 – Interface utilisateur de REPLICANTS.

B.4 MONITOR - Suivi de l'apprenant

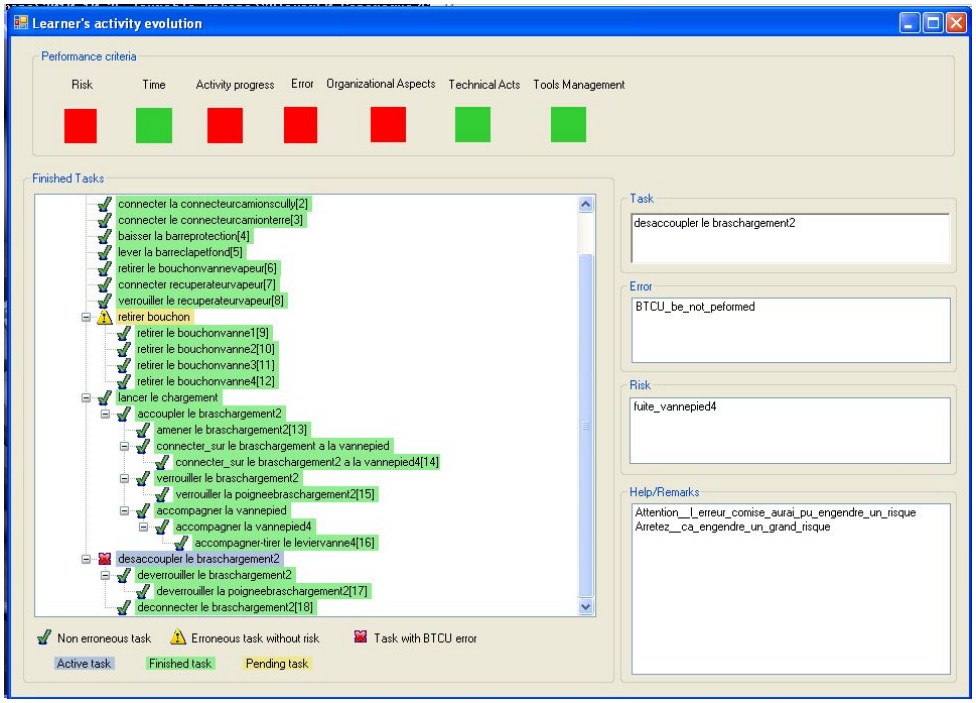


FIGURE B.4 – Interface utilisateur de MONITOR.

MONITOR est le module en charge du suivi et de l'analyse de l'activité de l'apprenant, et de la production de traces de l'environnement. Il est l'évolution du module HERA, issu des travaux de doctorat de Kahina Amokrane [Amokrane, 2010].
MONITOR produit des traces de l'activité de l'apprenant, à la fois à destination du module de scénari-

sation SELDON, de sorte à adapter le scénario de manière pertinente, et à destination du formateur et de l'apprenant, afin de faciliter la compréhension de l'activité de ce dernier et de favoriser pour lui l'adoption d'une posture réflexive. Il utilise pour cela de la reconnaissance de plan à partir du modèle d'activité, et détecte les erreurs. Ces erreurs sont classifiées selon le modèle CREAM [Hollnagel, 1994]. Les violations faites consciemment, identifiées comme telles dans le modèle d'activité [Garza and Fadier, 2007], sont elles aussi détectées. Suite à cette détection, MONITOR peut également fournir des explications sur la gravité et les conséquences possible de l'erreur, en raisonnant sur les informations contenues dans le modèle de la causalité. A partir de ces traces d'activité, MONITOR évalue en temps réel l'apprenant par rapport à un certain nombre de critères de performances. Ces critères peuvent être génériques (ex : erreurs, réactivité), ou dépendre de l'application considérée (ex : hygiène, posture professionnelle). Ces critères visent à assister le formateur dans sa tâche de suivi, lui permettant ainsi de considérer plusieurs apprenants à la fois.
En plus des traces de l'activité de l'apprenant, MONITOR produit également des traces de l'environnement, permettant d'appréhender, en temps-réel ou *a posteriori*, les chaînes de conséquences mettant en jeu les actions des personnages virtuels et le fonctionnement des systèmes techniques.

B.5 TAILOR - Génération de situations d'apprentissage

Le module TAILOR a pour but la sélection des situations d'apprentissage pertinentes en regard du profil et de l'activité de l'apprenant. La notion de situations d'apprentissage est ici prise au sens large : il ne s'agit pas seulement de sélectionner une situation but, mais également spécifier les situations à favoriser ou à éviter au cours du scénario, ainsi qu'un ensemble de propriétés du scénario : complexité, gravité, crédibilité...
Les contraintes générées par TAILOR sont utilisées en entrée par DIRECTOR afin de générer et d'exécuter un scénario répondant à ces contraintes.
TAILOR est l'objet du travail de doctorat de Kevin Carpentier [Carpentier et al., 2013].

Annexe C

Diagrammes UML

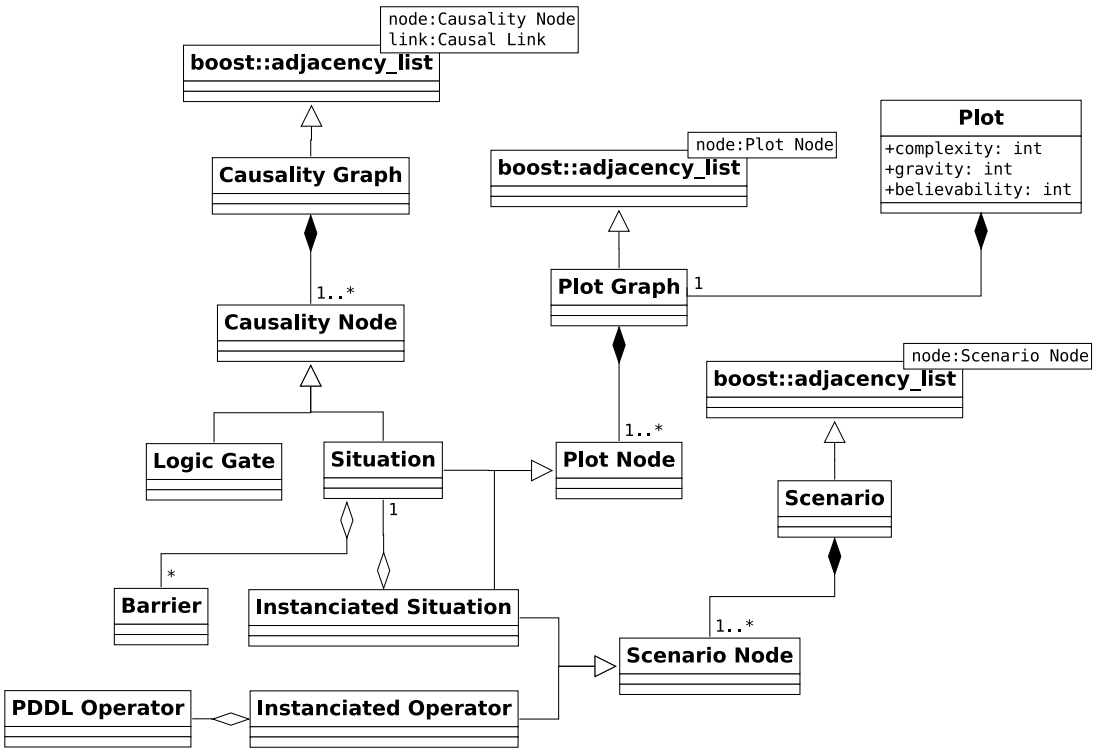


FIGURE C.1 – Diagramme UML des graphes utilisés dans DIRECTOR

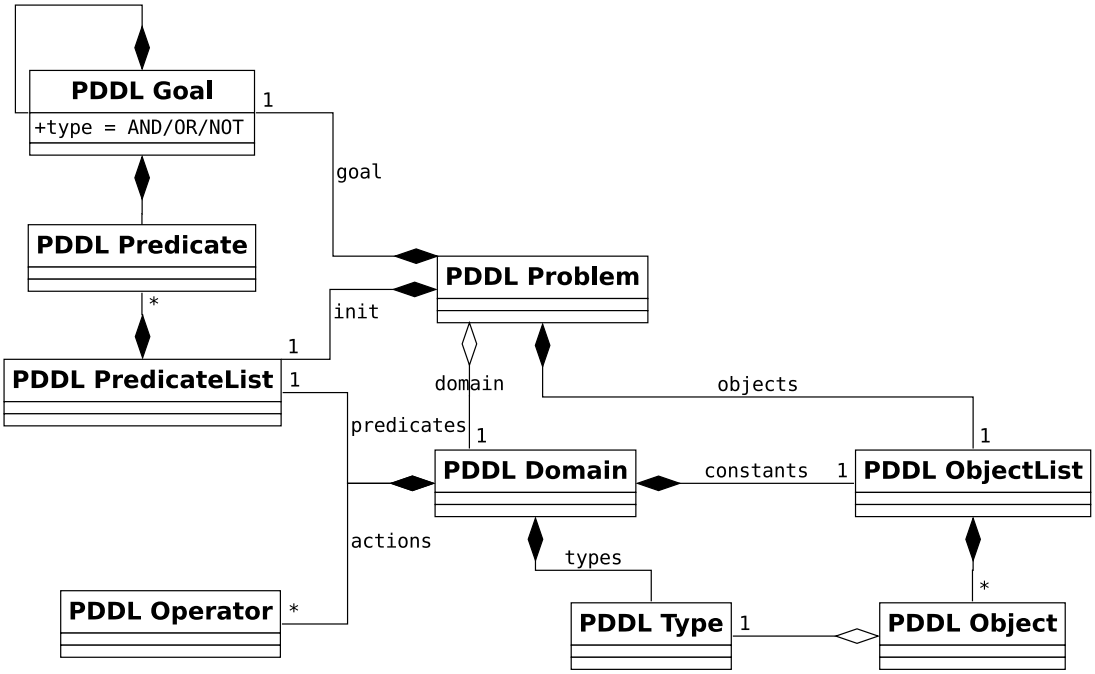


FIGURE C.2 – Diagramme UML des concepts PDDL utilisés dans DIRECTOR

Annexe D

Algorithmes

Algorithm 4 Algorithme d'ordonnancement de sous-tâches séquentielles

```
function OrderSequentialSubtasks(subtasks)
    for all task ∈ subtasks do
        prev_task ← get_previous_task(task)
        while prev_task do
            fav_prec ← prev_task.favourable_precondition
            if fav_prec then
                tree_or ← Tree("OR")
                tree_or.add(Tree("NOT").add(fav_prec))
                tree_or.add("task-state", prev_task, "done")
                task.precondition.add(tree_or)
            prev_task ← get_previous_task(prev_task)
        prev_task ← get_previous_task(task)
        tree_and ← Tree("AND")
        while prev_task do
            tree_and.add("task-state", prev_task, "done")
            prev_task ← get_previous_task(prev_task)
        next_task ← get_next_task(task)
        tree_or ← Tree("OR")
        while next_task do
            fav_prec ← next_task.favourable_precondition
            if fav_prec then
                tree_or.add(Tree("NOT").add(fav_prec))
                tree_or.add("task-state", next_task, "done")
                tree_and.add(tree_or)
            next_task ← get_next_task(next_task)
        if task.favourable_precondition then
            tree_or ← Tree("OR")
            tree_or.add(task.favourable_precondition)
            tree_or.add(tree_and)
            task.precondition.add(tree_or)
        else
            task.precondition.add(tree_and)
```

Algorithm 5 Algorithme de génération d'un plan partiellement ordonné en PDDL à partir du graphe de scénario généré par DIRECTOR

```
function GENERATEPDDL(S)
    current_nodes ← []
    next_nodes ← []
    plan ← []
    add S.goal to current_nodes
    i ← 0
    while current_nodes do
        if current_nodes does not contain only plot points then i ← i − 1
        while current_nodes do
            pop N from current_nodes
            for all N′ in previous-nodes(N) do
                if N′ is a plot point then add N′ to current_nodes
                else
                    if N′ is not in next_nodes then
                        if ∀x in following-nodes(N), x = N or x is a plot point or x is in plan then
                            add N′ to next_nodes
        while previous_nodes do
            pop N from previous_nodes
            add N to next_nodes
            if N is not a plot point then
                plan ← ⟨i, N⟩
        for all ⟨t, S⟩ in plan do
            t ← t − i
    return plan
```

Annexe E

Exemple “Gas Station”

Sommaire

E.1 Codes

Modèle d’activité XML

```
<?xml version="1.0" encoding="UTF-8"?>
<model version="1.0">
  <tasks>
    <task_m>
      <task id="t_refill_gas" name="Refill Gas">
        <context>
          <refs>
            <refconcept concept="Car" ref="?car"/>
            <refconcept concept="Tank" ref="?tank"/>
          </refs>
          <constraints>
            <constraint>
              <triple predicate="has-tank" subject="?car" object="?tank"/>
            </constraint>
          </constraints>
        </context>
        <subtasks>
          <subtask id="t_turn_off_phone"/>
          <subtask id="t_open_tank"/>
          <subtask id="t_pump_gas"/>
          <subtask id="t_close_tank"/>
        </subtasks>
        <constructor type="SEQ-ORD">
          <relation lh="t_turn_off_phone" operator="&lt;" rh="t_open_tank"/>
          <relation lh="t_open_tank" operator="&lt;" rh="t_pump_gas"/>
          <relation lh="t_pump_gas" operator="&lt;" rh="t_close_tank"/>
        </constructor>
        <conditions>
          <contextual id="1">
            <AND>
              <statement type="domain">
                <triple predicate="fill-state-value" subject="?tank" object="empty"/>
```

```
            </statement>
          </AND>
        </contextual>
        <satisfaction id="1">
          <AND>
            <statement type="activity">
              <triple predicate="task-state" subject="t_open_tank" object="done"/>
            </statement>
            <statement type="activity">
              <triple predicate="task-state" subject="t_pump_gas" object="done"/>
            </statement>
            <statement type="activity">
              <triple predicate="task-state" subject="t_close_tank" object="done"/>
            </statement>
          </AND>
        </satisfaction>
      </conditions>
    </task>
  </task_m>
  <task_f>
    <task id="t_turn_off_phone" name="Turn Off Phone">
      <tags>
        <tag type="expert"/>
      </tags>
      <context>
        <refs>
          <refconcept concept="Car" ref="?car"/>
          <refconcept concept="Tank" ref="?tank"/>
          <refconcept concept="Phone" ref="?phone"/>
        </refs>
        <constraints>
          <constraint>
            <triple predicate="has-tank" subject="?car" object="?tank"/>
          </constraint>
        </constraints>
      </context>
      <operation>
        <action id="Turn-Off"/>
        <resources>
          <refconcept ref="?phone" predicate="turned-off-object"/>
        </resources>
      </operation>
      <conditions>
        <contextual id="2">
          <AND>
            <statement type="domain">
              <triple predicate="on-state-value" subject="?phone" object="true"/>
            </statement>
          </AND>
        </contextual>
      </conditions>
    </task>
    <task id="t_open_tank" name="Open Tank">
      <context>
        <refs>
          <refconcept concept="Car" ref="?car"/>
          <refconcept concept="Tank" ref="?tank"/>
        </refs>
        <constraints>
          <constraint>
            <triple predicate="has-tank" subject="?car" object="?tank"/>
          </constraint>
```

```
      </constraints>
    </context>
    <operation>
      <action id="Open"/>
      <resources>
        <refconcept ref="?tank" predicate="opened-object"/>
      </resources>
    </operation>
    <conditions>
      <nomological id="2">
        <AND>
          <statement type="domain">
            <triple predicate="open-state-value" subject="?tank" object="false"/>
          </statement>
        </AND>
      </nomological>
      <satisfaction id="3">
        <AND>
          <statement type="domain">
            <triple predicate="open-state-value" subject="?tank" object="true"/>
          </statement>
        </AND>
      </satisfaction>
    </conditions>
  </task>
  <task id="t_pump_gas" name="Pump Gas">
    <context>
      <refs>
        <refconcept concept="Car" ref="?car"/>
        <refconcept concept="Tank" ref="?tank"/>
      </refs>
      <constraints>
        <constraint>
          <triple predicate="has-tank" subject="?car" object="?tank"/>
        </constraint>
      </constraints>
    </context>
    <operation>
      <action id="Pump"/>
      <resources>
        <refconcept ref="?tank" predicate="has-recipient"/>
      </resources>
    </operation>
    <conditions>
      <nomological id="4">
        <AND>
          <statement type="domain">
            <triple predicate="open-state-value" subject="?tank" object="true"/>
          </statement>
        </AND>
      </nomological>
      <satisfaction id="5">
        <AND>
          <statement type="domain">
            <triple predicate="fill-state-value" subject="?tank" object="full"/>
          </statement>
        </AND>
      </satisfaction>
    </conditions>
  </task>
  <task id="t_close_tank" name="Close Tank">
    <context>
```

```
          <refs>
            <refconcept concept="Car" ref="?car"/>
            <refconcept concept="Tank" ref="?tank"/>
          </refs>
          <constraints>
            <constraint>
              <triple predicate="has-tank" subject="?car" object="?tank"/>
            </constraint>
          </constraints>
        </context>
        <operation>
          <action id="Close"/>
          <resources>
            <refconcept ref="?tank" predicate="closed-object"/>
          </resources>
        </operation>
        <conditions>
          <satisfaction id="6">
            <AND>
              <statement type="domain">
                <triple predicate="open-state-value" subject="?tank" object="false"/>
              </statement>
            </AND>
          </satisfaction>
        </conditions>
      </task>
    </task_f>
  </tasks>
</model>
```

Domaine PDDL généré pour DIRECTOR

```
(define
        (domain gasstation)
        (:requirements :adl)
        (:types use task obj level taskstate behaviour filling bool - object
                joint car agent phone tank - obj)
        (:constants unknown-filling empty full - filling
                unknown-use ok used - use
                pending ongoing done - taskstate
                unknown-level novice expert - level
                open-tank pump-gas close-tank refill-gas turn-off-phone - task
                open pump close leak fire spark turn-off - behaviour
                true false - bool)
        (:predicates (turn-order ?a1 - agent ?a2 - agent)
                (parameters-open ?tank - tank)
                (level-state-value ?a - agent ?l - level)
                (flammable-pool)
                (has-turn ?a - agent)
                (activate-behaviour ?b - behaviour)
                (force-occurrence ?beh - behaviour)
                (fill-state-value ?t - tank ?f - filling)
                (parameters-pump ?tank - tank)
                (force-non-occurrence ?beh - behaviour)
                (parameters-close ?tank - tank)
                (has-joint ?t - tank ?j - joint)
                (parameters-leak ?tank - tank)
                (fire)
                (use-state-value ?j - joint ?u - use)
                (task-state ?t - task ?a - agent ?s - taskstate)
                (on-state-value ?p - phone ?b - bool)
                (open-state-value ?t - tank ?b - bool)
                (parent-parameters-refill-gas ?tank - tank ?a2 - object ?car - car)
                (ignition-source)
                (parameters-turn-off ?phone - phone)
                (parameters-spark ?p - phone)
                (has-acted ?a - agent)
                (has-tank ?car - car ?tank - tank))

(:action action-abstract-parent-refill-gas
        :parameters (?a - agent ?tank - tank ?car - car)
        :precondition (and (not (task-state refill-gas ?a ongoing))
                        (has-turn ?a)
                        (has-tank ?car ?tank)
                        (fill-state-value ?tank empty)
                        (not (has-acted ?a))
                        (forall (?a2 - agent)
                                (not (parent-parameters-refill-gas ?tank ?a2 ?car)))
                        (or (not (task-state open-tank ?a done))
                                (not (task-state pump-gas ?a done))
                                (not (task-state close-tank ?a done))))
        :effect (and (parent-parameters-refill-gas ?tank ?a ?car)
                        (task-state turn-off-phone ?a pending)
                        (task-state refill-gas ?a ongoing)))

(:action action-abstract-end-parent-refill-gas
        :parameters (?tank - tank ?a - agent ?car - car)
        :precondition (and (task-state open-tank ?a done)
                        (task-state pump-gas ?a done)
                        (task-state close-tank ?a done)
                        (parent-parameters-refill-gas ?tank ?a ?car)
```

```
                    (has-turn ?a)
                    (task-state refill-gas ?a ongoing)
                    (forall (?beh - behaviour) (not (activate-behaviour ?beh))))
      :effect (and (task-state refill-gas ?a done)
                    (not (task-state refill-gas ?a ongoing))
                    (not (parent-parameters-refill-gas ?tank ?a ?car))))

(:action action-turn-off-phone
      :parameters (?a - agent ?phone - phone ?tank - tank ?car - car)
      :precondition (and (parent-parameters-refill-gas ?tank ?a ?car)
                    (not (task-state turn-off-phone ?a ongoing))
                    (has-turn ?a)
                    (has-tank ?car ?tank)
                    (on-state-value ?phone true)
                    (not (level-state-value ?a novice))
                    (task-state turn-off-phone ?a pending)
                    (not (has-acted ?a)))
      :effect (and (not (task-state turn-off-phone ?a pending))
                    (task-state turn-off-phone ?a ongoing)
                    (activate-behaviour turn-off)
                    (has-acted ?a)
                    (parameters-turn-off ?phone)))

(:action action-abstract-end-turn-off-phone
      :parameters (?a - agent)
      :precondition (and (task-state turn-off-phone ?a ongoing)
                    (has-turn ?a)
                    (forall (?beh - behaviour) (not (activate-behaviour ?beh))))
      :effect (and (not (task-state turn-off-phone ?a ongoing))
                    (task-state turn-off-phone ?a done)
                    (task-state open-tank ?a pending)))

(:action action-ignore-turn-off-phone
      :parameters (?a - agent ?phone - phone ?tank - tank ?car - car)
      :precondition (and (parent-parameters-refill-gas ?tank ?a ?car)
                    (not (task-state turn-off-phone ?a ongoing))
                    (has-turn ?a)
                    (has-tank ?car ?tank)
                    (task-state turn-off-phone ?a pending)
                    (not (has-acted ?a))
                    (or (not (on-state-value ?phone true))
                           (level-state-value ?a novice)))
      :effect (and (task-state turn-off-phone ?a done)
                    (not (task-state turn-off-phone ?a pending))
                    (task-state open-tank ?a pending)))

(:action action-open-tank
      :parameters (?a - agent ?tank - tank ?car - car)
      :precondition (and (parent-parameters-refill-gas ?tank ?a ?car)
                    (not (open-state-value ?tank true))
                    (not (task-state open-tank ?a ongoing))
                    (has-turn ?a)
                    (has-tank ?car ?tank)
                    (task-state open-tank ?a pending)
                    (not (has-acted ?a))
                    (task-state turn-off-phone ?a done))
      :effect (and (not (task-state open-tank ?a pending))
                    (task-state open-tank ?a ongoing)
                    (activate-behaviour open)
                    (has-acted ?a)
                    (parameters-open ?tank)))
```

```
(:action action-abstract-end-open-tank
       :parameters (?a - agent)
       :precondition (and (task-state open-tank ?a ongoing)
                     (has-turn ?a)
                     (forall (?beh - behaviour) (not (activate-behaviour ?beh))))
       :effect (and (not (task-state open-tank ?a ongoing))
                     (task-state open-tank ?a done)
                     (task-state pump-gas ?a pending)))

(:action action-pump-gas
       :parameters (?a - agent ?tank - tank ?car - car)
       :precondition (and (parent-parameters-refill-gas ?tank ?a ?car)
                     (not (fill-state-value ?tank full))
                     (not (task-state pump-gas ?a ongoing))
                     (has-turn ?a)
                     (has-tank ?car ?tank)
                     (task-state pump-gas ?a pending)
                     (not (has-acted ?a))
                     (task-state open-tank ?a done))
       :effect (and (not (task-state pump-gas ?a pending))
                     (task-state pump-gas ?a ongoing)
                     (activate-behaviour pump)
                     (has-acted ?a)
                     (parameters-pump ?tank)))

(:action action-abstract-end-pump-gas
       :parameters (?a - agent)
       :precondition (and (task-state pump-gas ?a ongoing)
                     (has-turn ?a)
                     (forall (?beh - behaviour) (not (activate-behaviour ?beh))))
       :effect (and (not (task-state pump-gas ?a ongoing))
                     (task-state pump-gas ?a done)
                     (task-state close-tank ?a pending)))

(:action action-close-tank
       :parameters (?a - agent ?tank - tank ?car - car)
       :precondition (and (parent-parameters-refill-gas ?tank ?a ?car)
                     (not (open-state-value ?tank false))
                     (not (task-state close-tank ?a ongoing))
                     (has-turn ?a)
                     (has-tank ?car ?tank)
                     (task-state close-tank ?a pending)
                     (not (has-acted ?a))
                     (task-state pump-gas ?a done))
       :effect (and (not (task-state close-tank ?a pending))
                     (task-state close-tank ?a ongoing)
                     (activate-behaviour close)
                     (has-acted ?a)
                     (parameters-close ?tank)))

(:action action-abstract-end-close-tank
       :parameters (?a - agent)
       :precondition (and (task-state close-tank ?a ongoing)
                     (has-turn ?a)
                     (forall (?beh - behaviour) (not (activate-behaviour ?beh))))
       :effect (and (not (task-state close-tank ?a ongoing))
                     (task-state close-tank ?a done)))

(:action behaviour-spark
       :parameters (?phone - phone)
       :precondition (and (activate-behaviour spark)
                     (force-occurrence spark)
```

```
                    (parameters-spark ?phone)
                    (on-state-value ?phone true))
      :effect  (and (not (activate-behaviour spark))
                    (not (parameters-spark ?phone))
                    (not (force-occurrence spark))
                    (ignition-source)
                    (activate-behaviour fire)))

(:action occurrence-spark-true
      :parameters ()
      :effect (force-occurrence spark))

(:action occurrence-spark-false
      :parameters ()
      :effect (force-non-occurrence spark))

(:action action-abstract-endturn
      :parameters (?current - agent ?next - agent)
      :precondition (and (forall (?beh - behaviour) (not (activate-behaviour ?beh)))
                    (has-turn ?current)
                    (has-acted ?current)
                    (turn-order ?current ?next))
      :effect  (and (not (has-turn ?current))
                    (not (has-acted ?current))
                    (has-turn ?next)))

(:action behaviour-ignore-spark
      :parameters (?phone - phone)
      :precondition (and (activate-behaviour spark)
                    (parameters-spark ?phone)
                    (or (on-state-value ?phone false)
                    (force-non-occurrence spark)))
      :effect  (and (not (activate-behaviour spark))
                    (not (parameters-spark ?phone))
                    (not (force-occurrence spark))
                    (not (force-non-occurrence spark))))

(:action behaviour-fire
      :parameters ()
      :precondition (and (activate-behaviour fire)
                    (ignition-source)
                    (flammable-pool))
      :effect  (and (fire)
                    (not (activate-behaviour fire))))

(:action behaviour-ignore-fire
      :parameters ()
      :precondition (and (activate-behaviour fire)
                    (or (not (ignition-source))
                    (not (flammable-pool))))
      :effect  (and (not (activate-behaviour fire))
                    (not (ignition-source))))

(:action behaviour-open
      :parameters (?t - tank)
      :precondition (and (activate-behaviour open)
                    (parameters-open ?t)
                    (open-state-value ?t false))
      :effect  (and (open-state-value ?t true)
                    (not (open-state-value ?t false))
                    (not (activate-behaviour open))
                    (not (parameters-open ?t))
```

```
                    (activate-behaviour leak)
                    (parameters-leak ?t)))

(:action behaviour-turn-off
       :parameters (?p - phone)
       :precondition (and (activate-behaviour turn-off)
                    (parameters-turn-off ?p)
                    (on-state-value ?p true))
       :effect  (and (not (on-state-value ?p true))
                    (on-state-value ?p false)
                    (not (activate-behaviour turn-off))
                    (not (parameters-turn-off ?p))))

(:action behaviour-leak
       :parameters (?t - tank ?j - joint)
       :precondition (and (activate-behaviour leak)
                    (parameters-leak ?t)
                    (has-joint ?t ?j)
                    (use-state-value ?j used)
                    (open-state-value ?t true)
                    (fill-state-value ?t full))
       :effect  (and (flammable-pool)
                    (not (parameters-leak ?t))
                    (not (activate-behaviour leak))))

(:action behaviour-ignore-leak
       :parameters (?t - tank ?j - joint)
       :precondition (and (activate-behaviour leak)
                    (parameters-leak ?t)
                    (has-joint ?t ?j)
                    (or (use-state-value ?j ok)
                    (open-state-value ?t false)
                    (fill-state-value ?t empty)))
       :effect  (and (not (parameters-leak ?t))
                    (not (activate-behaviour leak))))

(:action behaviour-pump-gas
       :parameters (?t - tank ?old - filling)
       :precondition (and (activate-behaviour pump)
                    (parameters-pump ?t)
                    (open-state-value ?t true)
                    (fill-state-value ?t ?old))
       :effect  (and (not (fill-state-value ?t ?old))
                    (fill-state-value ?t full)
                    (not (activate-behaviour pump))
                    (not (parameters-pump ?t))
                    (activate-behaviour leak)
                    (parameters-leak ?t)))

(:action behaviour-close
       :parameters (?t - tank)
       :precondition (and (activate-behaviour close)
                    (parameters-close ?t)
                    (open-state-value ?t true))
       :effect  (and (not (open-state-value ?t true))
                    (open-state-value ?t false)
                    (not (activate-behaviour close))
                    (not (parameters-close ?t))))

(:action commitment-joint-ok
       :parameters (?j - joint)
       :precondition (use-state-value ?j unknown-use)
```

```
        :effect (and (not (use-state-value ?j unknown-use))
                (use-state-value ?j ok)))

(:action commitment-joint-used
        :parameters (?j - joint)
        :precondition (use-state-value ?j unknown-use)
        :effect (and (not (use-state-value ?j unknown-use))
                (use-state-value ?j used)))

(:action commitment-tank-empty
        :parameters (?t - tank)
        :precondition (fill-state-value ?t unknown-filling)
        :effect (and (not (fill-state-value ?t unknown-filling))
                (fill-state-value ?t empty)))

(:action commitment-tank-full
        :parameters (?t - tank)
        :precondition (fill-state-value ?t unknown-filling)
        :effect (and (not (fill-state-value ?t unknown-filling))
                (fill-state-value ?t full)))

(:action commitment-agent-Novice
        :parameters (?a - agent)
        :precondition (level-state-value ?a unknown-level)
        :effect (and (not (level-state-value ?a unknown-level))
                (level-state-value ?a novice)))

(:action commitment-agent-expert
        :parameters (?a - agent)
        :precondition (level-state-value ?a unknown-level)
        :effect (and (not (level-state-value ?a unknown-level))
                (level-state-value ?a expert)))

(:action happening-phonecall
        :parameters (?phone - phone)
        :effect  (and (activate-behaviour spark)
                      (parameters-spark ?phone)))

)
```

Problème PDDL généré par DIRECTOR - But : fin de la tâche principale

```
(define (problem problem1) (:domain gasstation)
(:objects
        marcel - agent
        joint1 - joint
        tank1 - tank
        car1 - car
        phone1 - phone
)
(:init
        (on-state-value phone1 true)
        (open-state-value tank1 false)
        (has-tank car1 tank1)
        (use-state-value joint1 unknown-use)
        (level-state-value marcel unknown-level)
        (has-joint tank1 joint1)
        (fill-state-value tank1 unknown-filling)
        (turn-order marcel marcel)
        (has-turn marcel)
)
(:goal (or  (and  (task-state Refill-Gas marcel done))
        )
)
)
```

Problème PDDL généré par DIRECTOR - But : feu

```
(define (problem problem2) (:domain gasstation)
(:objects
        marcel - agent
        joint1 - joint
        tank1 - tank
        car1 - car
        phone1 - phone
)
(:init
        (on-state-value phone1 true)
        (open-state-value tank1 false)
        (has-tank car1 tank1)
        (use-state-value joint1 unknown-use)
        (level-state-value marcel unknown-level)
        (has-joint tank1 joint1)
        (fill-state-value tank1 unknown-filling)
        (turn-order marcel marcel)
        (has-turn marcel)
)
(:goal (or  (and  (fire))
        )
)
)
```

E.2 Scénarios générés

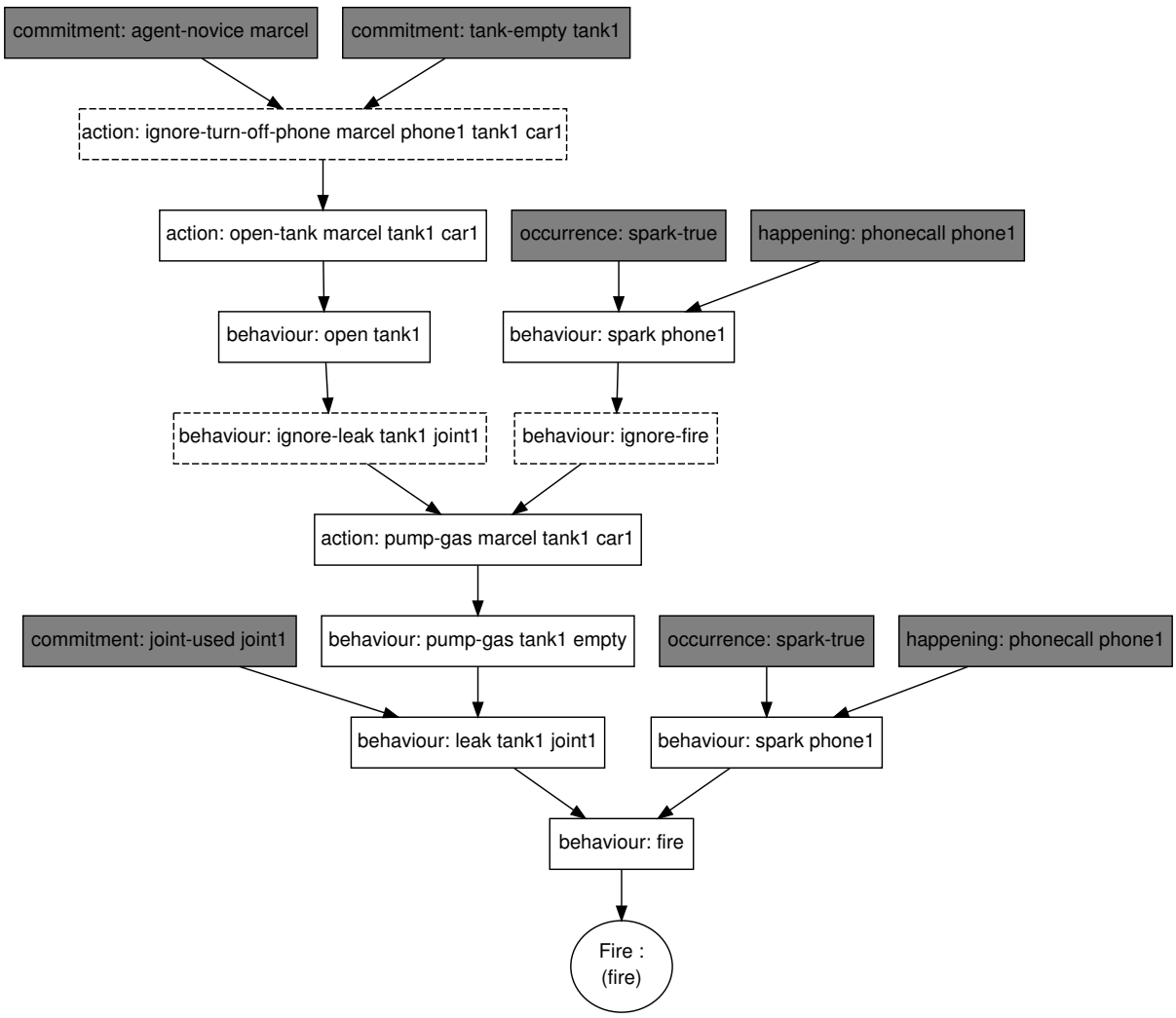


FIGURE E.1 – Scénario généré par FF

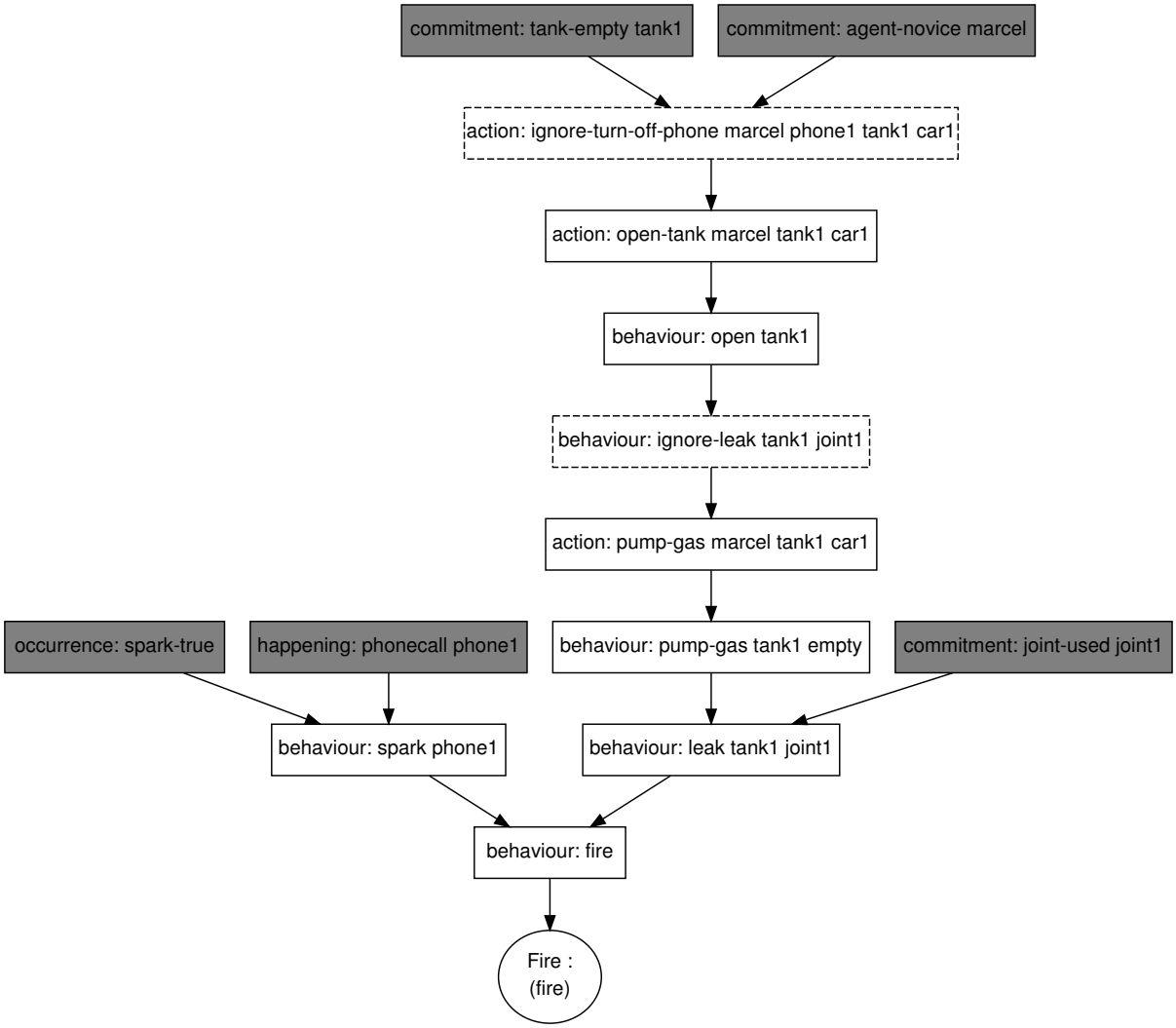


FIGURE E.2 – Scénario généré par Fast Downward (A* blind)

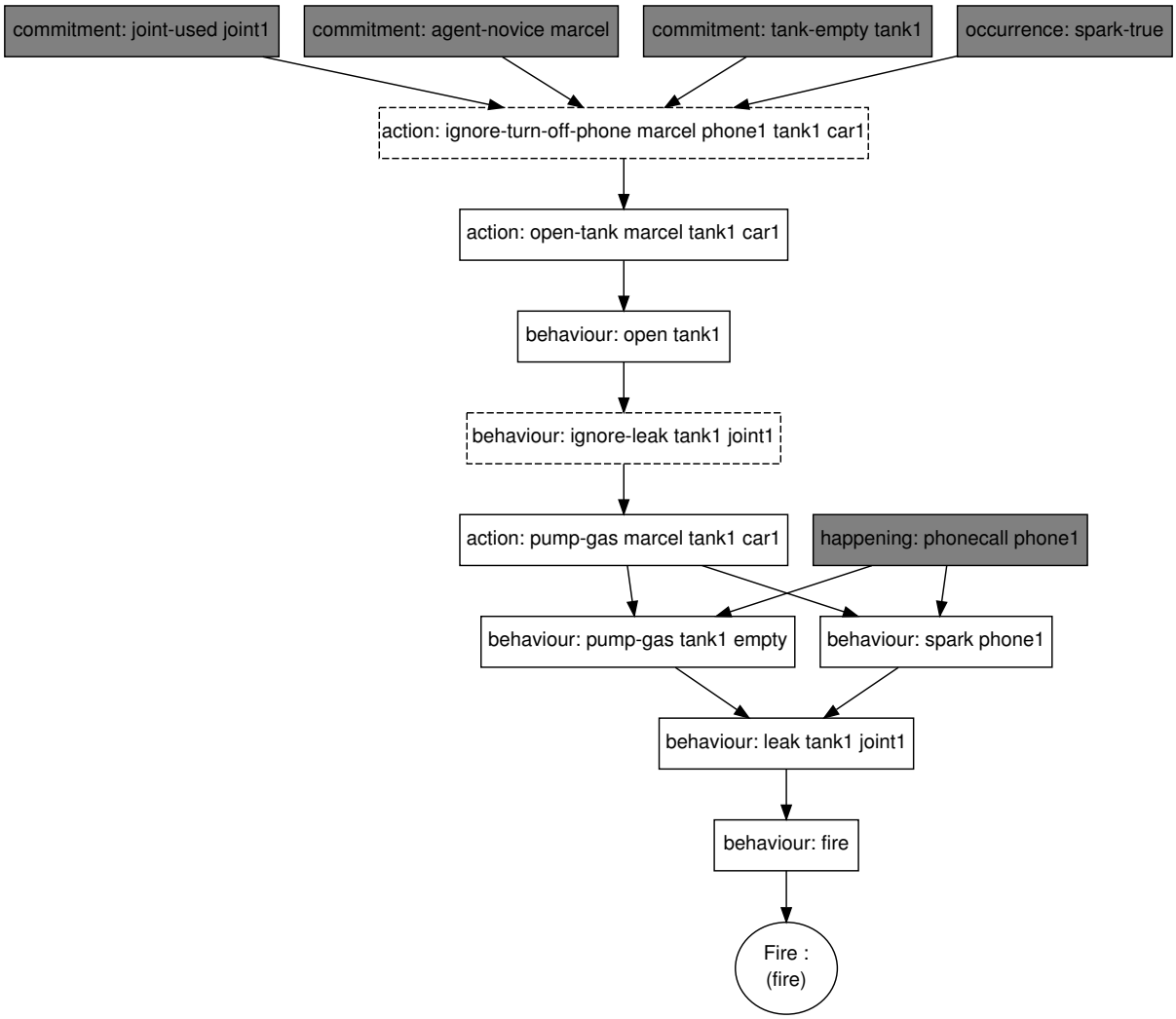


FIGURE E.3 – Scénario généré par VHPOP

Table des figures

Liste des tableaux

Publications

[1] C. Barot, D. Lourdeaux, J.-M. Burkhardt, K. Amokrane, and D. Lenne. V3S : A Virtual Environment for Risk-Management Training Based on Human-Activity Models. *Presence : Teleoperators and Virtual Environments*, 22(1) :1–19, 2013.

[2] C. Barot, D. Lourdeaux, and D. Lenne. Using planning to predict and influence autonomous agents behaviour in a virtual environment for training. In *Proceedings of the 12th IEEE International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC 2013)*, pages 274–281, New York, USA, 2013.

[3] C. Barot, D. Lourdeaux, and D. Lenne. Dynamic scenario adaptation balancing control, coherence and emergence. In *Proceedings of ICAART'13 : International Conference on Agents and Artificial Intelligence*, volume 2, pages 232–237, Barcelona, Spain, 2013.

[4] V. Lanquepin, D. Lourdeaux, C. Barot, K. Carpentier, M. Lhommet, and K. Amokrane. HUMANS : a HUman Models based Artificial eNvironments Software platform. In *Proceedings of the Virtual Reality International Conference (VRIC 2013)*, Laval, France, 2013.

[5] C. Barot, K. Carpentier, M. Collet, A. Cuella-Martin, V. Lanquepin, M. Muller, E. Pasquier, L. Picavet, A. Van Ceulen, and K. Wagrez. The Wonderland Builder : using storytelling to guide dream-like interaction. In *Proceedings of the 2013 IEEE Symposium on 3D User Interfaces*, Orlando, USA, 2013.

[6] C. Barot, J.-M. Burkhardt, D. Lourdeaux, and D. Lenne. V3S, a Virtual Environment for Risk Management Training. In *Proceedings of JVRC11 : Joint Virtual Reality Conference of EGVE - EuroVR*, pages 95–102, Nottingham, United Kingdom, 2011. **Best Paper Award**.

[7] C. Barot, D. Lourdeaux, D. Lenne, and J.-M. Burkhardt. V3s, un environnement virtuel pour la formation à la maîtrise des risques. In *EIAH'2011, Atelier IHM avancées pour l'apprentissage*, Mons, Belgium, 2011.

Bibliographie

[Abedmouleh, 2012] Abedmouleh, A. (2012). Formalisation du langage de conception pédagogique implicite d'un LMS : motivations et processus. In *Quatrièmes Rencontres Jeunes Chercheurs en EIAH*, Amiens.

[Allen, 1983] Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11) :832–843.

[Amokrane, 2010] Amokrane, K. (2010). *Suivi de l'apprenant en environnement virtuel pour la formation à la prévention des risques sur des sites SEVESO*. PhD thesis, Université de Technologie de Compiègne.

[Amokrane et al., 2008] Amokrane, K., Lourdeaux, D., and Burkhardt, J. M. (2008). HERA : learner tracking in a virtual environment. *The International Journal of Virtual Reality*, 7(3) :23–30.

[Anderson et al., 1998] Anderson, C. R., Smith, D. E., and Weld, D. S. (1998). Conditional effects in graphplan. In *Proceedings of AIPS '98*.

[Aylett, 1999] Aylett, R. (1999). Narrative in virtual environments-towards emergent narrative. In *Working notes of the Narrative Intelligence Symposium*.

[Aylett et al., 2006] Aylett, R., Figueiredo, R., Louchart, S., Dias, J., and Paiva, A. (2006). Making it up as you go along-improvising stories for pedagogical purposes. In *Intelligent Virtual Agents*, page 304–315.

[Aylett et al., 2011] Aylett, R., Louchart, S., and Weallans, A. (2011). Research in interactive drama environments, role-play and story-telling. *Interactive Storytelling*, page 1–12.

[Bacchus and Kabanza, 1998] Bacchus, F. and Kabanza, F. (1998). Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2) :5–27.

[Badawi and Donikian, 2004] Badawi, M. and Donikian, S. (2004). Autonomous agents interacting with their virtual environments through synoptic objects. In *Proceedings of CASA 2004*, pages 179–187.

[Balas et al., 2008] Balas, D., Brom, C., Abonyi, A., and Gemrot, J. (2008). Hierarchical petri nets for story plots featuring virtual humans. *Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Barot et al., 2013] Barot, C., Lourdeaux, D., Burkhardt, J.-M., Amokrane, K., and Lenne, D. (2013). V3S : a virtual environment for risk management training based on human-activity models. *PRESENCE : Teleoperators and Virtual Environments*.

[Barthès and Ramos, 2002] Barthès, J.-P. A. and Ramos, M. P. (2002). Agents assistants personnels dans les systèmes multi-agents mixtes - réalisation sur la plate-forme OMAS. *Technique et Science Informatiques*, 21 :473–498.

[Bates, 1992] Bates, J. (1992). The nature of character in interactive worlds and the oz project. Technical report.

[Baudouin, 2007] Baudouin, C. (2007). Un environnement virtuel d'Apprentissage humain pour les travaux pratiques : la paillasse virtuelle. In *Actes du séminaire annuel du Laboratoire d'Informatique des Systèmes Complexes*.

[Bechhofer et al., 2004] Bechhofer, S., Harmelen, F. v., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL web ontology language reference. Technical report, W3C.

[Belmonte, 2008] Belmonte, F. (2008). *Impact des postes centraux de supervision de trafic ferroviaire sur la sécurité*. PhD thesis, PhD Thesis, Université de Technologie de Compiègne.

[Bernuchon et al., 2006] Bernuchon, E., Salvi, O., and Debray, B. (2006). Oméga 7 - méthodes d'analyse des risques générés par une installation industrielle. Technical report, INERIS.

[Blum and Furst, 1997] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, 90(1) :281–300.

[Blumberg and Galyean, 1997] Blumberg, B. and Galyean, T. (1997). Multi-level control for animated autonomous agents : Do the right thing... oh, not that... In Trappl, R. and Petta, P., editors, *Creating Personalities for Synthetic Actors*, number 1195 in Lecture Notes in Computer Science, pages 74–82. Springer Berlin Heidelberg.

[Bonet and Geffner, 2001] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129 :5–33. ACM ID : 380332.

[Bosse and Mogles, 2013] Bosse, T. and Mogles, N. M. (2013). Studying aviation incidents by agent-based simulation and analysis. In *Proceedings of the Fifth International Conference on Agents and Artificial Intelligence*.

[Bourdot et al., 2006] Bourdot, P., Jessel, J.-P., and Thouvenin, I. (2006). Industries manufacturières. In *Le traité de la réalité virtuelle : Volume 4, Les applications de la réalité virtuelle*. Les Presses-Mines Paris.

[Burkhardt et al., 2009] Burkhardt, J., Lourdeaux, D., Couix, S., and Rouillé, M. (2009). La modélisation de l'activité humaine finalisée. In *Le traité de la réalité virtuelle : Volume 5, L'humain virtuel*. Les Presses-Mines Paris.

[Burkhardt, 2010] Burkhardt, J.-M. (2010). *Conception, utilisation et formation : trois perspectives sur l'apprentissage en ergonomie des technologies émergentes*. Habilitation à diriger des recherches.

[Burkhardt et al., 2003] Burkhardt, J. M., Bardy, B., and Lourdeaux, D. (2003). Immersion, réalisme et présence dans la conception et l'évaluation des environnements virtuels. *Psychologie française*, 48 :35–42.

[Burkhardt et al., 2005] Burkhardt, J.-m., Lourdeaux, D., and Lequatre, F. (2005). Environnements virtuels pour l'apprentissage : de l'image d'epinal à la réalité des usages et des configurations socio-techniques. In *Proceedings of the 17th international conference on Francophone sur l'Interaction Homme-Machine*, IHM 2005, page 163–170, New York, NY, USA. ACM.

[Burkhardt et al., 2006] Burkhardt, J.-M., Lourdeaux, D., and Mellet-d'Huart, D. (2006). La réalité virtuelle pour l'apprentissage humain. In *Le traité de la réalité virtuelle : Volume 4, Les applications de la réalité virtuelle*. Les Presses-Mines Paris.

[Buttussi et al., 2013] Buttussi, F., Pellis, T., Cabas Vidani, A., Pausler, D., Carchietti, E., and Chittaro, L. (2013). Evaluation of a 3D serious game for advanced life support retraining. *International Journal of Medical Informatics*, 82(9) :798–809.

[Camus, 2010] Camus, F. (2010). *Conception d'environnements virtuels dédiés à la maîtrise des risques : l'approche MELISSA*. PhD thesis, Université de Technologie de Compiègne.

[Carpentier et al., 2013] Carpentier, K., Lourdeaux, D., and Mouttapa-Thouvenin, I. (2013). Dynamic selection of learning situations in virtual environment. In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART 2013)*, Barcelona, Spain.

[Castagna et al., 2001] Castagna, P., Mebarki, N., and Gauduel, R. (2001). Apport de la simulation comme outil d'aide au pilotage des systemes de production-exemples d'application. In *Proceedings of the 3e Conférence Francophone de MOdélisation et SIMulation «Conception, Analyse et Gestion des Systèmes Industriels» MOSIM'01*.

[Cavazza et al., 2002] Cavazza, M., Charles, F., and Mead, S. J. (2002). Character-based interactive storytelling. *IEEE Intelligent Systems*, 17(4) :17—24.

[Champagnat et al., 2005] Champagnat, R., Prigent, A., and Estraillier, P. (2005). Scenario building based on formal methods and adaptative execution. *ISAGA, Atlanta (USA)*.

[Chang and Soo, 2008] Chang, H.-M. and Soo, V.-W. (2008). Planning to influence other characters in agent-based narratives. In *Integrating Technologies for Interactive Stories Workshop, International Conference on Intelligent Technologies for Interactive Entertainment*, pages 12—17.

[Charles et al., 2003] Charles, F., Lozano, M., Mead, S. J., Bisquerra, A. F., and Cavazza, M. (2003). Planning formalisms and authoring in interactive storytelling. In *Proceedings of TIDSE*, volume 3.

[Charles et al., 2002] Charles, F., Mead, S. J., and Cavazza, M. (2002). Generating dynamic storylines through characters' interactions. *Int. J. Intell. Games & Simulation*, 1(1) :5–11.

[Charlet, 2002] Charlet, J. (2002). *L'ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales*. PhD thesis, Université Pierre et Marie Curie - Paris VI.

[Cheong, 2007] Cheong, Y. (2007). *A Computational Model of Narrative Generation for Suspense*. PhD thesis, North Carolina State University.

[Chevaillier et al., 2009] Chevaillier, P., Querrec, R., and Septseault, C. (2009). VEHA, un métamodèle d'environnement virtuel informé et structuré. *Techniques et sciences informatiques*, 28(6-7) :715–740.

[Ciarlini et al., 2005] Ciarlini, A. E. M., Pozzer, C. T., Furtado, A. L., and Feijó, B. (2005). A logic-based tool for interactive generation and dramatization of stories. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '05, page 133–140, New York, NY, USA. ACM.

[Costa and MacCrae, 1992] Costa, P. T. and MacCrae, R. R. (1992). *Revised NEO Personality Inventory (NEO PI-R) and NEO Five-Factor Inventory (NEO FFI) : Professional Manual*. Psychological Assessment Resources.

[Couix and Burkhardt, 2011] Couix, S. and Burkhardt, J. (2011). Task descriptions using academic oriented modelling languages : a survey of actual practices across the SIGCHI community. *Human-Computer Interaction–INTERACT 2011*, page 555–570.

[Courty et al., 2003] Courty, N., Lamarche, F., Donikian, S., and Marchand, E. (2003). A cinematography system for virtual storytelling. In *Int. Conf. on Virtual Storytelling ICVS'03*, Toulouse, France. Balet, O. and Subsol, G. and Torguet, P.

[Crison et al., 2005] Crison, F., Lecuyer, A., d'Huart, D. M., Burkhardt, J.-M., Michel, G., and Dautin, J.-L. (2005). Virtual technical trainer : Learning how to use milling machines with multi-sensory feedback in virtual reality. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 139–146, 322. IEEE Computer Society.

[Da Dalto, 2004] Da Dalto, L. (2004). CS WAVE : la réalité virtuelle pour la formation au soudage. *Techniques de l'ingénieur. Génie mécanique*, BT1(IN27).

[Damiano and Pizzo, 2008] Damiano, R. and Pizzo, A. (2008). Emotions in drama characters and virtual agents. In *AAAI Spring Symposium : Emotion, Personality, and Social Behavior*, pages 30—37.

[De Vries and Baille, 2006] De Vries, E. and Baille, J. (2006). Apprentissage : référents théoriques pour les EIAH. In *Environnements Informatiques pour l'Apprentissage Humain*.

[Dede et al., 1996] Dede, C., Salzman, M. C., and Loftin, R. B. (1996). ScienceSpace : virtual realities for learning complex and abstract scientific concepts. In *Proceedings of IEEE VRAIS*, page 246–252.

[Delmas, 2009] Delmas, G. (2009). *Pilotage de récits interactifs et mise en oeuvre de formes narratives dans le contexte du jeu vidéo*. PhD thesis, Université de La Rochelle.

[Deust et al., 2008] Deust, C., Bolvin, C., Kordek, M.-A., and Chaumette, S. (2008). Programme EAT-DRA 71-opération c2.1 : Estimation des aspects probabilistes. fiches pratiques : Intégration de la probabilité dans les études de dangers. Technical report, INERIS, Verneuil-en-Halatte (Oise).

[Devillers, 2001] Devillers, F. (2001). *Langage de scénario pour des acteurs semi-autonomes*. PhD thesis, Université de Rennes 1.

[Donikian, 2001] Donikian, S. (2001). HPTS : a behaviour modelling language for autonomous agents. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, page 401–408, New York, NY, USA. ACM.

[Donikian, 2004] Donikian, S. (2004). *Modélisation, contrôle et animation d'agents virtuels autonomes évoluant dans des environnements informés et structurés*. Habilitation à diriger des recherches.

[Edelkamp and Hoffmann, 2004] Edelkamp, S. and Hoffmann, J. (2004). PDDL2.2 : the language for the classical part of the 4th international planning competition. Technical Report Technical Report No. 195, Institut für Informatik.

[Edward, 2011] Edward, L. (2011). *Modélisation décisionnelle de personnages virtuels autonomes évoluant dans un environnement pour la présentation des risques sur les sites SEVESO*. PhD thesis, Université de Technologie de Compiègne.

[Edward et al., 2010] Edward, L., Lourdeaux, D., and Barthès, J.-P. (2010). Knowledge representation : an ontology for managing a virtual environment. In *ICAART 2010*, Valencia, Spain.

[Edward et al., 2008] Edward, L., Lourdeaux, D., Lenne, D., Barthes, J. P., and Burkhardt, J. (2008). Modelling autonomous virtual agent behaviours in a virtual environment for risk. *IJVR : International Journal of Virtual Reality*, 7(3) :13–22.

[El-Kechaï, 2007] El-Kechaï, N. (2007). *Suivi et Assistance des apprenants dans les environnements virtuels de formation*. PhD thesis.

[El-Nasr and Horswill, 2003] El-Nasr, M. S. and Horswill, I. (2003). Expressive lighting for interactive entertainment. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 1, page I–425.

[Fabre et al., 2006] Fabre, D., Couix, S., Burkhardt, J. M., Gounelle, C., and Cabon, P. (2006). Virtual reality to support human factors for safety : where we are and where we (aim to) go. In *ESREL - Safety and Reliability Annual Conference*, Estoril, Portugal.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS : a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, pages 189–208.

[Fitzgerald et al., 2009] Fitzgerald, A., Kahlon, G., and Riedl, M. (2009). A computational model of emotional response to stories. *Interactive Storytelling*, page 312–315.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). PDDL2.1 : an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20 :61–124.

[Fuchs et al., 2006] Fuchs, P., Moreau, G., Arnaldi, B., Berthoz, A., Bourdot, P., Vercher, J. L., Burkhardt, J. M., and Auvray, M. (2006). *Le traité de la réalité virtuelle : Volume 1, L'homme et l'environnement virtuel*. Les Presses-Mines Paris.

[Garcia-Olaya et al., 2011] Garcia-Olaya, A., Jiménez, S., and Lopez, C. L. (2011). Seventh international planning competition : Deterministic part.

[Garza and Fadier, 2007] Garza, C. D. l. and Fadier, E. (2007). Le retour d'expérience en tant que cadre théorique pour l'analyse de l'activité et la conception sûre. *Activités*, 4(1) :188–197.

[Genvo, 2002] Genvo, S. (2002). Transmédialité de la narration vidéoludique : quels outils d'analyse ? *Comparaison*, 2 :103—112.

[Gerbaud, 2008] Gerbaud, S. (2008). *Contribution à la formation en réalité virtuelle : scénarios collaboratifs et intégration d'humains virtuels collaborant avec des utilisateurs réels*. PhD thesis, INSA de Rennes.

[Gerbaud and Arnaldi, 2009] Gerbaud, S. and Arnaldi, B. (2009). Humains virtuels et collaboration dans un environnement virtuel de formation. *Technique et science informatiques (RSTI - TSI)*, 28(6-7) :741–766.

[Gerevini and Long, 2005] Gerevini, A. and Long, D. (2005). Plan constraints and preferences in PDDL3. *The Language of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*.

[Goel, 1992] Goel, A. K. (1992). Representation of design functions in experienced-based design. In *Intelligent Computer Aided Design*, pages 283–303. D. Brown, M. Waldron, and H. Yoshikawa, intelligent computer aided design edition.

[Goldberg, 1992] Goldberg, L. R. (1992). The development of markers for the big-five factor structure. *Psychological Assessment*, 4(1) :26–42.

[Gordon et al., 2004] Gordon, A., Van Lent, M., Van Velsen, M., Carpenter, P., and Jhala, A. (2004). Branching storylines in virtual reality environments for leadership development. In *Proceedings of the national conference on Artificial Intelligence*, page 844–851.

[Gounelle et al., 2007] Gounelle, C., Cabon, P., Burkhardt, J. M., Couix, S., Fabre, D., Anastassova, M., Salem, W., and Colombo., S. (2007). Integrating human factors approaches with virtual reality for safety in the VIRTHUALIS project. In *Proceedings of the Virtual Reality International Conference VRIC*, Laval, France.

[Graesser et al., 1991] Graesser, A. C., Lang, K. L., and Roberts, R. M. (1991). Question answering in the context of stories. *Journal of Experimental Psychology : General*, 120(3) :254–277.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5 :199–220.

[Guéraud and Lejeune, 2011] Guéraud, V. and Lejeune, A. (2011). Une approche auteur pour la scénarisation et le suivi de situations d'apprentissage. In *Atelier Méthodes de Conception EIAH, Environnements Informatiques pour l'Apprentissage Humain*, page 11, Mons.

[Haslum, 2012] Haslum, P. (2012). Narrative planning : Compilations to classical planning. *J. Artif. Intell. Res.(JAIR)*, 44 :383–395.

[Hedge, 2011] Hedge, J. (2011). Gaining insight into decisionmaking through a virtual world environment application. Technical report, RTI International.

[Helmert, 2006] Helmert, M. (2006). The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26 :191–246.

[Helmert et al., 2008] Helmert, M., Do, M., and Refanidis, I. (2008). IPC 2008 : Deterministic competition.

[Hill et al., 2001] Hill, R., Gratch, J., Johnson, W. L., Kyriakakis, C., LaBore, C., Lindheim, R., Marsella, S., Miraglia, D., Moore, B., Morie, J., Rickel, J., Thiébaux, M., Tuch, L., Whitney, R., Douglas, J., and Swartout, W. (2001). Toward the holodeck : integrating graphics, sound, character and story. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, page 409–416, New York, NY, USA. ACM.

[Hoffmann and Nebel, 2001a] Hoffmann, J. and Nebel, B. (2001a). The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 :2001.

[Hoffmann and Nebel, 2001b] Hoffmann, J. and Nebel, B. (2001b). What makes the difference between HSP and FF. In *Proceedings IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*.

[Hollnagel, 1994] Hollnagel, E. (1994). *Human Reliability Analysis : Context and Control*. Academic Press, 1 edition.

[Hullett and Mateas, 2009] Hullett, K. and Mateas, M. (2009). Scenario generation for emergency rescue training games. In *Proceedings of the 4th International Conference on Foundations of Digital Games - FDG '09*, page 99, Orlando, Florida.

[Iuppa et al., 2004] Iuppa, N., Weltman, G., and Gordon, A. (2004). Bringing hollywood storytelling techniques to branching storylines for training applications. In *Proceedings of the Third International Conference for Narrative and Interactive Learning Environments, Edinburgh, Scotland*.

[Jhala and Young, 2010] Jhala, A. and Young, R. M. (2010). Cinematic visual discourse : Representation, generation, and evaluation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2) :69–81.

[Joab et al., 2006] Joab, M., Guéraud, V., and Auzende, O. (2006). Les simulations pour la formation. In *Environnements Informatiques pour l'Apprentissage Humain*.

[Juul, 1998] Juul, J. (1998). A clash between game and narrative. In *Digital Arts and Culture conference*, Bergen, Norway.

[Kallmann and Thalmann, 1999a] Kallmann, M. and Thalmann, D. (1999a). Direct 3d interaction with smart objects. In *Proceedings of the ACM symposium on Virtual reality software and technology*, page 124–130.

[Kallmann and Thalmann, 1999b] Kallmann, M. and Thalmann, D. (1999b). Modeling objects for interaction tasks. In *Computer animation and simulation'98 : proceedings of the Eurographics Workshop in Lisbon, Portugal, August 31-September 1, 1998*, page 73.

[Klinger et al., 2006] Klinger, E., Marié, R.-M., and Viaud-Delmon, I. (2006). Applications de la réalité virtuelle aux troubles cognitifs et comportementaux. In *Le traité de la réalité virtuelle : Volume 4, Les applications de la réalité virtuelle*. Les Presses-Mines Paris.

[Koehler, 2000] Koehler, J. (2000). *IPP - A Planning System for ADL and Resource-Constrained Planning Problems*. Habilitation à diriger des recherches.

[Koper et al., 2003] Koper, R., Olivier, P., and Anderson, T. (2003). IMS learning design information model. Technical report, IMS Global Learning Consortium.

[Kovacs, 2011] Kovacs, D. L. (2011). The complete and commented BNF syntax of PDDL 3.1.

[Laforcade et al., 2007] Laforcade, P., Barré, V., and Zendagui, B. (2007). Scénarisation pédagogique et ingénierie dirigé par les modèles cadre d'étude pour la définition de langages et environnements-outils de scénarisation pédagogique spécifiques à des domaines. In *Actes de la conférence EIAH 2007*.

[Laird et al., 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR : an architecture for general intelligence. *Artificial Intelligence*, 33(1) :1–64.

[Lamarche and Donikian, 2002] Lamarche, F. and Donikian, S. (2002). Automatic orchestration of behaviours through the management of resources and priority levels. In *In Proc. of Autonomous Agents and Multi Agent Systems AAMAS'02*, page 15–19. ACM.

[Lanquepin et al., 2013] Lanquepin, V., Carpentier, K., Lourdeaux, D., Lhommet, M., Barot, C., and Amokrane, K. (2013). HUMANS : a HUman models based artificial eNvironments software platform. In *Proceedings of the Virtual Reality International Conference : Laval Virtual*, VRIC '13, pages 9 :1–9 :8, New York, NY, USA. ACM.

[Lazarus, 1966] Lazarus, R. S. (1966). *Psychological Stress and the Coping Process*. McGraw-Hill.

[Lhommet, 2012] Lhommet, M. (2012). *REPLICANTS : Humains virtuels cognitifs, émotionnels et sociaux. De l'empathie cognitive à l'empathie affective*. PhD thesis, Université de Technologie de Compiègne.

[Lino et al., 2010] Lino, C., Christie, M., Lamarche, F., Schofield, G., and Olivier, P. (2010). A real-time cinematography system for interactive 3D environments. In *Proceedings of the 2010 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, Madrid, Spain.

[Loomis et al., 1999] Loomis, J. M., Blascovich, J. J., and Beall, A. C. (1999). Immersive virtual environment technology as a basic research tool in psychology. *Behavior Research Methods, Instruments, & Computers*, 31(4) :557–564.

[Lourdeaux, 2001] Lourdeaux, D. (2001). *Réalité Virtuelle et Formation : Conception d'Environnements Virtuels Pédagogiques*. PhD thesis, Ecole des Mines de Paris.

[Lourdeaux, 2012] Lourdeaux, D. (2012). *Modélisation de l'activité humaine pour la scénarisation adaptative d'environnements virtuels : Des comportements cognitifs aux comportements erronés en situation dégradée*. Habilitation à diriger des recherches, Université de Technologie de Compiègne.

[Luengo, 2009] Luengo, V. (2009). *Les rétroactions épistémiques dans les Environnements Informatiques pour l'Apprentissage Humain*. Habilitation à diriger des recherches.

[Lugrin and Cavazza, 2006] Lugrin, J. L. and Cavazza, M. (2006). AI-based world behaviour for emergent narratives. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 25.

[Magerko, 2005] Magerko, B. (2005). Story representation and interactive drama. In *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-05)*.

[Magerko, 2007] Magerko, B. (2007). A comparative analysis of story representations for interactive narrative systems. In *AIIDE 07*, pages 91–94.

[Magerko et al., 2005] Magerko, B., Wray, R. E., Holt, L. S., and Stensrud, B. (2005). Improving interactive training through individualized content and increased engagement. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, volume 2005.

[Marion, 2010] Marion, N. (2010). *Modélisation de scénarios pédagogiques pour les environnements de réalité virtuelle d'apprentissage humain*. PhD thesis, Université de Bretagne Occidentale.

[Martel et al., 2007] Martel, C., Lejeune, A., Ferraris, C., and Vignollet, L. (2007). Scénariser les 4 piliers de la pédagogie.

[Mateas, 2002] Mateas, M. (2002). *Interactive drama, art and artificial intelligence*. PhD thesis, Citeseer.

[Mateas and Stern, 2000] Mateas, M. and Stern, A. (2000). Towards integrating plot and character for interactive drama. In *In Working notes of the Social Intelligent Agents : The Human in the Loop Symposium. AAAI Fall Symposium Series. Menlo Park*, page 113–118. AAAI Press.

[Mateas and Stern, 2005] Mateas, M. and Stern, A. (2005). Structuring content in the façade interactive drama architecture. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2005)*, pages 93—98.

[McBride, 2002] McBride, B. (2002). Jena : A semantic web toolkit. *Internet Computing, IEEE*, 6(6) :55–59.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - the planning domain definition language. Technical Report Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

[Meehan, 1977] Meehan, J. R. (1977). TALE-SPIN, an interactive program that writes stories. In *In Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 91–98.

[Miche et al., 2009] Miche, E., Perinet, R., Bolvin, C., Kordek, M.-A., Chaumette, S., and Mace, Y. (2009). Omega 20 - démarche d'évaluation des barrières humaines de sécurité. Technical report, INERIS.

[Mikropoulos and Natsis, 2011] Mikropoulos, T. A. and Natsis, A. (2011). Educational virtual environments : A ten-year review of empirical research (1999–2009). *Computers & Education*.

[Mollet, 2005] Mollet, N. (2005). *De l'Objet-Relation au Construire en Faisant : application à la spécification de scénarios de formation à la maintenance en réalité virtuelle*. PhD thesis.

[Mollet and Arnaldi, 2006] Mollet, N. and Arnaldi, B. (2006). Storytelling in virtual reality for training. In Pan, Z., Aylett, R., Diener, H., Jin, X., Göbel, S., and Li, L., editors, *Technologies for E-Learning and Digital Entertainment*, volume 3942, pages 334–347. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Mott and Lester, 2006] Mott, B. W. and Lester, J. C. (2006). U-DIRECTOR : a decision-theoretic narrative planning architecture for storytelling environments. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06*, page 977, Hakodate, Japan.

[Nau et al., 1999] Nau, D. S., Cao, Y., Lotem, A., and Muñoz-Avila, H. (1999). SHOP : simple hierarchical ordered planner. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 968–975.

[Niehaus and Riedl, 2009] Niehaus, J. and Riedl, M. (2009). Scenario adaptation : An approach to customizing computer-based training games and simulations. In *14th International Conference on Artificial Intelligence in Education (AIED 2009) Workshops Proceedings*, page 89.

[Noy and Mcguinness, 2001] Noy, N. F. and Mcguinness, D. L. (2001). Ontology development 101 : A guide to creating your first ontology. Technical report.

[Ochs et al., 2009] Ochs, M., Sabouret, N., and Corruble, V. (2009). Simulation de la dynamique des émotions et des relations sociales de personnages virtuels= simulation of the socio-emotional dynamics for virtual characters. *Revue d'intelligence artificielle*, 23(2-3) :327–357.

[Ortony et al., 1988] Ortony, A., Collins, A., and Clore, G. L. (1988). *The cognitive structure of emotions / Andrew Ortony, Gerald L. Clore, Allan Collins.* Cambridge University Press, Cambridge [England] ; New York. Includes indexes. Bibliography : p. 193-200.

[Ouraiba, 2012] Ouraiba, E. A. (2012). *Scénarisation pédagogique pour des EIAH ouverts : Une approche dirigée par les modèles et spécifique au domaine métier.* PhD thesis, Université du Maine.

[Paternò, 2003] Paternò, F. (2003). ConcurTaskTrees : an engineered approach to model-based design of interactive systems. In *The Handbook of Task Analysis for Human-Computer Interaction*, pages 483–503.

[Pednault, 1989] Pednault, E. P. D. (1989). ADL : exploring the middle ground between STRIPS and the situation calculus. pages 324–332.

[Pernin and Lejeune, 2004] Pernin, J.-P. and Lejeune, A. (2004). Modèles pour la réutilisation de scénarios d'apprentissage. *TICE Méditerranée, Nice*.

[Pizzi and Cavazza, 2007] Pizzi, D. and Cavazza, M. (2007). Affective storytelling based on characters' feelings. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*.

[Porteous and Cavazza, 2009] Porteous, J. and Cavazza, M. (2009). Controlling narrative generation with planning trajectories : the role of constraints. *Interactive Storytelling*, page 234–245.

[Porteous et al., 2010] Porteous, J., Cavazza, M., and Charles, F. (2010). Applying planning to interactive storytelling : Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2) :10 :1—10 :21.

[Porteous et al., 2013] Porteous, J., Charles, F., and Cavazza, M. (2013). NetworkING : using character relationships for interactive narrative generation. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, page 595–602.

[Porteous et al., 2011] Porteous, J., Teutenberg, J., Pizzi, D., and Cavazza, M. (2011). Visual programming of plan dynamics using constraints and landmarks. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, page 186–193.

[Propp, 1928] Propp, V. (1928). *Morphology of the Folktale*.

[Pynadath and Marsella, 2004] Pynadath, D. V. and Marsella, S. C. (2004). PsychSim : agent-based modeling of social interactions and influence. In *Proceedings of the international conference on cognitive modeling*, page 243–248.

[Querrec and Chevaillier, 2001] Querrec, R. and Chevaillier, P. (2001). Virtual storytelling for training : An application to fire fighting in industrial environment. In *Proceedings of the International Conference on Virtual Storytelling : Using Virtual Reality Technologies for Storytelling*, ICVS '01, page 201–204, London, UK. Springer-Verlag. ACM ID : 713640.

[Rausand and Høyland, 2004] Rausand, M. and Høyland, A. (2004). System analysis - event tree analysis. In *System reliability theory : models, statistical methods, and applications*. Wiley-Interscience, Hoboken, NJ.

[Rempulski, 2013] Rempulski, N. (2013). *Synthèse dynamique de superviseur pour l'exécution adaptative d'applications interactives*. PhD thesis, Université de La Rochelle.

[Rickel and Johnson, 1998] Rickel, J. and Johnson, W. L. (1998). Animated agents for procedural training in virtual reality : Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13 :343–382.

[Riedl et al., 2003] Riedl, M., Saretto, C. J., and Young, R. M. (2003). Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03*, page 741, Melbourne, Australia.

[Riedl and Young, 2006] Riedl, M. and Young, R. (2006). From linear story generation to branching story graphs. *IEEE Computer Graphics and Applications*, 26(3) :23–31.

[Riedl and Young, 2010] Riedl, M. and Young, R. (2010). Narrative planning : Balancing plot and character. *Journal of Artificial Intelligence Research*.

[Riedl, 2004] Riedl, M. O. (2004). *Narrative generation : balancing plot and character*. PhD thesis, North Carolina State University. AAI3154351.

[Riedl et al., 2008a] Riedl, M. O., Rowe, J. P., and Elson, D. K. (2008a). Toward intelligent support of authoring machinima media content : story and visualization. In *Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, page 1–10.

[Riedl et al., 2008b] Riedl, M. O., Stern, A., Dini, D., and Alderman, J. (2008b). Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, 4(2) :23–42.

[Riedl and Young, 2005] Riedl, M. O. and Young, R. M. (2005). An objective character believability evaluation procedure for multi-agent story generation systems. In *Intelligent Virtual Agents*, page 278–291.

[Rizzo et al., 2012] Rizzo, A., Parsons, T. D., and Buckwalter, J. G. (2012). Using virtual reality for clinical assessment and intervention. In *Handbook of Technology in Psychology, Psychiatry, and Neurology : Theory, Research, and Practice*.

[Rizzo et al., 1999] Rizzo, P., Veloso, M., Miceli, M., and Cesta, A. (1999). Goal-based personalities and social behaviors in believable agents. *Applied Artificial Intelligence*, pages 239–271.

[Robotham and Shao, 2012] Robotham, A. and Shao, F. (2012). The value of simulation and immersive virtual reality environments to design decision making in new product development. In *Proceedings of the EWG-DSS Liverpool-2012 Workshop on "Decision Support Systems & Operations Management Trends and Solutions in Industries"*, pages 71—76.

[Rogalski, 2004] Rogalski, J. (2004). La didactique professionnelle : une alternative aux approches de «cognition située» et «cognitiviste» en psychologie des acquisitions. *Activités*, 1(2) :103–120.

[Roupé, 2013] Roupé, M. (2013). *Development and implementations of virtual reality for decision-making in urban planning and building design*. PhD thesis, Chalmers University of Technology.

[Schlemminger, 2013] Schlemminger, G. (2013). Réalité virtuelle et éducation : l'exemple de l'apprentissage des langues. In *Les 7èmes Journées de l'Association Française de Réalité virtuelle, augmentée, mixte et d'interaction 3D*.

[Sebillotte and Scapin, 1994] Sebillotte, S. and Scapin, D. L. (1994). From users' task knowledge to high-level interface specification. *International Journal of Human-Computer Interaction*, 6(1) :1–15.

[Shawver, 1997] Shawver, D. M. (1997). Virtual actors and avatars in a flexible user-determined-scenario environment. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, page 170–171.

[Si, 2010] Si, M. (2010). *Thespian : a decision-theoretic framework for interactive narratives*. PhD thesis.

[Si et al., 2009] Si, M., Marsella, S., and Pynadath, D. (2009). Directorial control in a decision-theoretic framework for interactive narrative. In *Proceedings of the Second International Conference on Interactive Digital Storytelling (ICIDS 09)*, pages 221—233.

[Si et al., 2010] Si, M., Marsella, S., and Pynadath, D. (2010). Importance of well-motivated characters in interactive narratives : an empirical evaluation. In *Proceedings of the Third Joint Conference on Interactive Digital Storytelling (ICIDS 2010)*, pages 16—25.

[Sierhuis et al., 2003] Sierhuis, M., Clancey, W. J., Damer, B., Brodsky, B., and van Hoof, R. (2003). Human activity behavior and gesture generation in virtual worlds for long-duration space missions. In *Proceedings of Intelligent Motion and Interaction within Virtual Environments (IMIVE)*, pages pp. 103–133, London. S. R. Ellis, M. Slater, and T. Alexander.

[Silva et al., 2003] Silva, A., Raimundo, G., and Paiva, A. (2003). Tell me that bit again... bringing interactivity to a virtual storyteller. In Balet, O., Subsol, G., and Torguet, P., editors, *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling*, number 2897 in Lecture Notes in Computer Science, pages 146–154. Springer Berlin Heidelberg.

[Slater and Usoh, 1993] Slater, M. and Usoh, M. (1993). Representation systems, perceptual position, and presence in immersive virtual environments. *Presence : Teleoperators and Virtual Environments*, (2) :221–233.

[Spierling and Iurgel, 2006] Spierling, U. and Iurgel, I. (2006). Pre-conference demo workshop "Little red cap" : The authoring process in interactive storytelling. In Göbel, S., Malkewitz, R., and Iurgel, I., editors, *Technologies for Interactive Digital Storytelling and Entertainment*, number 4326 in Lecture Notes in Computer Science, pages 193–194. Springer Berlin Heidelberg.

[Spierling and Szilas, 2009] Spierling, U. and Szilas, N. (2009). Authoring issues beyond tools. *Interactive Storytelling*, page 50–61.

[Swartjes, 2010] Swartjes, I. (2010). *Whose Story Is It Anyway*. PhD thesis, University of Twente.

[Szilas, 2003] Szilas, N. (2003). IDtension : a narrative engine for interactive drama. In *Proc. TIDSE*, volume 3, page 187–203.

[Szilas, 2007] Szilas, N. (2007). A computational model of an intelligent narrator for interactive narratives. *Applied Artificial Intelligence*, 21 :753–801. ACM ID : 1392617.

[Teutenberg and Porteous, 2013] Teutenberg, J. and Porteous, J. (2013). Efficient intent-based narrative generation using multiple planning agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, AAMAS '13, page 603–610, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Thomas et al., 2011] Thomas, P., Yessad, A., and Labat, J.-M. (2011). Réseaux de petri et ontologies : des outils pour le suivi de l'apprenant dans les jeux sérieux. In *EIAH'2011*, Mons, Belgique.

[Thouvenin, 2009] Thouvenin, I. (2009). *Interaction et connaissance : construction d'une expérience dans le monde virtuel*. Habilitation à diriger des recherches, Université de Technologie de Compiègne.

[Thue et al., 2010] Thue, D., Bulitko, V., Spetch, M., and Romanuik, T. (2010). Player agency and the relevance of decisions. *Interactive Storytelling*, page 210–215.

[Trabasso et al., 1989] Trabasso, T., van den Broek, P., and Suh, S. Y. (1989). Logical necessity and transitivity of causal relations in stories. *Discourse Processes*, 12(1) :1–25.

[Turner, 1993] Turner, S. R. (1993). *Minstrel : a computer model of creativity and storytelling*. PhD thesis, University of California at Los Angeles.

[Uschold et al., 1996] Uschold, M., Gruninger, M., Uschold, M., and Gruninger, M. (1996). Ontologies : Principles, methods and applications. *Knowledge Engineering Review*, 11 :93–136.

[Van den Bergh and Coninx, 2004] Van den Bergh, J. and Coninx, K. (2004). Contextual ConcurTaskTrees : integrating dynamic contexts in task based design. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004*, pages 13–17.

[Van Der Veer et al., 1996] Van Der Veer, G. C., Lenting, B. F., and Bergevoet, B. A. J. (1996). GTA : groupware task analysis - modeling complexity. *Acta Psychologica*, 91 :297—322.

[Van Rijsselbergen et al., 2009] Van Rijsselbergen, D., Van De Keer, B., Verwaest, M., Mannens, E., and Van de Walle, R. (2009). Movie script markup language. In *Proceedings of the 9th ACM symposium on Document engineering*, page 161–170.

[Veloso et al., 1995] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., and Blythe, J. (1995). Integrating planning and learning : The PRODIGY architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(1) :81–120.

[Vesely et al., 1981] Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F. (1981). Fault tree handbook. Technical report, DTIC Document.

[Vidani and Chittaro, 2009] Vidani, A. C. and Chittaro, L. (2009). Using a task modeling formalism in the design of serious games for emergency medical procedures. pages 95–102.

[Wardrip-Fruin et al., 2009] Wardrip-Fruin, N., Mateas, M., Dow, S., and Sali, S. (2009). Agency reconsidered. *Breaking New Ground : Innovation in Games, Play, Practice and Theory. Proceedings of DiGRA 2009*.

[Ware and Young, 2010] Ware, S. G. and Young, R. M. (2010). Rethinking traditional planning assumptions to facilitate narrative generation. In *2010 AAAI Fall Symposium Series*.

[Weber et al., 2011] Weber, B. G., Mateas, M., and Jhala, A. (2011). Building human-level ai for real-time strategy games. In *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*, page 329–336.

[Weyhrauch, 1997] Weyhrauch, P. (1997). *Guiding interactive drama*. PhD thesis, Carnegie Mellon University.

[Windschitl and Winn, 2000] Windschitl, M. and Winn, W. D. (2000). A virtual environment designed to help students understand science. In *Proceedings of the International Conference of the Learning Sciences*, page 290–296.

[Younes and Simmons, 2003] Younes, H. L. and Simmons, R. G. (2003). VHPOP : versatile heuristic partial order planner. *J. Artif. Intell. Res.(JAIR)*, 20 :405–430.

[Young and Moore, 1994] Young, R. M. and Moore, J. D. (1994). DPOCL : a principled approach to discourse planning. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, page 13–20.

[Young et al., 2004] Young, R. M., Riedl, M. O., Branly, M., Jhala, A., Martin, R. J., and Saretto, C. J. (2004). An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1) :51–70.