Gestion adaptative des contenus numériques : proposition d'un framework générique par apprentissage et re-scénarisation dynamique

Damien Mondou

▶ **To cite this version:**

Damien Mondou. Gestion adaptative des contenus numériques : proposition d'un framework générique par apprentissage et re-scénarisation dynamique. Modélisation et simulation. La Rochelle Université, 2019. Français. NNT : . tel-02432536

HAL Id: tel-02432536
https://hal.science/tel-02432536

Submitted on 8 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

La Rochelle Université

LA ROCHELLE UNIVERSITÉ

ÉCOLE DOCTORALE EUCLIDE

LABORATOIRE L3i

THÈSE présentée par :

Damien MONDOU

soutenue le : **mercredi 11 décembre 2019**
pour obtenir le grade de : **Docteur de La Rochelle Université**
Discipline : **Informatique et Applications**

Gestion adaptative des contenus numériques : proposition d'un framework générique par apprentissage et re-scénarisation dynamique

---

Rapporteurs	Philippe Gaussier	Professeur des universités	Université de Cergy-Pontoise
	Eric Gressier	Professeur des universités	Conservatoire National des Arts et Métiers
Examinateurs	Catherine Pelachaud	Directrice de recherche	Institut des Systèmes Intelligents et de Robotique
	Jeanne Lallement	Maître de conférences HDR	La Rochelle Université
	Jean-Loup Guillaume	Professeur des universités	La Rochelle Université
	Cyril Faucher	Maître de conférences	La Rochelle Université
Invitée	Elise Patole-Edoumba	Directrice du Muséum d'Histoire Naturelle	La Rochelle
Direction	Armelle Prigent	Maître de conférences	La Rochelle Université
	Arnaud Revel	Professeur des universités	La Rochelle Université

Thèse réalisée au Laboratoire L3i
Faculté des Sciences et Technologies
Avenue Michel Crépeau
17042 La Rochelle cedex 01

Tel : +33 5 46 45 82 62
Web : `http://l3i.univ-larochelle.fr`

Sous la direction de Armelle Prigent Maitre de conférences
Arnaud Revel Professeur des universités

Financement Allocation de recherche de la région Nouvelle-Aquitaine

Résumé

L'interaction humain-robot (HRI : Human Robot Interaction) est un domaine de recherche particulièrement exploré ces dernières années. L'interactivité est présente dans toutes les formes de communication et d'échange informatisé où la conduite et le déroulement de la situation sont liés à des processus, de rétroaction, de collaboration, de coopération entre les acteurs qui produisent ainsi un contenu, réalisent un objectif ou plus simplement modifient et adaptent leur comportement.

L'interaction est un processus complexe nécessitant des moyens adaptés à la fois pour la phase de conception, mais aussi pour la garantie de la qualité de l'exécution. Ainsi, concevoir une scénarisation de qualité pour une expérience interactive peut soulever un certain nombre de défis. D'une part, il est parfois difficile de produire une arborescence de cas suffisamment riche pour offrir à l'utilisateur une sensation de liberté dans ses choix d'interaction. D'autre part, une trop grande liberté de l'utilisateur dans une arborescence importante réduit la capacité du concepteur à analyser la qualité de chacune des exécutions possibles.

Il est également nécessaire de mettre en place une mécanique adaptative permettant de recalculer le scénario dynamiquement en fonction de l'utilisateur, de ses comportements ou du contexte spécifique de l'interaction. Un tel bouclage de pertinence nécessite de mettre en oeuvre une mécanique d'analyse et d'apprentissage permettant d'intégrer dans le modèle des éléments issus d'un apprentissage réalisé au travers de toutes les interactions passées avec les autres utilisateurs.

Dans cette dynamique, cette thèse a pour objectif de proposer une architecture répondant aux problématiques de conception, de supervision, de pilotage et d'adaptation d'une expérience interactive. Nous proposons donc un framework complet destiné à faciliter la phase de modélisation d'un système interactif et garantissant une souplesse suffisante pour atteindre les objectifs de complexité, d'extensibilité, d'adaptabilité et d'amélioration par apprentissage automatique.

Il s'agit ici de prendre en compte les dimensions de l'interaction (le temps, l'interaction et le contenu) en utilisant un modèle formel capable de construire dynamiquement l'arborescence du scénario à partir de la description des agents, de leurs comportements et des contextes de leur exécution. Dans ces travaux de recherche, nous proposons un modèle formel, CIT, basé sur deux couches de description. Le processus de supervision dynamique consiste à contrôler l'expérience interactive au regard du modèle formel, basé sur des réseaux d'automates temporisés à entrées/sorties. L'un des avantages de ce modèle est qu'il permet de produire, après exécution, une trace complète de l'expérience de chaque utilisateur, modélisée sous la forme d'un automate temporisé. L'analyse de cette information permet d'introduire un processus d'apprentissage automatique pour la modification et la re-scénarisation dynamique de l'expérience interactive au cours de son exécution, par bouclage de pertinence. Nous appliquons pour cela un algorithme d'apprentissage par renforcement : le *Q-Learning*. Celui-ci permet, à travers l'utilisation de la Q-valeur, d'adapter certains paramètres contrôlant l'exécution de l'activité.

Deux plateformes logicielles, CELTIC (Common Editor for Location Time Interaction and Content) et EDAIN (Execution Driver based on Artificial INtelligence), implémentant respectivement le modèle CIT et le moteur de supervision de l'activité ont été développés au cours de cette thèse.

Adaptive content management : proposal of a generic framework through learning and dynamic rewriting

Abstract

Human Robot Interaction (HRI) is a field of research that has been particularly explored in recent years. Interactivity is present in all forms of communication and computerized exchange where the conduct and progress of the situation are linked to processes, feedback, collaboration and cooperation between actors who thus produce content, achieve an objective or simply modify and adapt their behaviour.

Interaction is a complex process requiring appropriate resources both for the design phase and for ensuring the quality of execution. Thus, designing a quality script for an interactive experience can raise some challenges. On the one hand, it is sometimes complex to produce a case tree rich enough to offer the user a feeling of freedom in his interaction choices. On the other hand, too much user freedom in a large tree structure reduces the designer's ability to analyze the quality of each of the possible executions.

It is also necessary to offer the possibility of setting up an adaptive mechanism to recompute the scenario dynamically according to the user, his behaviours or the specific context of the interaction. Such a relevance feedback requires the implementation of an analysis and learning mechanism to integrate into the model elements resulting from learning achieved through all past interactions with other users.

In this dynamic, this thesis aims to propose an architecture that responds to the issues of design, supervision, management and adaptation of an interactive experience. We therefore propose a complete framework to facilitate the modeling phase of an interactive system and guarantee sufficient flexibility to achieve the objectives of complexity, scalability, adaptability and improvement through automatic learning.

The aim here is to take into account the dimensions of interaction (time, interaction and content) by using a formal model capable of dynamically constructing the scenario tree structure from the description of agents, their behaviours and the contexts of their execution. In this research work, we propose a formal model, CIT, based on two description layers. The dynamic supervision process consists in controlling the interactive experience with respect to regard to the formal model, based on networks of timed input/output automata. One of the advantages of this model is that it allows to produce, after execution, a complete trace of each user's experience, modelled in the form of a timed automata. The analysis of this information makes it possible to introduce an automatic learning process for the dynamic modification and re-scripting of the interactive experience during its execution, by relevance feedback. To do this, we apply a reinforcement learning algorithm : the *Q-Learning*. It allows, through the use of the Q-value, to adapt certain parameters controlling the execution of the activity.

Two software platforms, CELTIC (Common Editor for Location Time Interaction and Content) and EDAIN (Execution Driver based on Artificial INtelligence), implementing the CIT model and the activity supervision engine respectively, were developed during this thesis.

Remerciements

Je tiens tout d'abord à remercier mes deux encadrants, Armelle PRIGENT et Arnaud REVEL qui m'ont tant aidé pendant ces années et sans qui rien n'aurait été possible. Toujours à l'écoute et présents, j'ai effectué ma thèse dans de très bonnes conditions grâce à vous. J'ai été très touché par les mots que vous avez pu m'adresser à la fin de ma soutenance. J'ai, moi aussi, pris un grand plaisir à travailler avec vous pendant des années. J'espère pouvoir continuer à le faire dans le futur. Enfin, merci à vous deux également de m'avoir confié vos étudiants dès les premiers jours. J'ai pu découvrir un domaine qui me passionne et pour lequel j'ai un grand plaisir à me lever chaque matin.

Je remercie bien évidemment les membres de mon jury qui ont accepté d'évaluer mes travaux, dans le contexte bien particulier des mouvements sociaux que la France connaît au moment où j'écris ces dernières lignes : Philippe GAUSSIER et Éric GRESSIER, en tant que rapporteurs. Merci pour votre travail de lecture et d'évaluation minutieuse de mes travaux. Merci à Catherine PELACHAUD, Jeanne LALLEMENT et Cyril FAUCHER pour avoir accepté d'être examinateurs lors de ma soutenance. Merci pour vos commentaires et remarques qui m'aideront à mener à bien mes futurs travaux de recherche. Merci également à Élise PATOLE-EDOUMBA pour votre analyse et pour avoir accepté de m'accueillir au sein du Muséum de La Rochelle pour effectuer mes expérimentations, créées conjointement avec vos équipes. Enfin, un grand merci à Jean-Loup GUILLAUME d'avoir accepté d'intégrer au pied levé mon jury en qualité de président, sans qui ma soutenance n'aurait pas été possible. Je remercie également la région Nouvelle Aquitaine pour le financement de mes travaux de recherche.

Cette aventure n'aurait jamais vu le jour sans le stage que j'ai effectué au L3i à la fin de mon master. Merci à Michel et Bruno de m'avoir donné le goût de la recherche et de m'avoir recommandé à Arnaud et Armelle. Cette même aventure n'aurait pas eu la même saveur sans les membres de l'openspace 123. Ne cherchez pas, vous ne trouverez pas mieux. Marcela, Imen, Chloé et Sovan, nous avons beaucoup rigolé, travaillé et échangé sur nos vies de jeunes chercheurs et de jeunes enseignants. Merci à vous pour ces excellents moments passés à vos côtés. Merci à Iuliia, nouvellement nommée MCF à Lyon, qui me fait visiter la planète pour nos vacances. A très vite pour le prochain voyage. Le dernier arrivé, Jordan, qui est, sans nul doute, la personne la plus patiente et gentille que je connaisse. Merci à toi pour ces (très) longues discussions, qui m'ont permis de souffler ces derniers mois, pendant la rédaction et la préparation de la soutenance. Mais tu ne vas pas te débarrasser de moi aussi vite, je te rassure.

Un grand merci à ma famille et plus particulièrement à ma maman et mon frère. Ces derniers mois de thèse nous ont un peu éloigné à cause de la charge de travail, mais dès ce soir, vous allez m'avoir sur le dos. Merci à vous d'être toujours présent quand j'en ai eu besoin.

Je remercie le L3i de m'avoir accueilli et donné les moyens de mener à bien cette thèse et de manière générale l'ensemble des membres du laboratoire et des départements informatique de la Fac de Sciences et de l'IUT, permanents ou non. Merci pour tous ces échanges et pour ces moments passés ensembles, scientifiques ou non. Un merci particulier à Elodie, Christophe, Clément et les plus jeunes Timothée, Valentin, Florian, Julien, Lionel et tous les autres. Je m'excuse si j'en oublie. Merci à tous les doctorants des autres laboratoires, que j'ai pu connaître grâce aux soirées de l'ADocs.

Alors je sais, vous cherchez vos noms depuis le début, c'est le moment. Je termine donc par ma fine équipe, qui m'a tant fait rire. Erlandri, Kathy, Muzzamil, Benjamin et même Dom et ses blagues qui vont (peut-être) me manquer. J'espère ne jamais vous perdre de vue, parce que j'ai découvert de très belles personnes, à l'écoute et toujours présentent dans les bons comme les mauvais moments. J'ai passé d'excellents moments à vos côtés ainsi qu'avec Geneviève et Julie.

C'est ainsi que la phase de rédaction se termine et que je m'apprête à entamer ma nouvelle vie de docteur.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Sommaire

L'Histoire nous montre à quel point les évolutions et les branches de la robotique peuvent être nombreuses. Elles couvrent aujourd'hui de très nombreux domaines d'application à tel point que [Wa17] montre que le marché de la robotique croit encore plus rapidement que prévu (figure 1.1). Ainsi, si les prédictions se confirment, ce marché va exploser dans les années à venir, notamment pour les particuliers.

La tâche première d'un robot consiste à suppléer l'Homme pour des activités ingrates, répétitives ou irréalisables. Le mot robot vient d'ailleurs de *robota* en tchèque, signifiant *travail*, *besogne* ou *corvée*. Si ce rôle est toujours d'actualité pour les robots militaires (ex : Syrano), médicaux (ex : CyberKnife) ou industriels (ex : Isybot de la SNCF), elle ne l'est plus dans de nombreux domaines, comme l'éducation (ex : Lego Mindstorms), les loisirs (ex : Robosapien) ou encore la recherche (ex : Nao). Ainsi, les robots représentent un support considérable pour la réalisation d'expériences ludiques et pédagogiques.

De plus en plus d'activités culturelles sont d'ailleurs proposées au travers des nouveaux supports technologiques pour prendre en compte les attentes de la nouvelle génération : *la génération Z*.

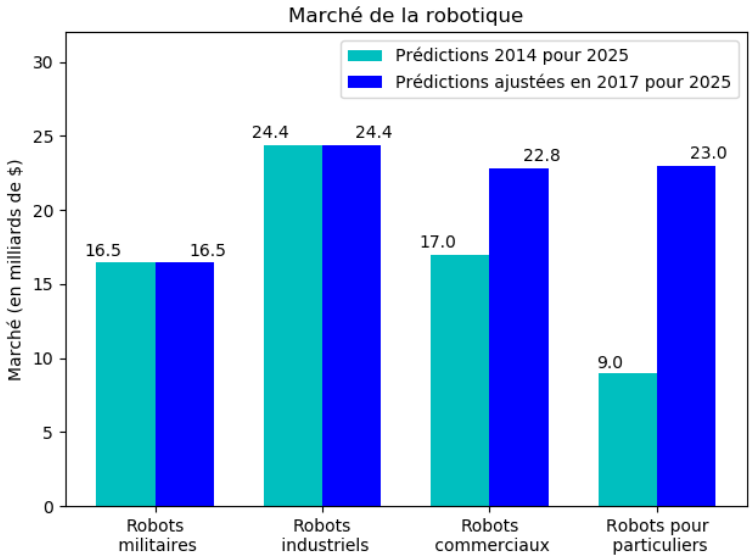


FIGURE 1.1 – Évolution de la prédiction du marché 2025 de la robotique en 2014 et 2017.

Qu'est-ce que la *génération Z* ?

Succédant aux *digital natives*, baptisée ainsi par Prensky [Pre01] (20-35 ans), la génération Z (moins de 20 ans), représente la population dont la caractéristique principale est d'être née dans un monde du "tout numérique", de l'Internet et des réseaux sociaux.

Les pratiques culturelles chez ces jeunes sont connues pour être éloignées de celles que peuvent avoir leurs aînés, du fait de la généralisation des technologies numériques rendant l'accès à l'information et à la culture facilité et ce, à tout instant de la journée, sans contrainte de grille horaire ou de programmation. Les moins de 30 ans sont ainsi 82% à déclarer que la culture représente une place importante dans leur vie [1] (Figure 1.2 à gauche), mais sont paradoxalement 77% à déclarer aller seulement de 1 à 2 fois ou moins par an dans les musées [1] (Figure 1.2 à droite). Ces chiffres rejoignent ceux du ministère de la culture [Ben07] qui, dès 2007, affichaient une baisse des visites de musée et d'exposition pour les jeunes de 15-24 ans, tout en montrant une hausse pour les 25-34 ans.

Si les domaines du tourisme et du commerce ont su s'adapter à cette *génération Z*, aux *digital natives* et aux nouveaux modes de consommation qu'elle impose (nouvelles technologies, adaptation des offres aux visiteurs/clients, promotion et vente en ligne, big data [JCC18]), les lieux de culture admettent un certain retard, qu'ils tentent de corriger, pour attirer à nouveau les jeunes à venir découvrir le patrimoine qu'ils présentent.

1. D'après l'étude opinionway de 2016 pour Agefa PME, réalisée sur 807 jeunes de moins de 30 ans, *Les jeunes et la culture*, `https://www.opinion-way.com/fr/sondage-d-opinion/sondages-publies/les-jeunes-et-la-culture-fevrier-2016-agefa-pme/viewdocument.html`

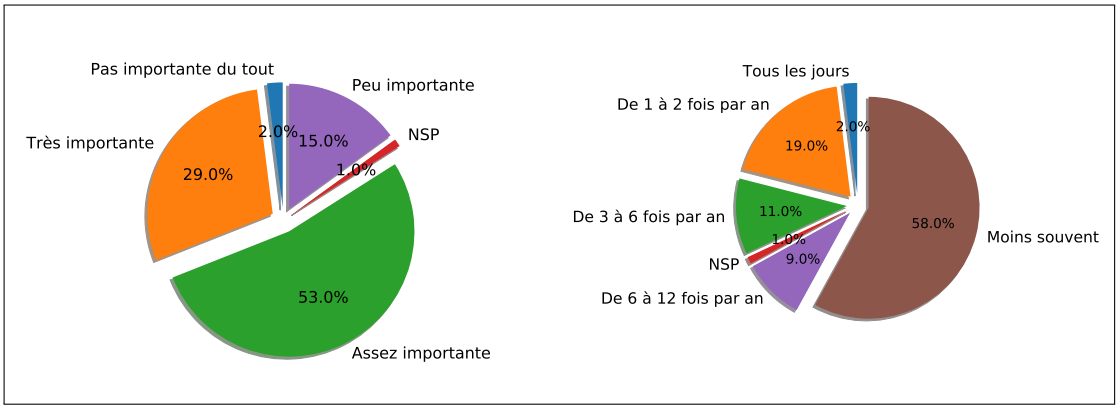


FIGURE 1.2 – A gauche : Place de la culture chez les jeunes. A droite : Fréquence des visites de musée chez les jeunes

Pour s'adapter à ce nouveau public du "tout numérique", les lieux de culture opèrent une modification dans la façon de transmettre le savoir, pour s'éloigner d'une approche unilatérale et verticale. L'objectif est donc de faire découvrir les oeuvres de façon participative et ludique en intégrant de nouveaux supports numériques et interactifs pour augmenter la qualité de la visite et le nombre de visiteurs.

1.1 Nouveaux modèles de médiation des musées

Dans cette optique, les institutions culturelles s'emparent de plus en plus des nouvelles technologies qu'elles proposent au public afin de faire évoluer leur expérience de visite. C'est bien ce que souligne Philippe Chantepie [Oct09] : "*les instances de transmission culturelle [...] sont appelées à revisiter leur modèle de médiation pour l'adapter aux jeunes générations*".

1.1.1 La technologie au service de la médiation muséale

Les audio-guides interactifs et personnalisés se répandent dans les musées et permettent au visiteur de bénéficier davantage d'informations concernant le patrimoine qu'il visite. Les dispositifs de médiation immersifs avec réalité augmentée [MMT+08], 3D et hologrammes font vivre des expériences riches et innovantes aux visiteurs et sont de plus en plus intégrés dans les musées (British Museum, musée d'art moderne de New York et plus modestement le musée Sainte-Croix de Poitiers).

Récemment, les robots ont aussi intégrés des oeuvres artistiques [ABG14], [LRO13] dans le cadre de projets transdisciplinaires entre artistes et scientifiques. De même, les robots investissent aujourd'hui les musées et les lieux de culture pour constituer une alternative aux dispositifs numériques de diffusion d'information ou de création d'engagement auprès des visiteurs. C'est ainsi que les robots ont fait leur entrée dans ce type

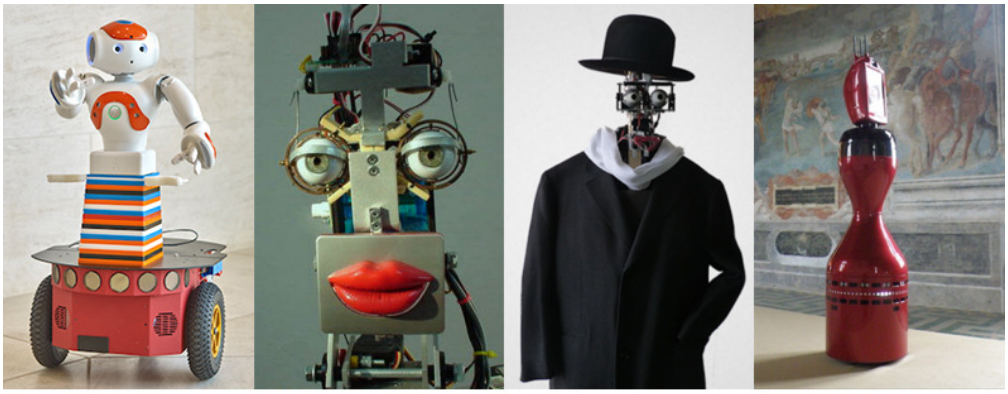


FIGURE 1.3 – Expériences avec des robots dans les musées

de lieux pour la réception (robot Inkha du King's College de Londres par exemple, figure 1.3(b)) et l'orientation des visiteurs. Depuis 2014, au musée national des sciences émergentes et de l'innovation de Tokyo, deux robots sont en charge d'accueillir les visiteurs et peuvent également leur donner les dernières nouvelles. Lors de l'exposition *Eppur Si Muove* du musée d'art moderne du Luxembourg, un robot humanoïde Nao couplé à une base mobile (Figure 1.3(a)) est une oeuvre à lui seul mais guide aussi les visiteurs en leur présentant les différentes oeuvres exposées. En France, dans le cadre d'un projet sur l'esthétisme artificiel, le robot Berenson (Figure 1.3(c)) a été déployé au musée du Quai Branly à Paris. Ce robot se comporte comme un critique d'art et exprime une émotion lorsqu'il voit une oeuvre. Son opinion évolue en fonction de ce qu'il perçoit du ressenti des visiteurs du musée [BGAH14].

Dans une autre dynamique, les robots peuvent répondre à la problématique de certains musées, qui en raison de leur architecture, ne sont pas accessibles aux personnes à mobilité réduite. Afin de rendre cet accès possible à tous, le musée du château d'Oiron a déployé, au premier étage du musée, un robot qui peut être contrôlé à distance par un joystick depuis le rez-de-chaussée, ce qui permet aux personnes handicapées de découvrir les œuvres qui leurs étaient jusqu'ici inaccessibles. Ce robot s'appelle Norio (Figure 1.3 (d)) et est en fonctionnement dans le musée depuis novembre 2014.

1.1.2 Apprendre par le jeu

Les musées cherchent en permanence de nouvelles façons d'augmenter le nombre de visiteurs et de leur permettre de découvrir les artefacts dans l'espace d'exposition tout en acquérant de nouvelles connaissances. Depuis plusieurs années, le *serious game* ou *jeu sérieux* a démontré qu'il peut aider les utilisateurs à améliorer leur compréhension d'un sujet [SF08, Die11]. Ainsi, de nombreux sites culturels ont été équipés de fonctionnalités

amusantes (jeux sur tablettes [CMN13] ou des solutions d'immersion virtuelle [CB10]) qui permettent aux visiteurs d'acquérir de nouvelles connaissances sur les collections ou les oeuvres d'art d'une autre manière.

FIGURE 1.4 – Robot Nao au musée Sainte Croix de Poitiers

A partir de ce constat et dans le cadre d'un projet collaboratif faisant intervenir quatre chercheurs en sciences du numérique (L3i - Laboratoire Informatique, Image et Interaction), et trois en sciences de gestion et économie numérique (CEREGE - CEntre de REcherche en GEstion), nous avons intégré dans les musées des robots humanoïdes Nao avec lesquels était proposé un jeu sérieux. Ce jeu sérieux permettait de faire découvrir aux visiteurs, d'une manière intéressante et ludique, une partie des collections. Nao dispose d'un ensemble de capteurs (par exemple une caméra, un sonar et des capteurs tactiles) qui lui permettent de percevoir l'environnement dans lequel il évolue. Equipé de microphones, de haut-parleurs et des modules logiciels de synthèse et reconnaissance vocale, il est capable d'interagir oralement avec une personne. L'activité, quant à elle, consistait en un jeu de piste dirigé par le robot qui posait des questions aux joueurs. Ces derniers partaient dans le musée à la recherche de la réponse et retournaient vers Nao pour lui répondre.

Ce projet portait un certain nombre d'objectifs :

1. Déterminer l'impact de l'intégration d'un robot dans un lieu culturel ;
2. Identifier les attitudes et comportements des visiteurs dans ces lieux face aux robots ;
3. Proposer un nouvel outil de médiation culturelle ;
4. Examiner les retombés, en terme de flux de visite et de revenus.

FIGURE 1.5 – Affiche de la première expérimentation

1.2 Problématiques scientifiques

Ainsi, un premier prototype de jeu sérieux a été développé en juin 2016. A travers une expérimentation, réalisée au Muséum d'Histoire Naturelle de La Rochelle, ce prototype a permis de valider les capacités d'interactions du robot envers les visiteurs. Développé en collaboration avec le Muséum, le scénario consistait en un ensemble d'interactions visant à mener les visiteurs dans un parcours progressif dans le musée. Ces interactions ont notamment été produites au travers de l'outil propriétaire de la société Aldebaran (Choregraphe). Si les résultats de ces trois jours d'expérimentation se sont

avérés positifs, la méthode de modélisation a montré ses limites. La complexité de modification de l'ordonnancement des actions et de modification des dialogues du robot ont fait apparaître le besoin de disposer d'un outil de scénarisation de plus haut niveau. Ce projet interdisciplinaire nous a donc amené à nous interroger, pour la partie sciences du numérique, sur les approches de conception d'expériences interactives et leur analyse.

A : Problématiques liées à la scénarisation

❶ Les premières réflexions ont porté sur l'**extensibilité** d'un tel jeu. En effet, l'ajout ou la modification de contenus proposés dans le jeu peut s'avérer difficile de même que le déploiement pour d'autres musées. Disposer d'un outil de modélisation simplifiant la représentation et permettant une gestion externalisée des contenus est donc primordial pour faciliter l'extension de jeux.

❷ Le second objectif d'un tel outil est de faciliter la **reproductibilité** d'un scénario pour d'autres lieux. L'objectif ici est de pouvoir disposer d'un ensemble d'éléments réutilisables lors de prochaines modélisations afin de faciliter le travail du concepteur. Il s'agit donc ici d'avoir une vision modulaire de la conception.

❸ Enfin, la **gestion du temps** dans les interactions, notamment homme-robot, est importante pour garantir une haute qualité de l'expérience. Il est en effet nécessaire de pouvoir gérer la dynamique de l'exécution en contraignant le temps alloué aux comportements ou aux attentes du robot. La plateforme de modélisation doit donc permettre de gérer le temps et disposer d'algorithmes de *model-cheking* et de *time-checking*, afin de vérifier la cohérence temporelle du scénario.

Ces trois problématiques nous amènent à nous interroger sur la gestion des contenus (comment les représenter, les stocker, les adapter...), à l'interaction elle même (comment représenter une interaction entre l'homme et la machine) et au temps nécessaire pour la réaliser ou pour réaliser un ensemble d'interactions.

B : Problématique liées à l'adaptation

❹ A l'issu du travail de conception et d'expérimentation du scénario, une phase d'analyse de l'expérience est nécessaire pour déterminer si la session s'est déroulée dans de bonnes conditions. Cette analyse est possible grâce aux traces récoltées pendant l'exécution du scénario. L'utilisation d'algorithmes de parcours de graphe est alors nécessaire pour déterminer des parcours types et des parcours d'erreurs. Sur cette base, le scénario initial doit pouvoir être optimisé pour tenter de diminuer le nombre de parcours menant à un échec. Nous cherchons donc, dans un second temps, à disposer d'une architecture permettant de modifier dynamiquement le scénario par bouclage de pertinence, par l'utilisation de traces d'exécutions de sessions antérieures et par l'analyse du comportement de l'utilisateur courant.

Pour résumer, nous souhaitons donc disposer d'une plateforme de modélisation de scénarios interactifs, disposant d'un système de gestion des contenus et de temps. Le scénario obtenu doit pouvoir être utilisé pour le contrôle de périphériques variés (tel un robot Nao, une tablette, une table tactile ou un jeu en réalité augmentée par exemple) sans en contraindre l'écriture. Un superviseur d'activité doit permettre de contrôler l'exécution du scénario et disposer d'un système d'analyse pour en déduire la prochaine tâche à exécuter. Enfin, la plateforme logicielle, par bouclage de pertinence, doit avoir la possibilité de modifier dynamiquement le scénario pour pallier les problèmes rencontrés et également répondre au comportement réel de l'utilisateur final. En effet, la perception de l'interaction avec un dispositif peut varier d'un utilisateur à un autre (en fonction de l'âge, du genre ou du contexte d'échange). Ces variations n'étant pas prévisibles, il est difficile pour le concepteur du scénario de les intégrer dans la modélisation, seule l'analyse de traces réelles permet d'affiner les éléments de scénarisation.

1.3 Contributions

L'interaction humain-robot (HRI : Human Robot Interaction) est un domaine de recherche particulièrement exploré ces dernières années. Certaines problématiques sont communes avec le domaine de recherche de l'interaction, d'autres sont particulières à la plateforme robotique. L'interactivité est présente dans toutes les formes de communication et d'échange informatisé où la conduite et le déroulement de la situation sont liées à des processus, de rétroaction, de collaboration, de coopération entre les acteurs qui produisent ainsi un contenu, réalisent un objectif, ou plus simplement modifient et adaptent leur comportement.

Un tel processus nécessite des moyens adaptés à la fois pour la phase de conception, mais aussi pour garantir la qualité de l'exécution. Ainsi, concevoir une scénarisation de qualité pour une expérience interactive peut soulever un certain nombre de défis. D'une part, il est parfois complexe de produire une arborescence de cas suffisamment riche pour offrir à l'utilisateur une sensation de liberté dans ses choix d'interaction. De la même manière, une trop grande liberté de l'utilisateur dans une arborescence importante réduit la capacité du concepteur à analyser la qualité de chacune des exécutions possibles. De plus, peu de modèles offrent aujourd'hui la possibilité de mettre en place une mécanique adaptative permettant de recalculer le scénario dynamiquement en fonction de l'utilisateur, de ses comportements ou du contexte spécifique de l'interaction. Enfin, un processus d'amélioration continue nécessite d'intégrer une mécanique d'analyse et d'apprentissage permettant d'intégrer dans le modèle des éléments issus d'un apprentissage réalisé au travers de toutes les interactions passées avec les autres utilisateurs.

[Cav17] identifie de manière tout à fait juste les challenges de la modélisation pour la robotique et souligne la nécessité de disposer de modèles sémantiques formels riches. Ces derniers doivent permettre de disposer d'outils de vérification afin d'éviter toute erreur pouvant s'avérer critique. Ceci est d'autant plus vrai depuis la mise à disposition au grand public de robots, qui ne sont alors plus manipulés par des experts dans des conditions optimales. Les modèles doivent également proposer une abstraction suffisante pour que

les experts disposent d'un environnement de conception efficace.

C'est dans cette dynamique que nous proposons un framework dédié complet destiné à faciliter la phase de modélisation d'un système interactif et garantissant une souplesse suffisante pour atteindre les objectifs de complexité, d'extensibilité, d'adaptabilité et d'amélioration par apprentissage automatique cités précédemment dans les problématiques scientifiques. Il s'agit ici de prendre en compte toutes les dimensions de l'interaction (le temps, l'interaction et le contenu) en utilisant un modèle formel capable de construire dynamiquement et de manière modulaire l'arborescence du scénario à partir de la description des agents, de leurs comportements et des contextes de leur exécution.

Dans ces travaux de recherche, nous proposons le modèle CIT, basé sur deux couches de description. Le processus de supervision dynamique consiste à contrôler l'expérience interactive au regard du modèle formel (à partir de réseaux d'automates temporisés à entrées/sorties). L'un des avantages de ce modèle est qu'il permet de produire, après exécution, une trace complète de l'expérience de chaque utilisateur, modélisée sous la forme d'un automate temporisé. L'analyse de cette information permet d'introduire un processus d'apprentissage automatique pour la modification et la re-scénarisation dynamique de l'expérience interactive au cours de son exécution, par bouclage de pertinence.

Nous proposons donc, dans cette thèse, deux contributions principales :

1. Le modèle formel CIT (Contenus, Interaction et Temps), dédié à la modélisation d'applications interactives. Ce modèle propose une approche de conception divisée en deux couches, facilitant le processus de création et répondant aux besoins de réutilisation, de gestion des contenus, des interactions et du temps ;

2. Deux plateformes logicielles, CELTIC (Common Editor for Location Time Interaction and Content) et EDAIN (Execution Driver based on Artificial INtelligence), implémentant respectivement le modèle CIT et le moteur de supervision de l'activité ;

Enfin, une dernière contribution, en phase exploratoire au moment d'écrire ce manuscrit, propose un système d'analyse et d'apprentissage des comportements de l'utilisateur, sur la base de logs d'expériences passées, qui, par bouclage de pertinence, adapte le scénario au comportement de l'utilisateur courant.

1.4 Organisation de la thèse

Ce manuscrit de thèse est divisé en trois parties. La première partie présente un état de l'art, qui portera, d'une part, sur les méthodes de modélisation formelle de la scénarisation et leur supervision et d'autre part, sur les méthodes permettant d'adapter l'activité au comportement de l'utilisateur final. Nous montrerons ici les enjeux et problématiques de la scénarisation d'expériences interactives, prenant en compte la dimension temporelle.

La deuxième partie de ce manuscrit est dédiée à la présentation de notre modèle CIT (Contenus, Interactions, Temps). Ce modèle, divisé en deux couches, permet une modélisation simplifiée de scénario interactifs, en prenant en compte l'ensemble des dimensions de l'interaction. Grâce à une représentation modulaire des différents éléments

constituant le scénario (agents et leurs comportements, regroupés en contextes d'exécutions), une représentation arborescente est construite, à base de réseaux d'automates temporisés à entrées/sorties. Nous présenterons ensuite, comment, à partir de cette représentation, nous supervisons l'activité, lors de l'exécution du scénario. Nous détaillerons ainsi la sémantique dynamique utilisée ainsi que l'architecture que nous proposons. Cette deuxième partie traite également des problématiques d'adaptation du scénario au cours de l'interaction. Nous présentons ainsi, notre architecture, qui par bouclage de pertinence, modifie dynamiquement le scénario, afin de prendre en compte le comportement réel de l'utilisateur final.

La troisième et dernière partie est consacrée à la présentation des différentes plateformes logicielles implémentées pour valider ces travaux de recherche. CELTIC, est l'éditeur et le générateur de scénarios et implémente l'architecture CIT de notre modèle. La seconde plateforme, EDAIN, est notre outil de supervision d'activité. Elle est en charge de l'exécution du scénario, conçu au travers de CELTIC. Nous présenterons enfin les différentes expérimentations publiques que nous avons réalisées au cours de cette thèse ainsi que les résultats que nous avons obtenus.
Enfin, nous terminerons ce document par présenter un bilan de nos recherches qui ouvrent vers un certain nombre de perspectives.

Première partie

Etat de l'art

Chapitre 2

Approches existantes pour la modélisation et la supervision des systèmes

Sommaire

Comme nous l'avons évoqué dans l'introduction générale de ce document, la conception et la supervision de l'interaction homme-robot sont des processus complexes qui nécessitent la mise en oeuvre de modélisations offrant le potentiel nécessaire pour la prise en compte de cette complexité. De nombreuses recherches sur le sujet ont été menées et au delà de la seule problématique de l'interaction dans le contexte de la robotique, des travaux proposent des modélisations pour l'interaction au sens large dans d'autres contextes, comme par exemple celui des jeux. Les dimensions liées à l'interaction, les contenus, le temps et l'espace lèvent également des interrogations qui peuvent être communes avec le domaine de la robotique. En effet, un concepteur de jeu peut vouloir imposer qu'un niveau du jeu s'effectue dans un laps de temps et un lieu donné, comme il peut être nécessaire de concevoir un scénario robotique contraint sur la vitesse et les déplacements du robot.

Dans le domaine de la conception d'un système robotique complexe et comme le souligne McGanne dans [MRO09], une approche monolithique peut difficilement être utilisée. La nécessité de diviser le modèle se fait alors ressentir. Les approches réductionnistes (également dites par composants) sont alors pertinentes et adaptées à cette problématique. [SGM02] propose ainsi la définition suivante :

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be developed independently and is subject to composition by third parties.

L'avantage majeur des approches par composants est la réutilisation. En effet, l'exécution d'un même composant peut se produire à différentes étapes de l'utilisation d'un logiciel mais dans des contextes particuliers. De même, la division par composants peut avoir des répercutions sur les coûts de production et sur les délais de mise sur le marché.

Notre état de l'art sur la modélisation de jeux et de plateforme robotique s'est donc orienté sur ces approches et principalement sur des méthodes graphiques. En effet, l'un des enjeux de cette thèse est de proposer une modélisation dont la lecture et la compréhension sont facilitées.

Nous nous intéressons donc, dans ce chapitre, à deux types de modélisation, très étudiées dans la littérature [KSB16, LFD+18] :

- les méthodes semi-formelles avec en particulier UML (qui s'est imposé comme un standard dans la modélisation de systèmes complexes). Certaines approches s'appuient donc sur ce formalisme pour la conception d'activités robotiques ;
- les méthodes formelles, qui utilisent un modèle mathématique pour la représentation, l'analyse, la vérification et la preuve de certaines propriétés. Ces aspects

deviennent très importants et indispensables dans la conception d'activités robotiques et dans l'univers du jeu.

Ces modèles permettent donc de représenter l'activité et peuvent ensuite être utilisés pour le pilotage d'expériences sur des processus donnés (par exemple des robots). Nous détaillerons donc, dans un troisième temps, différentes plateformes de supervision de l'activité. Enfin, nous proposerons une analyse des différentes approches présentées, qui nous ont permis d'aboutir à notre proposition.

2.1 Approches semi-formelles - UML

2.1.1 Avant propos

Actuellement en version 2.5 et reconnu comme langage de référence pour la modélisation, l'UML (Unified Modeling Language) [Gro99] a pour objectif de fournir, aux concepteurs de systèmes informatiques, des outils d'analyse, de conception et de mise en oeuvre. UML propose une représentation graphique des systèmes à base de diagrammes (de classes, de séquences, d'activités...), facilitant leur lecture et leur compréhension. Du fait de sa standardisation, UML a été utilisé pour la conception d'activités robotiques, notamment sur deux plateformes, *RobotML : Robotic Modeling Language* et *RobotChart*, mais également pour des robots chirurgicaux.

2.1.2 RobotML

Dans le cadre du projet de recherche français *PROTEUS* (Plate forme pour la Robotique Organisant les Transferts Entre Utilisateurs et Scientifiques), le langage de modélisation RobotML a été conçu pour faciliter la conception, la simulation et le déploiement d'applications robotiques [DKS+12]. Certaines approches étant dépendantes de la plateforme robotique, l'objectif principal de RobotML est de séparer le modèle de l'architecture robotique cible. Ainsi, un même modèle peut être déployé sur plusieurs plateformes cibles.

RobotML s'appuie sur des profils UML et des machines à états ; Il est structuré autour de quatre packages , que nous allons détailler autour de l'exemple proposé dans [DKS+12] (figure 2.1). Dans cet exemple, les auteurs désirent modéliser le comportement d'un robot taxi autonome en milieu urbain.

Le package **architecture** fournit des concepts permettant la modélisation des composants logiciels et matériels. Dans cet exemple, quatre composants ont été créés (en vert sur la figure 2.1) :

- *trajectoryPlanning* : gère la trajectoire à suivre pour arriver à la cible à partir de la position courante ;
- *obstacleDetectionSystem* : observe l'environnement du robot pour détecter les obstacles ;

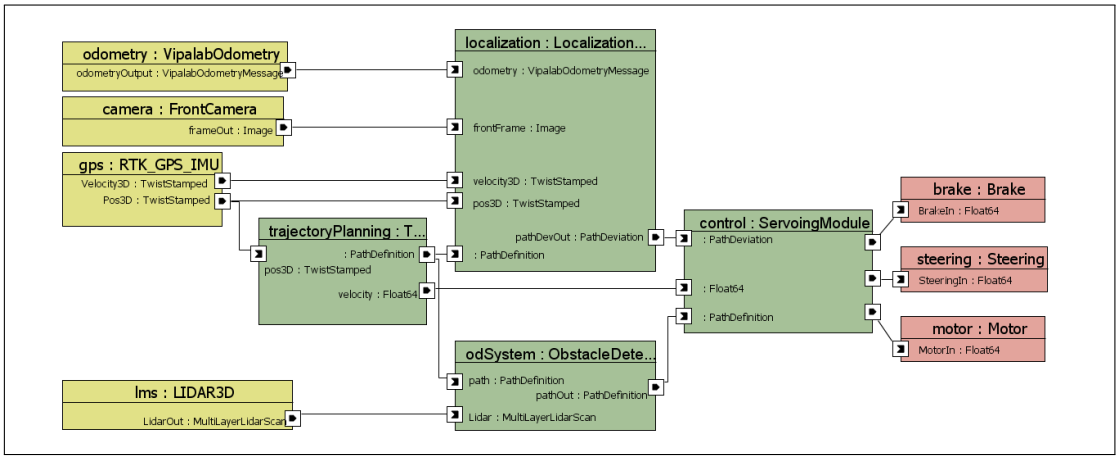


FIGURE 2.1 – Exemple d'architecture sous RobotML

- *localization* : calcule la position courante du robot et détermine le décalage du robot par rapport à la trajectoire désirée ;
- *control* : agrège les données reçues des différents composants pour commander les actionneurs.

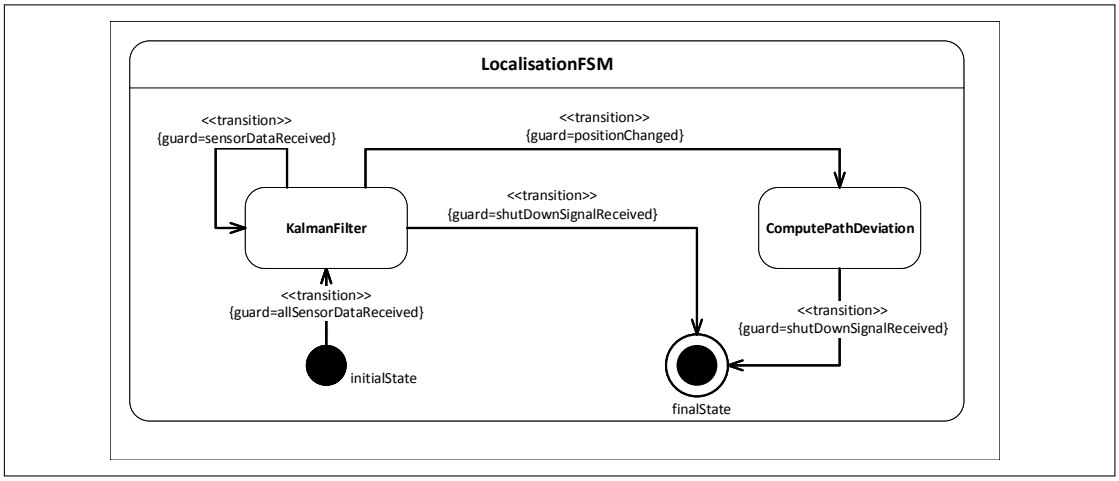


FIGURE 2.2 – Machine à état comportementale du composant *Localization*

Le package **communication** fournit des concepts pour la gestion des flux de données et le contrôle entre composants. Les composants *trajectoryPlannng, obstacleDetectionSystem, localization* et *control* communiquent entre eux des données de façon synchrone ou asynchrone au travers de ports. Le package **comportement** permet la modélisation, par machine à états finis ou par des algorithmes, des comportements de chaque composant. Le premier état, *KalmanFilter* gère les données issues des capteurs et propose une esti-

mation de la position du robot. Si la position du robot a changé, le calcul de la déviation du robot est assuré dans l'état *computePathDeviation*. Enfin, le package **déploiement** associe le système robotique à sa plateforme cible. Dans cet exemple, le middleware OROCOS a été utilisé pour communiquer avec le simulateur de robots Morse. Ce dernier package permet également de générer le code permettant le pilotage d'un robot réel ou d'un simulateur. Pour cela, la conversion du modèle est assuré par l'outil Model to Text (M2T) et permet une génération multi-plateformes.

La phase de création décrite ci-dessus est réalisée grâce au plugin Eclipse dédié, dont un exemple est présenté figure 2.3.

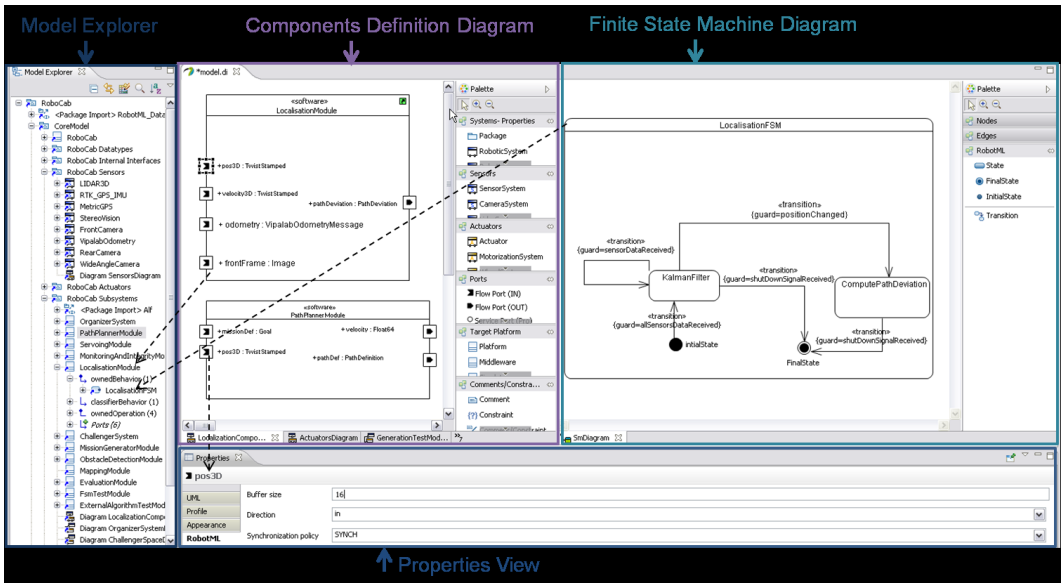


FIGURE 2.3 – Environnement de modélisation de RobotML

Analyse de l'approche : *Bien que RobotML propose une chaîne complète allant de la spécification à la génération de code pour plateforme robotique, il ne propose pas de sémantique formelle permettant la vérification de propriétés. De plus, la gestion du temps et des contenus n'est pas supportée. Cependant, le fait de séparer le modèle de la plateforme cible est l'avantage majeur. Ainsi, il est possible de réutiliser le même modèle pour le déploiement vers d'autres plateformes.*

2.1.3 RobotChart

RoboChart [MRL+16, MRL+19] est un framework de conception destiné lui aussi à la robotique. Il utilise une sémantique semi-formelle via une représentation UML (machines à états). Les primitives temporelles utilisées reposent sur la sémantique proposée dans les automates temporisés [AD92] et sur les CSP temporisés [Sch00] (Communicating

Sequential Processes). La spécification d'un système robotique sous RoboChart repose sur un ensemble de modules connectés à un ou plusieurs contrôleurs, dont la représentation semi-formelle est assurée par une machine à états UML.

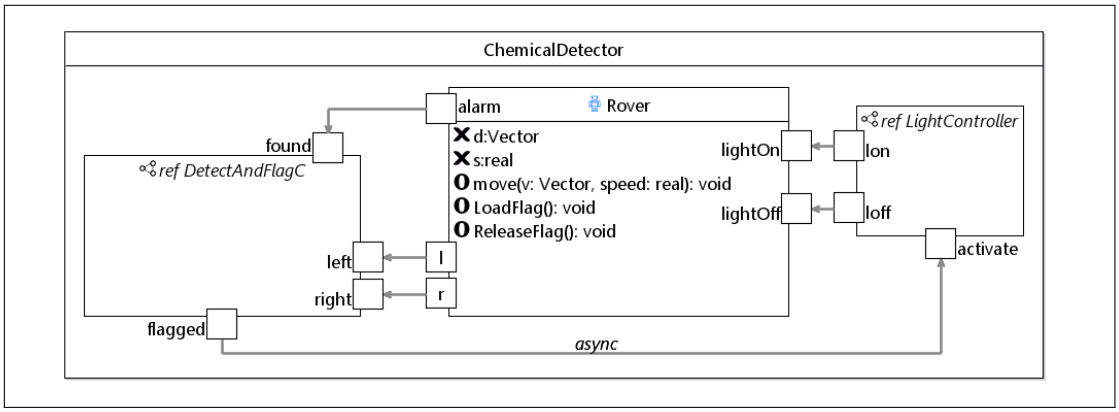


FIGURE 2.4 – Exemple du module *ChemicalDetector* modélisé sous RobotChart issu de [MRL+16]

Les auteurs proposent un exemple de modélisation destinée à la téléopération d'un robot détecteur de gaz. L'objectif ici est de faire naviguer le robot à la recherche de gaz et d'émettre une alerte lors d'un relevé positif (clignottement d'une lumière + drapeau marquant la zone). Dans l'exemple de la figure 2.4, le module *ChemicalDetector* est composé de la plateforme robotique *Rover* et de deux contrôleurs : *DetectAndFlagC* et *LightController* (représenté à la figure 2.5). Le contrôleur *LightC* est dans l'état *Rest* jusqu'à réception de l'événement *activate*.

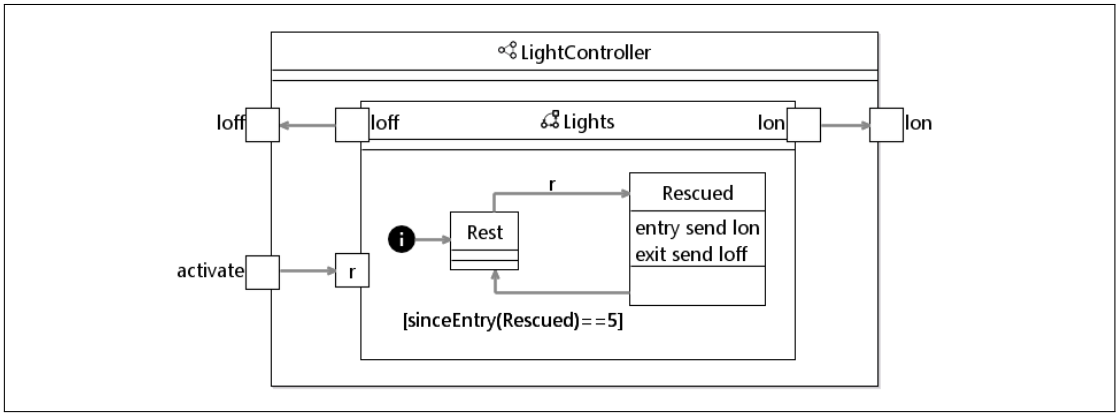


FIGURE 2.5 – Exemple du contrôleur *light* sous RobotChart issu de [MRL+16]

La communication entre la plateforme robotique et les contrôleurs est réalisée par un mécanisme d'événements synchrones (*alarm, l, r, lightOn* et *lightOff* dans l'exemple de la figure 2.4). La communication entre contrôleurs peut être, en revanche, asynchrone.

RoboChart dispose d'un outil de modélisation : *RoboTool*. *RoboTool* est un ensemble de plugins pour l'éditeur Eclipse. Cet outil permet de concevoir et de visualiser le modèle robotique, sous sa forme textuelle et graphique. En effet, un modèle RobotChart repose sur un langage de description textuel.

RoboChart ne permet pas de contrôler un robot directement. RobotTool permet de traduire le modèle en CSP. La vérification du modèle est alors possible en utilisant l'outil FDR (Failures Divergences Refinement) [GRABR14].

Analyse de l'approche : *RobotChart utilise les machines à états pour la modélisation de platformes robotiques. L'utilisation de l'UML permet une rapide compréhension du modèle. RobotChart dispose d'une chaîne complète allant de la conception à la vérification du modèle. La conception est facilitée par l'utilisation d'un outil dédié : RoboTool. Si la notion de temps est supportée, une gestion efficace des contenus n'est ici pas proposée.*

2.1.4 Autre approche

S'il existe un domaine où l'erreur n'est pas permise, c'est bien celui de l'univers médical et plus particulièrement des robots chirurgicaux. Si la plupart d'entre eux sont toujours contrôlés en téléopération par un chirurgien, la possibilité de réaliser des opérations simples par des robots autonomes fait aujourd'hui l'objet de recherches.

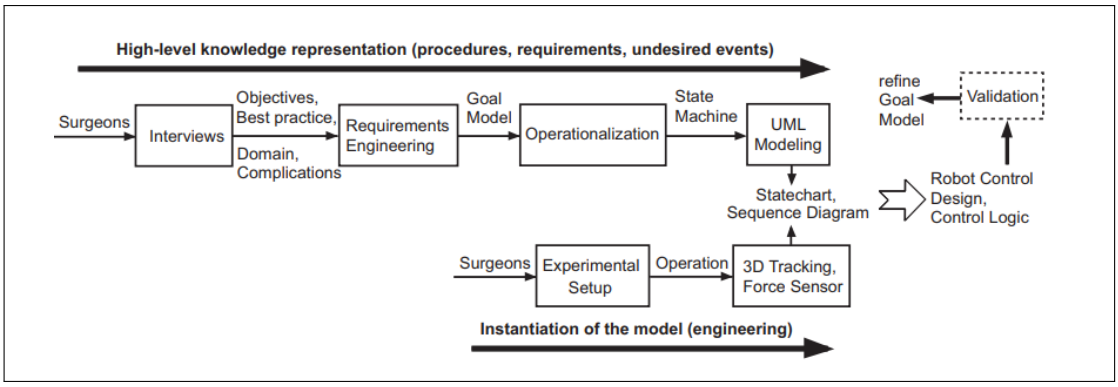


FIGURE 2.6 – Étapes de conception de la méthode proposée dans [BBD+12]

Ainsi, [BBD+12] propose, à partir des exigences médicales, une solution permettant de concevoir des tâches robotiques destinées à ce contexte (figure 2.6). Comme dans les approches présentées précédemment, les actions du robot sont modélisées par des machines à états UML. La vérification du modèle est réalisée par sa conversion en propriétés formelles en utilisant le langage Alloy et la logique temporelle linéaire. Les machines à états sont ensuite converties en code exécutable grâce au framework Orocos [KS10].

Analyse de l'approche : *Cette approche propose une modélisation par machines à état UML. Si celle-ci permet de concevoir des tâches robotiques destinées au milieu chirurgical, jusqu'à leurs exécutions, elle ne prend pas en compte la notion de temps.*

2.2 Approches formelles

L'utilisation de méthodes formelles se justifie par le fait qu'elles utilisent, contrairement aux méthodes semi-formelles, un modèle mathématique pour l'analyse, la vérification et la preuve de certaines propriétés. Cette analyse est nécessaire afin de garantir une haute qualité de l'activité, au travers de propriétés fondamentales. Nous retrouvons également ces problématiques dans la conception de jeux.

L'ensemble des approches proposées mettent en évidence la nécessité de disposer d'outils performants, contrôlant notamment certaines propriétés de sûreté. En effet, l'utilisation de robots en interaction avec des humains peut engendrer des événements non désirables et potentiellement dangereux. De ce fait, nous nous attardons sur deux catégories particulières :

- les approches à base de réseaux de Petri, orientées ressources ;
- les approches à base de réseaux d'automates, orientées actions.

2.2.1 Approches basées sur les réseaux de Petri

2.2.1.1 Avant propos

Initialement créés pour la description de processus chimiques par Carl Adam Petri [Pet66], les réseaux de Petri ont été largement utilisés dans d'autres domaines. Nous les retrouvons notamment pour la conception de réseaux de communication [JRR15], les bases de données relationnelles [BHM15, MR17] ou encore la modélisation formelle de jeux [Ara09, DOJSP11, BJ14, BJJ18]. Ils constituent un outil de modélisation et de vérification de systèmes à événements discrets, orientés sur les ressources.

Un réseau de Petri est un graphe constitué de *places* (représentées par un cercle), de *jetons* et de *transitions ou d'actions* connectées par des *arcs* représentant une fonction de transition. Les places d'un réseau de Petri peuvent représenter des conditions, des sources ou sorties de données ou encore des ressources. Les transitions, quant à elles, représentent des événement ou des tâches.

Lorsqu'une place dispose d'un nombre suffisant de jetons, une action est alors exécutée. A l'issue de l'exécution d'une transition, le(s) jeton(s) utilisé(s) est/sont alors supprimé(s) et de nouveaux jetons sont générés dans la place de destination atteinte. La fonction de transition spécifie le nombre de jetons à supprimer, pour un arc d'entrée allant d'une place source à une transition et spécifie également les jetons à générer, pour un arc de sortie allant de la transition à la place de sortie.

L'exemple de la figure 2.7 représente une réseau de Petri à trois places (P_0, P_1 et P_2), deux transitions (T_0 et T_1) et quatre arcs ((P_0, T_0), (P_1, T_1), (T_0, P_2) et (T_1, P_2)). Le déclenchement de la transition T_0 consomme le jeton de la place d'entrée P_0 et en génère

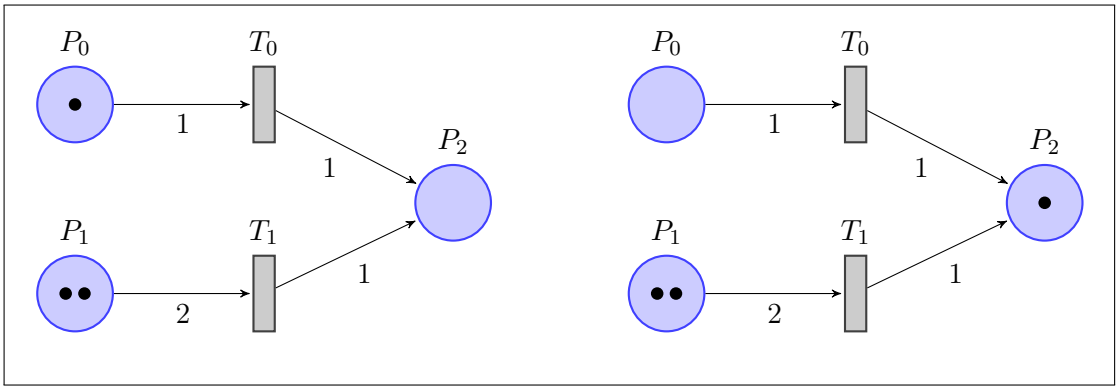


FIGURE 2.7 – Exemple de réseau de Petri à 3 places et 2 transitions. A gauche, le réseau avant l'exécution de T0 et T1. A droite, le réseau après l'exécution de T0 et T1.

un. Le franchissement de la transition T_1 nécessite les deux jetons de la place P_1 et en produit un en sortie. Le nombre de jetons consommés et produits est représenté par le poids des arcs.

Définition 2.2.1. Un **réseau de Petri** est défini formellement par le tuple $\langle P, T, A \rangle$ tel que :

- P : ensemble fini de places ;
- T : ensemble fini de transitions ;
- $A \subset (T \times P) \cup (P \times T)$: ensemble fini d'arcs, associant des places d'entrées aux transitions et les transitions à des places de sorties ;

Le marquage d'un réseau représente l'état du réseau. Ainsi, à chaque place est associé un entier représentant le nombre de jetons présents à un instant donné. On note M_0, le marquage initial du réseau. Ainsi, dans l'exemple de la figure 2.7, $M_0 = \{P_0 \to 1, P_1 \to 2, P_2 \to 0\}$. $M \xrightarrow{t}$ est la notation utilisée lorsque qu'une transition $t \in T$ est franchissable à partir du marquage M. Un nouveau marquage M' est alors atteint. Dans l'exemple de la figure 2.7, $M_0 \xrightarrow{T_0} M' = \{P_0 \to 0, P_1 \to 2, P_2 \to 1\}$.

Des extensions des réseaux de Petri ont été proposées dans [Sif77] pour la gestion du temps au niveau des places (P-TPN) et dans [Ram74] au niveau des transitions (T-TPN).

Définition 2.2.2. Un **réseau de Petri temporisé** est défini formellement par le tuple $\langle P, \Omega \rangle$ tel que :

- P est un réseau de Petri tel que défini à la définition 2.2.1 ;
- Ω est une fonction définie telle que :
  - $\Omega : T \to \mathbf{R}$ pour les réseau de Petri t-temporisé (temps appliqué aux transitions) ;

– $\Omega : P \rightarrow \mathbf{Q}^*$ pour les réseaux de Petri p-temporisé (temps appliqué aux places).

Plusieurs autres extensions des réseaux de Petri ont été proposées dans la littérature et peuvent être utilisées pour la conception d'applications robotiques ou de jeux, comme nous le verrons dans le reste de cette section. Citons notamment Jensen [Jen87] qui a proposé les réseaux de Petri colorés. Ce nouveau type de réseau permet d'affecter une information supplémentaire à chaque jeton : une valeur appelée *couleur*, discriminant chaque jeton. Un mécanisme de hiérarchie est proposé dans les réseaux de Petri colorés permettant de modéliser un système très large en sous réseaux, de taille réduite. Grâce au concept de *fusion de places*, il est possible de spécifier qu'un ensemble de places est identique [Jen98]. Ainsi, lorsqu'un événement se produit sur une place de cet ensemble, l'événement se produit également sur les autres places du même ensemble.

Enfin, [VDAVH04] propose une extension pour la modélisation de *workFlow*, les *WorkFlow nets*. Ils satisfont trois nouvelles propriétés :

- il existe une place unique initiale, possédant uniquement des arcs de sorties et une place unique finale, n'ayant que des arcs d'entrées ;
- un jeton dans la place initiale représente un cas à traiter et un jeton dans la place finale un cas traité ;
- l'ensemble des places P et des transitions T doivent se situer sur un chemin allant de la place initiale à la place finale.

Trois critères de robustesse doivent être satisfaits :

- un seul des jetons présents dans la place initiale peut se retrouver dans la place finale ;
- lorsqu'un jeton apparaît dans la place finale, l'ensemble des autres places ne doivent contenir aucun jeton ;
- aucune transition morte. Chaque transition doit pouvoir être franchie à partir de la place initiale.

2.2.1.2 Approches appliquées aux jeux

La notion de composants, que nous avons décrite en introduction, est pertinente lorsque l'on se place dans le domaine de la conception d'un jeux. Celle-ci s'intègre parfaitement à travers certains concepts propres au domaine. En effet, contrairement à la modélisation des processus d'un logiciel quelconque, celle d'un jeu entraîne la spécification d'activités plus spécifiques, telles que les niveaux ou les quêtes. Des contraintes s'ajoutent également du fait de l'obligation ou non de réaliser certaines tâches pour atteindre un

but. Ainsi, nous retrouvons dans la littérature des travaux où la modélisation d'un jeu est divisée en niveaux ou quêtes, que l'on peut comparer aux approches réductionnistes (ou par composant) quand on se place dans la conception d'activités robotiques.

[DOJSP11] propose, par exemple, une structuration du jeu sous forme de quêtes modélisées par Workflow nets. Chacune des quêtes peut être obligatoire ou optionnelle et représente une suite d'activités pouvant être réalisées de manière séquentielle ou parallèle. Ces activités peuvent elles-mêmes être optionnelles. Afin d'expliquer ces différents mécanismes, les auteurs prennent l'exemple du jeu *The Legend of Zelda* dont le gameplay propose un mélange d'actions, d'aventures et de résolutions de casse tête. Dans cet exemple, le joueur a le choix d'effectuer deux tâches : Semer des graines dans un endroit adéquat et trouver des fées. Le réseau de Petri correspondant à ces deux tâches est donné figure 2.8. Ici, la place P_{20} représente la place initiale et la place P_{33} indique l'objectif à atteindre. Le jeton de la place P_{20} représente le joueur. Les activités *Trouver un endroit pour semer les graines* et *Semer la graine de Gacha* sont des activités séquentielles. Le joueur peut en revanche être à la recherche de la forêt des fées et être au même moment à la recherche du meilleur endroit pour semer ses graines. Cette dernière activité est conditionnée par l'obtention d'une ressource R_2 représentant la graine.

La logique linéaire [Gir87] est ensuite utilisée pour vérifier la robustesse du scénario modélisé. Les règles de conversion du Workflow net en logique linéaire ont été présentées dans [RPCV01].

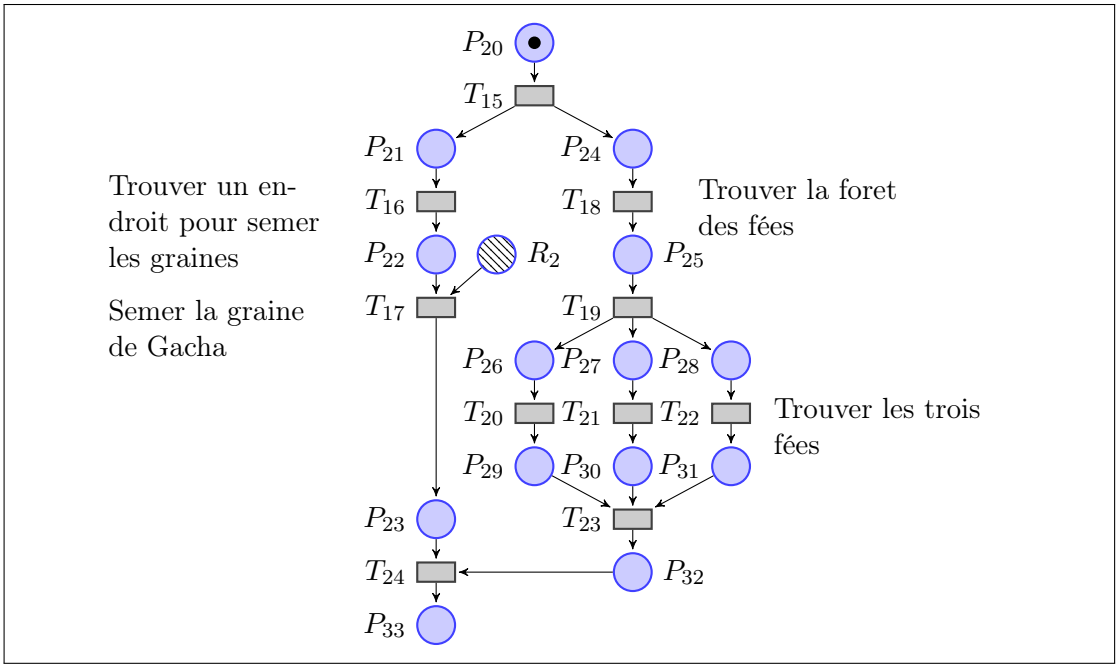


FIGURE 2.8 – Exemple de modélisation d'une quête du jeu *The Legend of Zelda* issu de [DOJSP11]

Analyse de l'approche : *Dans cette approche, l'objectif est de proposer une modélisation d'un jeu dont le scénario est divisé en quêtes (comparables aux composants). C'est ici l'atout majeur de cette approche qui est complétée par l'utilisation d'une représentation graphique : les workFlow nets. Leur utilisation permet de garantir au scénario modélisé une grande fiabilité grâce à la vérification par logique linéaire. Cependant, aucune modélisation temporelle des quêtes ou des activités n'est ici proposée.*

[BJ14] propose d'étendre l'approche précédente en ajoutant, à la modélisation des activités du joueur (par workflow net), une modélisation topologique de l'espace de jeu.

Cette représentation est réalisée par un réseau de Petri où les places représentent les différentes régions où l'action doit se dérouler et le jeton, la position du joueur. Des places supplémentaires peuvent être ajoutées afin de modéliser les conditions d'un passage d'une zone de jeu à une autre (par exemple, l'obtention d'une clef). L'exemple de la figure 2.10 représente un extrait de la topologie du jeu *Silent Hill II*, avec une représentation des conditions de franchissement de frontières et la figure 2.9 un extrait du modèle des activités du joueur. Si cette représentation s'avère efficace, elle n'est pas utilisable seule. En effet, une mise en relation des représentations des activités du joueur et de l'espace du jeu est nécessaire et permettra une gestion des interactions entre le joueur et l'univers virtuel.

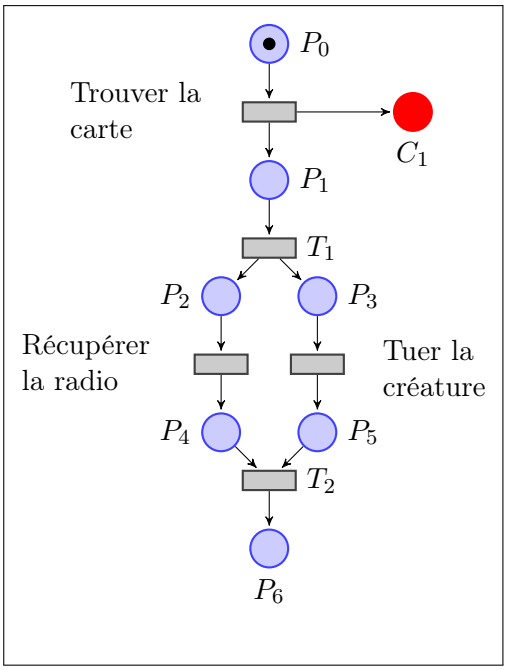


FIGURE 2.9 – Modélisation d'un niveau du jeu *Silent Hill II*

Cette double modélisation permet également d'imposer aux joueurs d'être dans une zone particulière pour effectuer des actions. Les auteurs proposent donc une dernière modélisation pour faire le parallèle entre modélisation formelle des activités du joueur et de l'espace de jeu. Ce parallélisme est mis en oeuvre par un réseau de Petri coloré par l'utilisation du concept des fusions de places. Dans l'exemple précédent, la place C_1 est commune aux modèles des activités du joueur et de l'espace de jeu. Ainsi, après avoir trouvé la carte, un jeton sera généré dans C_1. Automatiquement et en utilisant le principe de fusion de places, ce jeton sera également disponible dans le modèle de l'espace de jeu et permettra au joueur d'accéder à la zone de jeu de la forêt.

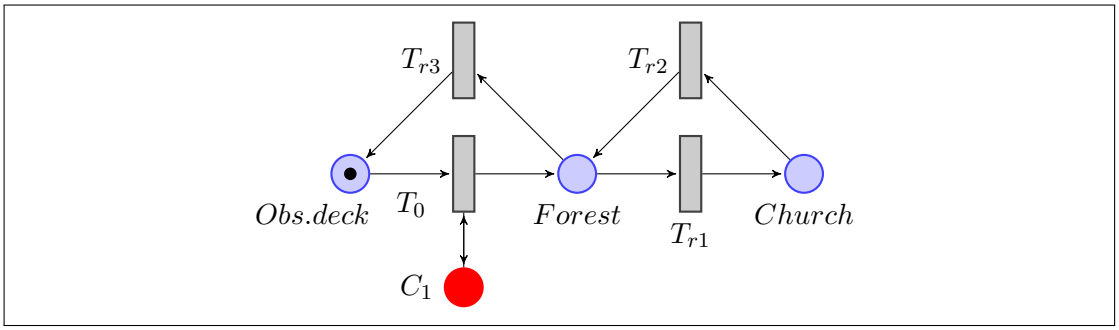


FIGURE 2.10 – Exemple (non complet) de modélisation de l'espace de jeu avec conditions, du jeu *Silent Hill II* issu de [BJ14]

Analyse de l'approche : *Dans cette approche, les auteurs ont proposé une représentation multidimensionnelle de l'environnement des jeux vidéo. Les activités du joueur sont découpées en niveaux. Chaque niveau du jeu est modélisé par un workflow net. L'apport de cette approche par rapport à [DOJSP11] réside dans l'ajout de la gestion de l'espace. L'exemple du jeu Silent Hill II est utilisé pour illustrer l'utilisation d'un réseau de Petri pour la modélisation topologique de l'espace de jeu. Ainsi, le concepteur du jeu peut associer une zone à chaque activité du joueur. La communication synchrone entre la représentation de l'activité et de l'environnement virtuel est réalisée par un réseau de Petri coloré. La vérification de propriétés de robustesse, notamment la vérification de l'accessibilité de chaque zone de jeu ainsi que l'exécution de toutes les activités du joueur, est mis en oeuvre grâce aux algorithmes contenus dans CPN Tools [Jen98]. Cependant, si ce n'est la gestion des ressources, aucun autre contenu n'est ici proposé. Les niveaux et activités ne sont également pas temporisés.*

2.2.1.3 Approches appliquées à la robotique

Les réseaux de Petri et leurs extensions temporisés ont été utilisés pour la modélisation de système à événements discrets et plus particulièrement au domaine de la robotique [KNTU02, LMA06]. Les architectures de contrôle basées sur des modélisations multi-couches, simplifiant l'étape de conception, sont d'ailleurs répandues. Elles prennent en compte la planification, la coordination et l'exécution de tâches.

Ainsi, [CL12] propose une modélisation simplifiée et divisée en trois couches, où chacune décrit les éléments à différent niveau d'abstraction :

- **Environnement** : chaque état de l'environnement est représenté par un réseau de Petri marqué (exemple pour l'environnement d'un ballon à la figure 2.11) ;
- **Exécuteur d'actions** : l'ensemble des actions exécutées sur le robot et modélisées par un réseau de Petri. Une action a un impact sur l'environnement et peut être conditionnée (pre-conditions, running-conditions et success-conditions). Ainsi, une

action de tir effectuée par le robot va avoir pour conséquence une modification du modèle de l'environnement de la balle ;

- **Coordinateur d'actions** : il planifie les actions du robot par un réseau de Petri de plus haut niveau.

Le modèle final est alors obtenu par la composition des réseaux de Petri issus des trois couches précédentes. Le temps nécessaire à la composition est proportionnel aux nombres d'états des différents réseaux. Cependant le temps d'analyse augmente de façon exponentiel par rapport aux nombres d'états. Cette analyse (probabilité qu'une condition soit respectée, temps nécessaire pour effectuer une action...) est réalisée par différents algorithmes proposés dans [Mur89, VN92, BK13]

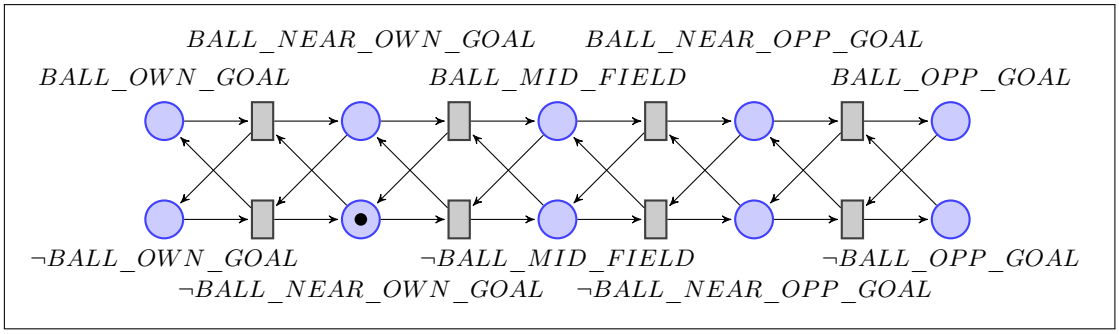


FIGURE 2.11 – Exemple de modélisation de l'environnement issu de [CL12]

Analyse de l'approche : *Comme nous avons pu le voir pour les approches appliquées au jeu, les réseaux de Petri sont particulièrement adaptés à la modélisation de l'espace de jeu ou dans le cadre de la robotique, de l'environnement. Cependant, ce modèle n'a pas fait l'objet d'une expérimentation sur de réels robots. Enfin, la notion de temps n'est ici pas proposée ni la gestion des contenus.*

[LP18] propose *ASPIC*, un *acting system* à base de réseau de Petri dont le modèle final est lui aussi obtenu par composition. Chaque action élémentaire, également nommée *skill*, représente un comportement ou une capacité exécutable par le robot. Chaque comportement dispose d'un flux de contrôle dans lequel trois types de transitions sont proposées : *start, stop* et une transition *except* pour intercepter les exceptions. La planification finale est réalisée par un ensemble d'opérateurs (séquence, concurrence ou branches). ASPIC intègre des algorithmes de vérification permettant d'analyser le système au travers de propriétés de sûretés. ASPIC a été utilisé pour le pilotage d'une embarcation lors d'une mission de protection. La supervision du modèle pour le pilotage de l'embarcation a été réalisée au travers de la plateforme ROS, présentée en section 2.4.

Analyse de l'approche : *L'interaction homme-robot est par définition incertaine. Les conditions de réalisation ne peuvent pas être garanties, si bien que des cas d'erreurs peuvent se produire. Cette gestion est proposée dans cette approche, qui, combinée à une modélisation modulaire, propose une conception efficace des activités robotiques. Cependant, la notion de temps n'est pas supportée.*

La notion de temps a été intégrée à ces modèles pour la supervision de robots footbaleurs dans [PYK16] et [WPK18]. Les comportements des robots sont modélisés par des réseaux de Petri temporisés. Ces comportements sont générés à partir d'un algorithme que les auteurs proposent : COPAM (Colaborative Passing And Movement). Différentes tactiques y sont décrites (tir, passe, dribble...) en fonction de la position de la balle dans les différentes zones du terrain. Le modèle final est obtenu en utilisant la plateforme TINA (TIme petri Net Analyzer) et son outil de simulation.

Analyse de l'approche : *Si la notion de temps est ici intégrée pour la spécification des comportements robotiques, ces approches ne permettent pas de concevoir le modèle de façon modulaire. De plus, l'extensibilité y est assez limitée.*

2.2.1.4 Analyse globale des approches à base de réseau de Petri

Nous venons de présenter plusieurs approches de conception de jeux et d'activités robotiques à base de réseaux de Petri. A travers celles-ci, nous voyons que ce modèle est adapté lorsque l'on adopte une vision orientée ressources. Elles mettent en évidence l'efficacité des approches réductionnistes qui consistent à concevoir les activités indépendemment les unes des autres. Contrairement aux approches semi-formelles, des outils, tel que CPN Tools, ou la conversion en logique linéaire, permettent d'effectuer une analyse sur les modèles.

Nous allons maintenant nous placer du point de vue événements et actions en présentant les approches basées sur des réseaux d'automates.

2.2.2 Approches basées sur les réseaux d'automates

2.2.2.1 Avant propos

Avant d'aborder la modélisation par automates que nous retrouvons dans la littérature, il est nécessaire d'en donner la définition et notamment celle des automates temporisés. Notre définition s'appuie sur les travaux issus de [GLPY97].

Un automate est un graphe dirigé dont les nœuds représentent les localités et les arcs (étiquetés) correspondent aux transitions. Ces dernières représentent l'occurrence d'événements particuliers, entraînant le passage d'une localité source à une localité de destination, ces deux localités pouvant être identiques.

Les automates temporisés sont conçus pour la modélisation et la vérification des systèmes temps-réel. Initialement conçus par Alur et Dill en 1992 [AD92], les automates temporisés ont été étendus par Henzinger, Nicollin, Sifakis et Yovine [HNSY94] en ajou-

tant une nouvelle contrainte sur les localités de l'automate : l'invariant. Ceci impose au système une durée maximale pendant laquelle il peut rester dans la localité actuelle. Un automate temporisé est un automate fini manipulant un ensemble d'horloges définies sur $\mathbb{N}$, dont les valeurs sont incrémentées de façon synchrone. Une transition peut être exécutée si toutes les valeurs d'horloges et de variables respectent toutes les contraintes de garde et d'invariant. Lorsqu'une transition est exécutée, les horloges peuvent être réinitialisées.

S'il existe différentes sémantiques d'automates pour la communication, nous nous intéressons ici aux automates temporisés à entrée/sortie.

Définition 2.2.3. Un **automate temporisé** est défini par le tuple $\langle L, l_0, T, V, I, X\rangle$ tel que :

- L : ensemble fini de localités ;
- $l_0 \in L$: la localité initiale ;
- T : ensemble fini de transitions ;
- V : ensemble fini de variables. Nous notons V^e la valuation des variables V (également appelé vecteur d'états) ;
- X : ensemble fini d'horloges définies sur $\mathbb{R}^+$.
- I : fonction associant à chaque localité $l \in L$ une contrainte d'horloge $i_l(X)$;

Nous notons $t \in T : \langle l, a_t, g_t, r_t, u_t, \lambda_t, l'\rangle$ une transition ayant pour localité source l et pour localité de destination l'. La transition t porte l'étiquette a_t et applique une garde g_t composée de $g_t(V)$ - garde appliquée sur les variables V - et $g_t(X)$ - garde appliquée sur les horloges X. $r_t \subseteq X$ est l'ensemble des horloges à réinitialiser et u_t une fonction de valuation des variables V. Enfin, λ_t représente un signal de synchronisation. Ainsi, *signal*! est un signal d'émission et *signal*? est un signal de réception.

Un invariant $i \in I$ est une contrainte de temps appliquée à la localité. Le temps s'écoule au plus m unités de temps dans la même localité. Un invariant est de la forme $x \sim m$ avec $x \in X$, $\sim \in \{<, \leq\}$.

Une garde temporelle est une contrainte sur le franchissement d'une transition $t \in T$. Le temps s'écoule au moins n unités de temps. Une garde temporelle est de la forme $x \sim n$ avec $x \in X$, $\sim \in \{>, \geq\}$, et m $\leq$ n.

Dans l'exemple de la figure 2.12, le premier automate possède une transition ayant pour étiquette a. Cette transition a pour condition d'exécution une garde appliquée sur la variable nb dont la valeur doit être inférieure à 4. Elle est également composée d'une contrainte temporelle sur l'horloge y. Ainsi, cette transition doit être exécutée dans l'intervalle $[2, 4]$ unité de temps (y$\leq$4 étant l'invariant appliqué à la localité S_0 et $y \geq 2$ la garde temporelle de la transition). Enfin, à l'issu de l'exécution de la transition, la variable nb sera incrémentée, l'horloge y sera remise à zéro et le signal c sera envoyé. Le second automate est en attente de réception du signal c, qu'il doit recevoir dans l'intervalle de temps $[1, 8[$.

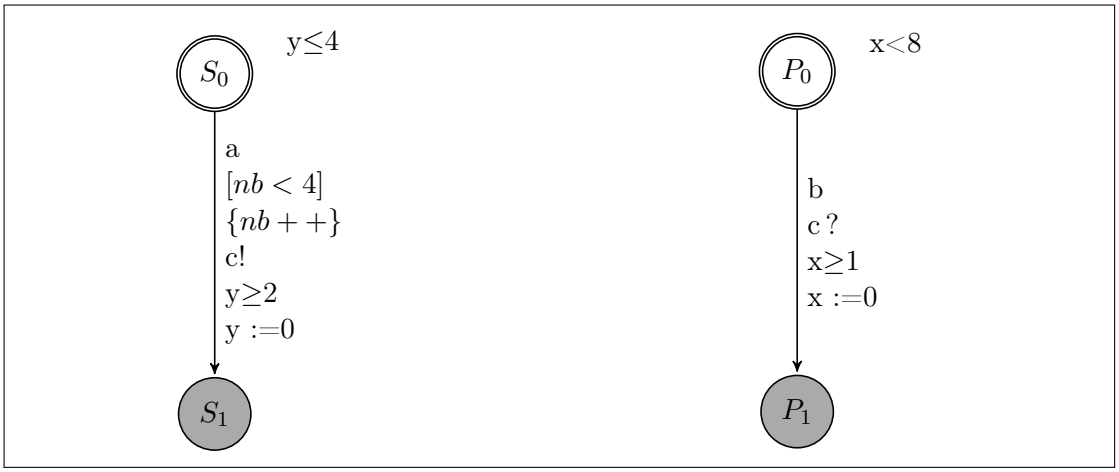


FIGURE 2.12 – Exemple d'automates temporisés communicants

2.2.2.2 Approches appliquées aux jeux

Lors de précédents travaux de thèse réalisés au L3i, un premier modèle haut niveau de modélisation de scénario à base d'automates communiquants a été proposé [RPE+09, Rem13]. Afin de réduire la complexité liée à la modélisation du scénario, cette approche propose une conception automatique du scénario par la définition d'un ensemble de comportements génériques et de scènes d'exécution. Le processus de création d'un scénario interactif est ici décomposé en trois couches (figure 2.13).

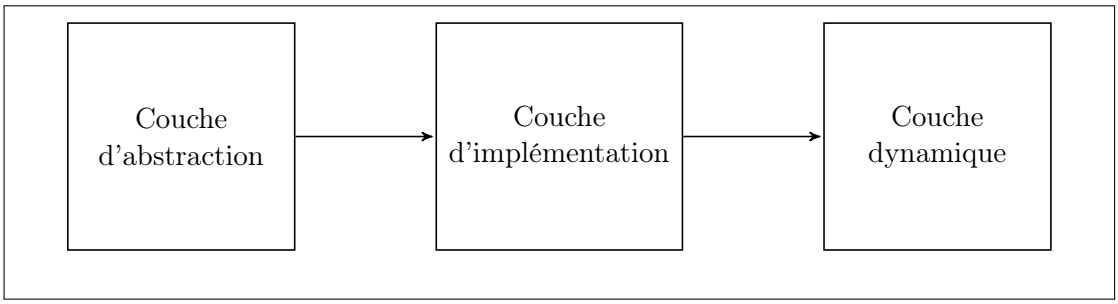


FIGURE 2.13 – Modèle à trois couches de [Rem13]

La **couche d'abstraction** est la couche dans laquelle les entités génériques du scénario sont décrits. Ainsi, les différents *comportements* sont créés et modélisés par un automate. Ces entités sont ensuite regroupées au sein d'*actants*, modélisés à leur tour par un automate. Leurs automates sont construits par le produit synchronisé des différents automates de comportements qui les composent. Enfin, ces actants sont mis en relation au sein de *situations*. Dans un deuxième temps, la **couche d'implémentation** permet, par un mécanisme d'héritage multiple, d'implémenter les *actants* en *acteurs* et les *situations* en *scènes*. Chaque entité est également modélisée par un automate, résul-

tat du produit synchronisé des entités qui la compose. Enfin, la **couche dynamique** ordonne les *scènes*, encapsulées dans des *jalons*. Cette sur-couche spécifie les éléments nécessaires à la réalisation de la scène.

Exemple : Considérons un exemple simple, pouvant se dérouler dans un western et faisant intervenir quatre actants : un cavalier, un cheval, un tireur et un passant. Trois situations peuvent alors apparaître : une balade, une fusillade ou une course.

Dans l'exemple de la figure 2.14, la situation de balade est représentée. Ainsi, les actants *Cavalier* et *Cheval* sont impliqués. Le cavalier possède les comportements *Monter, Descendre, Tirer sur les rênes* et *Eperonner*. Ces deux derniers comportements auront pour conséquence l'exécution des comportements *Ralentir* et *Accelerer*, respectivement, de l'actant cheval, par le mécanisme de synchronisation, proposé dans les réseaux d'automates au travers des *channels*.

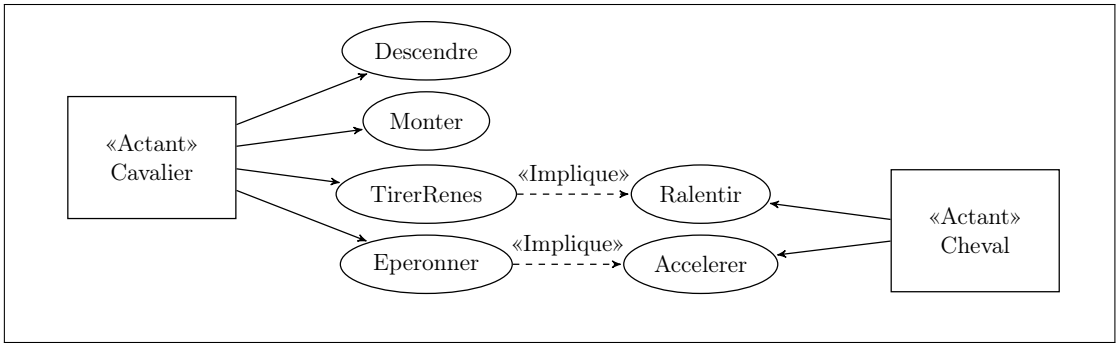


FIGURE 2.14 – Exemple de situation mettant en relation les actants "Cavalier" et "Cheval"

La représentation sous forme d'automate de la situation *Balade* est donnée figure 2.15.

Ces entités génériques réutilisables sont ensuite instanciées. Ainsi, l'acteur *Lucky L.* peut implémenter les rôles de *Cavalier* et de *Tireur*. Par héritage multiple, cet acteur possédera à la fois les comportements d'un cavalier mais également les comportements d'un tireur. Les différents acteurs ainsi créés vont ensuite être mis en relation au sein de scènes, héritant elles mêmes d'une situation de la couche d'abstraction. La dernière étape consiste alors à ordonnancer ces scènes grâce au jalons pour déterminer l'aspect dynamique du scénario. L'exemple de la figure 2.16 représente la séquence de jalon d'un scénario western.

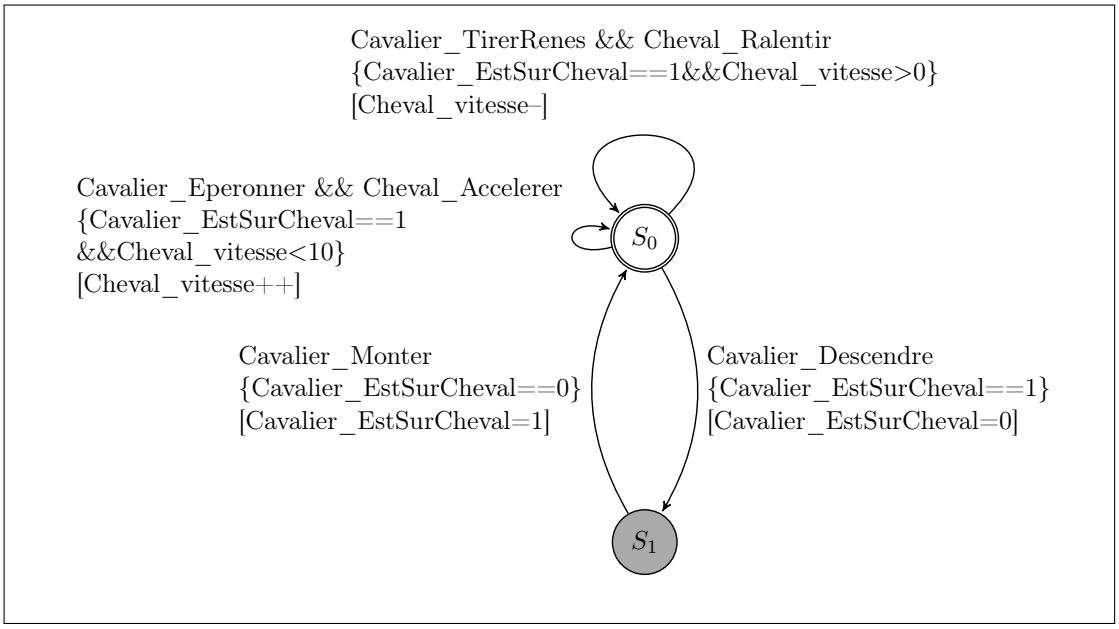


FIGURE 2.15 – Représentation sous forme d'automate de la situation *Ballade*

Analyse de l'approche : *Ces travaux adressent la problématique de conception et de supervision des narrations interactives. Un enjeu majeur porte ici sur la facilité d'extension ou de modification que ce soit au niveau de la structure ou au niveau des entités narratives. La structuration en trois couches de cette approche permet de répondre à cette problématique. Les concepts d'extensibilité et de réutilisation des entités génériques par héritage multiple sont un atout majeur. Bien que cette approche ait prouvé son efficacité, la conception d'un scénario nécessite de longues étapes qui peuvent être un frein pour le concepteur. Enfin, cette contribution ne répond pas à l'ensemble des critères adressés par cette thèse. En effet, la dimension temporelle n'est pas prise en compte, que ce soit au niveau des comportements ou au niveau des scènes pour le passage de l'une à l'autre.*

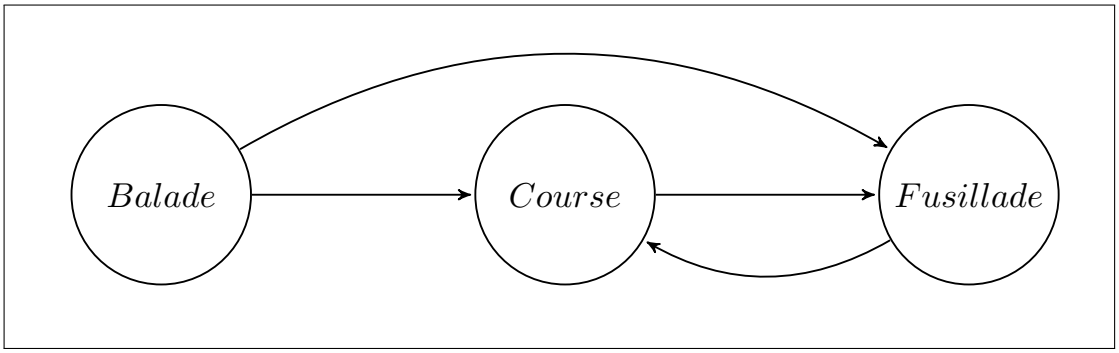


FIGURE 2.16 – Exemple d'une séquence de jalons

Enfin, la gestion des contenus n'est ici pas supportée.

2.2.2.3 Approches appliquées à la robotique

L'utilisation des automates est également présente pour la conception de plateformes robotiques. Ainsi, [WLG+14] propose un système de planification multi-couches destiné à la robotique (figure 2.17) dans lequel les comportements du robot et sont environnement son modélisés par des réseaux d'automates temporisés.

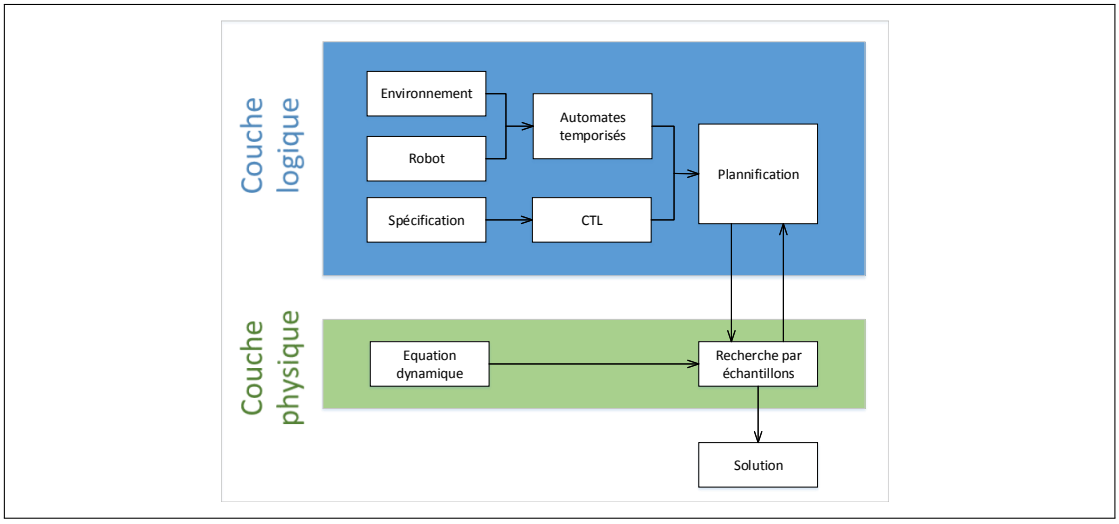


FIGURE 2.17 – Architecture de l'approche de [WLG+14]

La couche logique est en charge de construire la représentation de l'environnement et des comportements du robot par des automates temporisés. Les propriétés de vérification sont exprimées en logique CTL (décrite en détail dans la section 4.2.1) et sont vérifiées avec l'outil UPPAAL (décrit en section 4.2.2). Deux types de propriétés sont ici vérifiées :

- **propriétés d'accessibilité :** tous les robots doivent pouvoir atteindre leur destination ;
- **propriétés de sûreté :** l'évitement d'obstacles et la durée d'exécution.

La vérification de ces propriétés par UPPAAL permet de générer une trace, satisfaisant l'ensemble des contraintes. Ces traces sont alors utilisées dans la seconde couche, la couche physique, qui va générer une trajectoire pour chaque robot sur la base d'équations dynamiques. Une trace de contre exemple peut également être générée lorsqu'une contrainte n'est pas respectée. Ce procédé a d'ailleurs été utilisé dans [RPE+09] pour la vérification de scénario conçus par l'approche décrite en section 2.2.2.

Ces travaux ont été étendus dans [WGS+18]. En effet, jusqu'ici l'approche proposée est restée théorique et n'avait pas fait l'objet d'expérimentations. Une nouvelle approche en quatre couches a donc émergé (figure 2.18) :

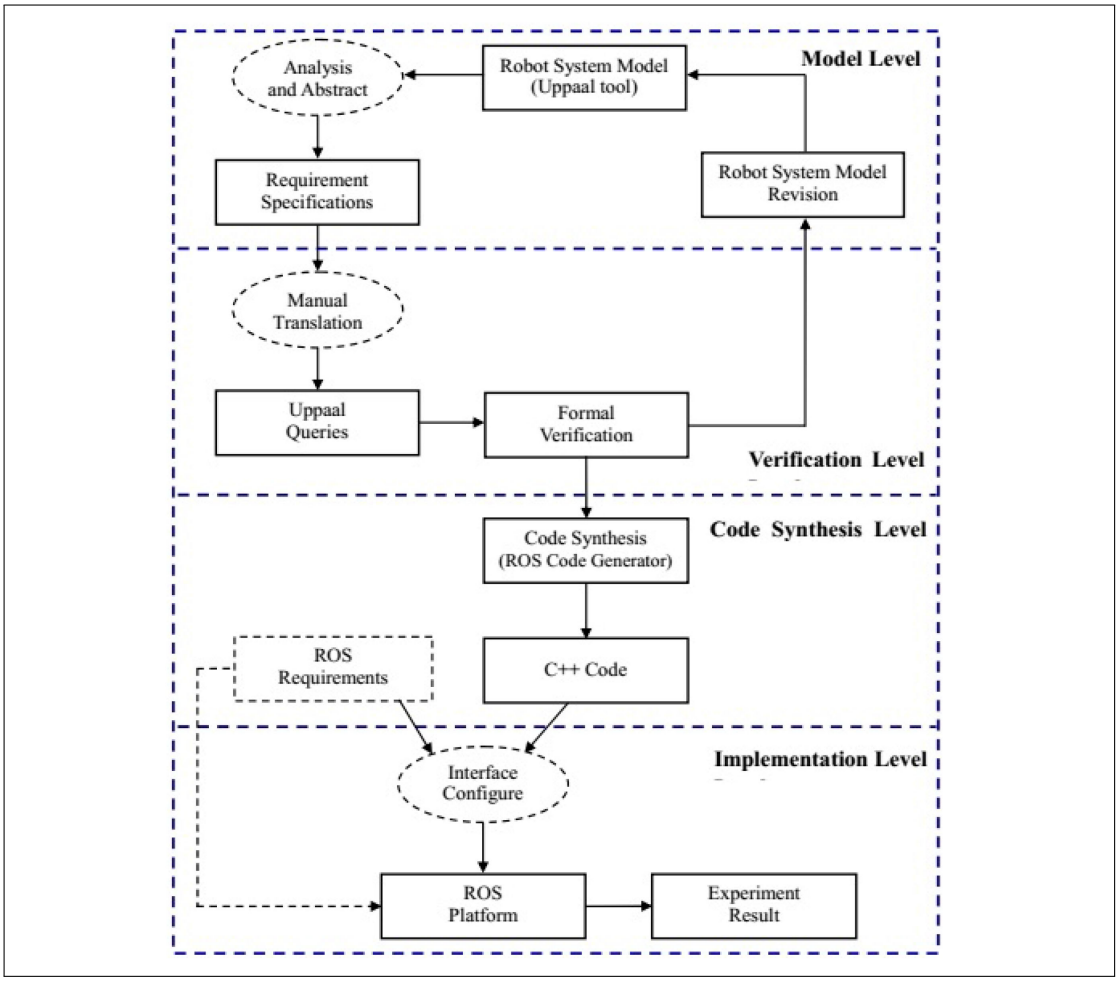


FIGURE 2.18 – Conception d'un système robotique issu de [WGS+18]

- la couche "modèle" que l'on peut comparer à la couche logique de l'approche précédente ;
- la couche de vérification sous forme de requête CTL sous UPPAAL ;
- la couche de génération de code. Dans celle-ci, le générateur produit automatiquement le code C++ pour la plateforme ROS (décrite en section 2.4) ;
- la couche d'implémentation permettant l'exécution du code sur le simulateur Gazebo.

Analyse de l'approche : *Cette approche propose une conception multi-couches et utilise les automates temporisés pour la modélisation de comportements robotiques et de l'espace. L'utilisation de UPPAAL permet de vérifier des propriétés temporelles, de*

sûreté et d'accessibilité. La consistance comportementale est assurée par la conversion du formalisme des automates temporisés en code C++. Si la gestion de l'espace est ici présente, aucune gestion de contenu n'est supportée.

2.2.2.4 Analyse globale des approches à base de réseau d'automates

A travers les approches que nous venons de présenter, nous remarquons que les automates, grâce aux différents mécanismes de composition et de synchronisation, sont adaptés à la modélisation modulaire des activités. Leurs formalisme permet également d'appliquer une vérification du modèle afin de prouver leur robustesse. Enfin, la possibilité de pouvoir contraindre temporellement les comportements est un atout majeur quand on sait l'importance de ces paramètres dans le domaine des interactions homme-robot.

2.3 Comparaison des approches de modélisation

Nous proposons, dans la figure 2.19, une représentation permettant de comparer les différentes approches que nous venons de présenter dans les sections précédentes. Cette représentation s'attache à évaluer les approches sur les trois critères suivants :

- l'aspect formel ou semi-formel propre au type de modèle utilisé ;
- la prise en compte du temps dans la représentation du système ;
- la prise en compte des différents espaces de l'interaction dans le modèle.

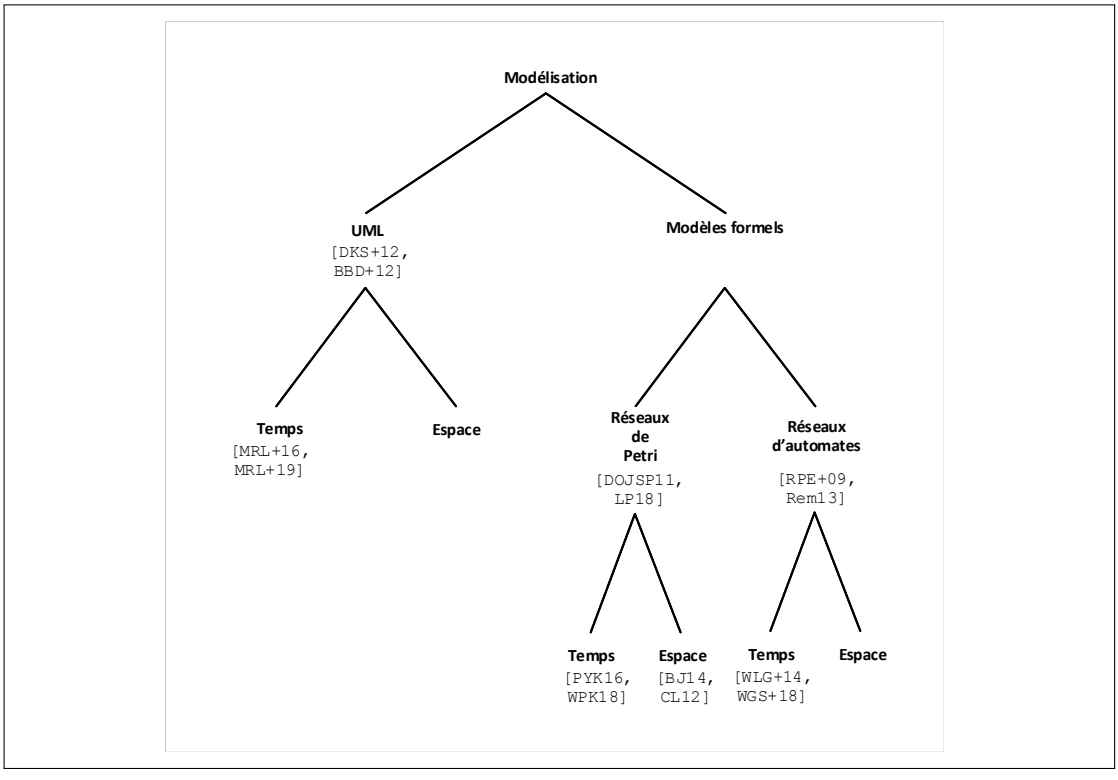


FIGURE 2.19 – Comparaison des approches de modélisation

Le tableau 2.1 présente une évaluation détaillée des différentes approches.

	Extensibilité	Vérifiabilité	Modularité	Pilotage de l'activité
[DKS+12]	+	impossible	++	via OROCOS
[MRL+16, MRL+19]	+	via FDR	++	impossible
[BBD+12]	+	Alloy et LTL	++	via OROCOS
[DOJSP11]	++	logique linéaire	++	impossible
[BJ14]	++	via CPN tools	++	impossible
[CL12]	++	oui	++	impossible
[LP18]	++	oui	++	via ROS
[RPE+09, Rem13]	+++	UPPAAL (traces de contre exemple)	+++	impossible
[WLG+14, WGS+18]	++	UPPAAL (traces de contre exemple)	+++	via ROS

TABLE 2.1 – Analyse des approches de modélisation

Les approches que nous trouvons dans l'état de l'art sont principalement basées sur des modèles spécifiques aux questions abordées (planification, narration...). Celles ci reposent sur des modèles formels permettant la vérification de propriétés. L'état de l'art montre la nécessité de disposer d'outils de modélisation efficace permettant une conception modulaire du système ou du scénario. Si la modélisation UML facilite la compréhension et la lecture d'un système, la vérification de certaines propriétés telles que la sûreté ou l'accessibilité s'avère parfois difficile voire impossible. Les approches à base de réseaux de Petri sont, quant à elles, orientées ressources. Notre problématique nous imposant plutôt une représentation par actions des comportements robotiques, notre choix de formalisme pour la spécification des comportements robotiques s'est donc orienté vers une représentation par automates temporisés.

Concevoir de manière formelle une activité robotique garanti au concepteur d'obtenir un modèle répondant à ses attentes. Basée sur des mathématiques, une représentation formelle de l'activité permet également de s'assurer, à travers des techniques de *model-checking*, que le modèle répond à des critères de sûreté. Il est alors envisageable de piloter un processus quelconque et notamment une plateforme robotique. La section suivante sera donc consacrée à la présentation d'outils de supervision d'applications robotiques.

2.4 Supervision et pilotage d'applications robotiques

La supervision d'une application robotique consiste, à partir d'un modèle, à déterminer et exécuter toutes les actions adéquates en fonction du contexte dans lequel elles se déroulent.

Dans cet optique, nous présentons quatre des principaux frameworks de supervision :

- GenoM : Generator of Modules ;
- BIP : Behavior Interaction Priorities ;
- ROS : Robot Operation System ;
- Choregraphe.

2.4.1 GenoM

GenoM [FHC97] est un framework de conception par composants pour robot. Chaque composant de GenoM peut être vu comme un serveur, disposant de ports d'entrée et de sortie afin de communiquer avec d'autres composants. Ils proposent des services (asynchrone) pouvant s'exécuter en parallèle et modélisés par un automate à états finis temporisé ou non. Les algorithmes implémentés dans les services sont décomposés en fragments et sont appelés *codels*. L'architecture typique d'un composant GenoM dans sa troisième version, issu de [Fou18], est donné figure 2.20.

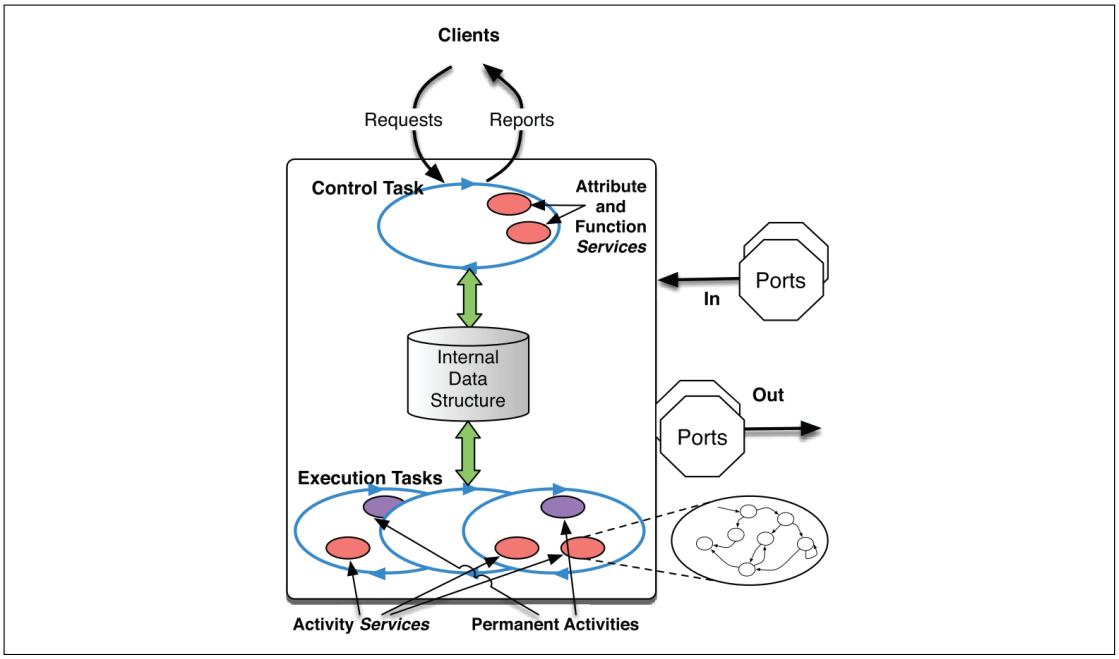


FIGURE 2.20 – Architecture d'un composant GenoM3 [Fou18]

Le contrôle d'un composant est réalisé par le *Control Task*. Il assure la gestion des clients effectuant des requêtes sur le composant et gère la réponse au travers de *reports*. Les *Execution Tasks* sont des structures en charge des traitements à travers l'aggrégation de services modélisés par des automates. Chaque localité de l'automate est un *codel* et permet l'exécution d'un algorithme particulier. La spécification de l'ensemble du composant est réalisé dans un fichier *.gen*.

Analyse de l'approche : *Cette approche a prouvé son efficacité par son déploiement sur des robots réels (quadcopter et Osmosis). Elle offre une modélisation formelle d'applications robotiques et intègre une gestion du temps avec l'utilisation des Timed Transitions Systems, TTS. Cependant, la spécification des composants peut s'avérer être une tâche complexe pour un non initié.*

GenoM a d'ailleurs été utilisé dans l'architecture LAAS (figure 2.21) [ACF+98] qui repose sur une représentation en trois niveaux :

- le niveau décisionnel en charge notamment de la planification de tâches et de la reconnaissance de cas d'erreurs. Il supervise les actions et réagit aux événements venant du deuxième niveau ;
- le niveau de contrôle d'exécution vérifie les requêtes envoyées au troisième niveau et gère les ressources ;
- le niveau fonctionnel intègre les différentes actions et fonctions de perception de l'agent. Chaque module est ainsi conçu grâce à GenoM.

D'autres architectures, également orientées actions, reposent sur des représentations à trois couches. Dans de tels architectures, la couche de haut niveau est chargée de la planification et de l'ordonnancement ; la couche intermédiaire exécute le modèle proposé en première couche et la dernière couche gère le niveau matériel. Cette représentation se retrouve dans les architectures MOBAR (Model Based Architecture) [MRMBR19], CLARAty [ANSG+06], Saphira [KM98], RA (Remote Agent) [MNPW98] ou ATLANTIS [Gat92].

2.4.2 BIP : Behavior Interaction Priorities

BIP [BBS06] propose une modélisation par composants modélisés par automates à états finis temporisés. Chaque composant est conçu par une spécification en trois couches :

- Behaviors ;
- Interactions ;
- Priorities.

La couche *Behaviors* permet de définir les composants atomiques du système. Il se compose d'un ensemble de *ports* permettant la synchronisation avec d'autres composants, de *control states*, de variables et de transitions représentant les différentes fonctionnalités

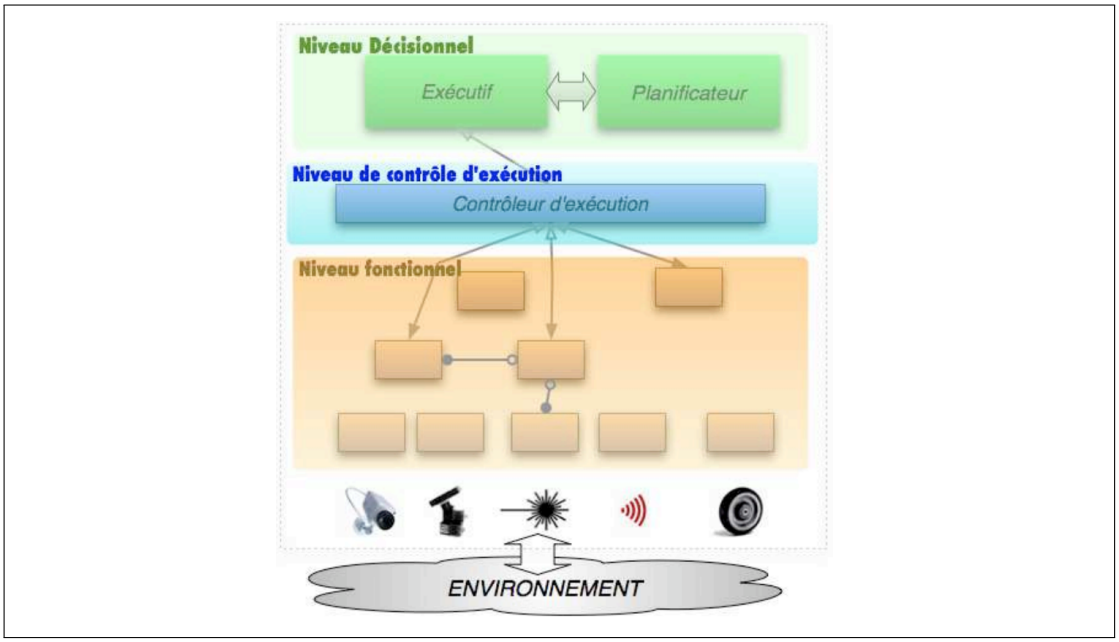
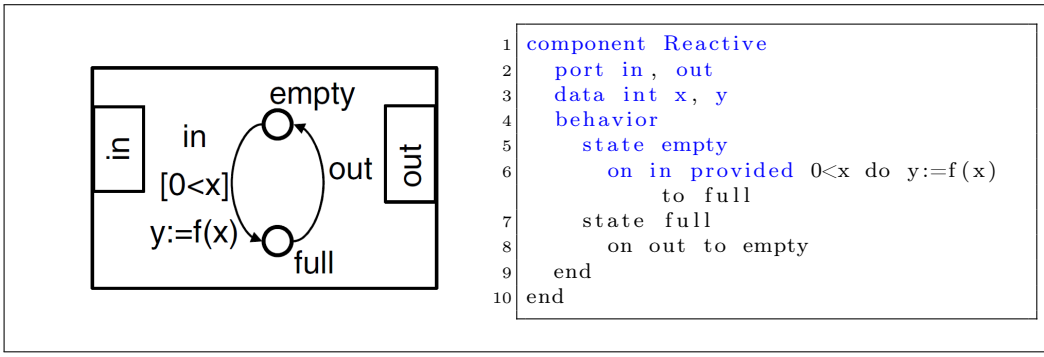


FIGURE 2.21 – Architecture LAAS issu de [Clo07]

du composant. Un exemple de représentation issu de [BBS06] est donné à la figure 2.22. Dans celui-ci, deux ports *in* et *out* ont été créés ainsi que deux *control states*, *empty full* et deux variables x et y.




```
1  component Reactive
2    port in, out
3    data int x, y
4    behavior
5      state empty
6        on in provided 0<x do y:=f(x)
             to full
7      state full
8        on out to empty
9    end
10 end
```

FIGURE 2.22 – Exemple de composant BIP et de sa spécification textuelle issu de [BBS06]

Dans la couche *Interactions*, un ensemble de connecteurs est créé pour établir une connexion entre les différents composants du système. Un ensemble de règles peut être appliqué sur les connecteurs : les priorités. Ainsi, lorsque deux interactions sont possibles, celle possédant la plus haute priorité sera exécutée. Cette mécanique permet de concevoir des systèmes complexes de manière hiérarchique à l'aide de composants composés, qui encapsulent des sous-systèmes constitués de composants par l'utilisation de connecteurs et de priorités. BIP propose également sa propre plateforme d'exécution.

2.4.3 ROS : Robot Operating System

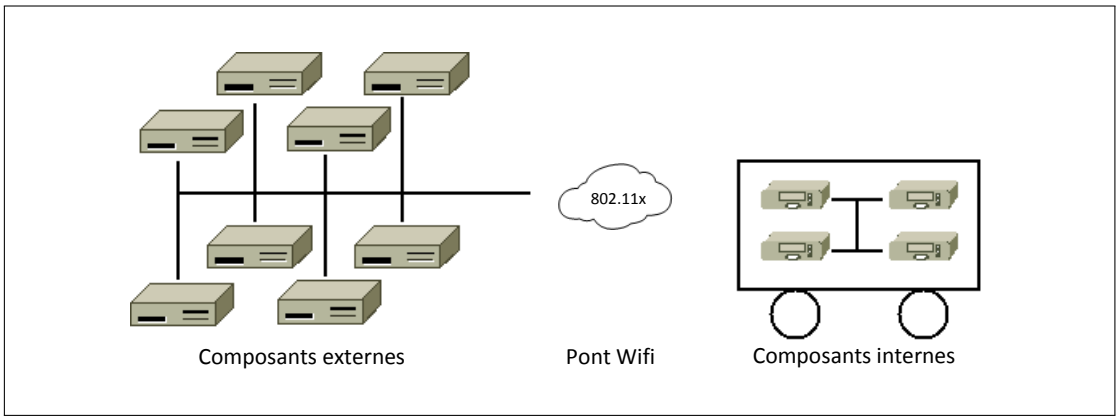


FIGURE 2.23 – Architecture réseau typique de ROS [QCPG+09]

L'objectif de ROS [QCPG+09] est de fournir une architecture (figure 2.23) permettant l'exécution parallèle de plusieurs processus à destination d'une plateforme robotique. Chaque processus est vu comme un noeud pouvant communiquer, avec d'autres noeuds, par échange de message synchrone (utilisation d'un service) ou asynchrone (sur le principe des *publisher/subscriber*) au travers d'un *master*. Les messages échangés sont discriminés par sujet (topic). Ainsi, chaque noeud s'abonne à un sujet particulier et effectue un traitement propre sur les données reçues.

La philosophie mise en avant par ROS repose sur cinq points :

- **Peer to Peer** : Un système robotique développé avec l'architecture ROS est composé d'un certain nombre de processus, potentiellement hébergés sur différentes machines (afin de réaliser des tâches complexes, comme de la reconnaissance d'images ou vocale). Chaque processus communique alors de façon synchrone ou asynchrone grâce à une architecture Peer to Peer (utilisation de XML-RPC) ;
- **Multi langages** : ROS n'est pas dépendant d'un langage particulier : possibilité de programmer en C++, Python, Lisp ;
- **Basé sur des outils** : basé sur un design de micro kernel, ROS utilise des petits outils pour réaliser différentes tâches : modification de paramètres de configuration, visualiser la topologie du réseau Peer to Peer, effectuer le *build*... ;
- **Léger** : Afin de palier à la dépendance de certains drivers ou algorithmes, ceux-ci sont embarqués dans des bibliothèques indépendantes du framework ROS ;
- **Open source**.

ROS est devenue l'une des plateformes les plus populaires et utilisée dans l'univers de la robotique. Elle est compatible avec de nombreux robots et notamment avec les robots Nao.

Analyse de l'approche : *ROS a pour avantage majeure de permettre une exécution parallèle de plusieurs composants et propose une communication inter-processus. L'application robotique finale est construite en paramétrant chacun des noeuds du système. L'abstraction du matériel est également intéressante, ce qui permet de proposer des fonctionnalités génériques.*

2.4.4 Choregraphe

Choregraphe (figure 2.24) est l'outil de programmation graphique proposé par Aldebaran pour piloter les robots humanoïdes Nao et Pepper. La conception d'une activité sous cet outil est réalisée par un développement purement linéaire. C'est un outil simple de combinaison de comportements prédéfinis, avec pour langage support Python. Les comportements pré-existants et livrés avec Nao peuvent être modifiés (puisque le code python est accessible à partir de Choregraphe) et regroupés au sein de structure de plus haut niveau permettant de concevoir l'activité de manière hiérarchique.

Le langage visuel de Choregraphe se traduit à travers des comportements. Chaque comportement est défini par un nom, par une/des entrée(s), une/des sortie(s) et un/des paramètre(s) et généralement un script Python, contenant les commandes à exécuter. Lorsqu'un signal est envoyé en entrée, le comportement est chargé puis exécuté. Une fois la sortie stimulée, il est déchargé.

Choregraphe gère des contenus sonores, que ce soit à travers les discours des robots ou à partir de sons diffusés par les hauts parleurs. Le robot Pepper permet, grâce à sa tablette, de pouvoir proposer des affichages, notamment vidéos.

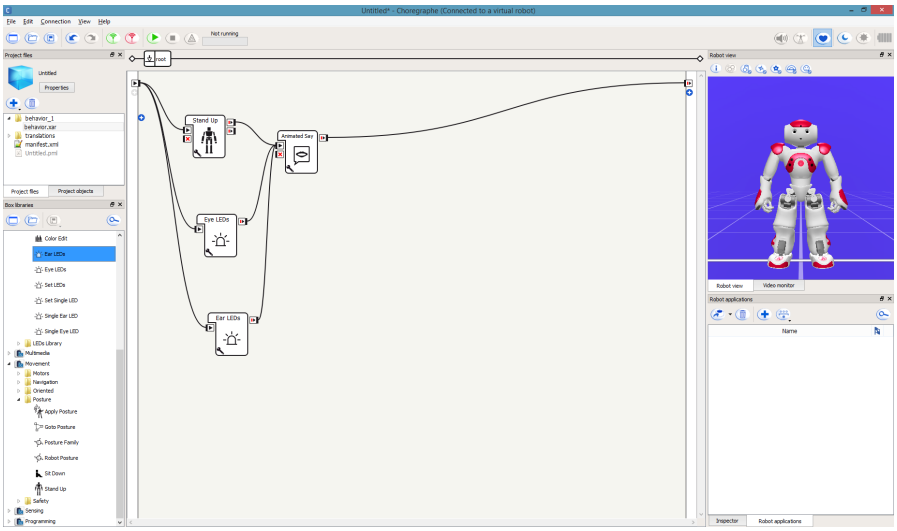


FIGURE 2.24 – Exemple de séquence Choregraphe

L'outil propose une certaine abstraction visuelle et utilise donc son propre langage qui est ensuite interprété par NAOqi sur le robot. NAOqi est une librairie fournie par

le constructeur des robots et elle est disponible en Python, C++ et Java. Cette librairie permet d'utiliser l'ensemble des capteurs et moteurs dont est équipé Nao ainsi que ses fonctionnalités de reconnaissance de visage ou vocale par exemple.

Analyse de l'approche : *Si Choregraphe s'est révélé efficace pour la conception de nos premières expérimentations au musée, décrites au chapitre introductif de ce manuscrit, il ne permet pas de définir un scénario, en tenant compte de l'espace pour lequel il a été conçu. L'objectif est de pouvoir déployer une même activité dans différents lieux avec un effort de traitement minimal. Pour qu'une expérience puisse être adaptée à divers lieux, le contenu de l'expérience doit également être modifié. Bien que la conception soit relativement simplifiée par l'utilisation des comportements pré-programmés, les modifications éventuelles des dialogues et des contenus ou la reprogrammation des interactions peuvent se révéler être une tâche risquée et hasardeuse lorsqu'ils sont intervenus peu de temps avant une session avec un joueur. La dernière limitation concerne la temporalité de l'expérience. Nous souhaitons pouvoir appliquer une période de temps à un événement particulier du scénario, par exemple une limite de temps pendant laquelle l'utilisateur doit répondre à la question posée par le robot.*

2.5 Conclusion

Nous avons présenté dans ce chapitre différentes approches de modélisation de scénario interactifs et identifié notamment les approches à base de réseaux de Petri, d'UML et d'automates. Ces trois types de modélisation offrent l'avantage majeur d'être graphiques et donc facilement lisibles et compréhensibles. Cependant, les automates étant orientés actions et disposant d'algorithme de composition synchronisé sont plus adaptés aux problématiques scientifiques énoncés ; nous nous orientons donc vers cette approche.

Nous détaillons, dans les chapitres suivants notre proposition permettant de répondre aux problématiques énoncées au chapitre introductif.

Deuxième partie

Approche proposée

Chapitre 3

Modéliser les Contenus, l'Interaction et le Temps

Sommaire

Nous avons présenté, dans le chapitre précédent, un état de l'art dans lequel différentes méthodes de conception et de modélisation ont été détaillées. Bien que ces approches s'avèrent efficaces, nous avons pu établir que les modèles sur lesquels elles se basent n'adressent pas l'ensemble des critères énoncés dans notre problématique, présentés en introduction.

L'objectif de ce chapitre tient dans la proposition d'un modèle générique prenant en compte l'ensemble des dimensions de l'interaction : la représentation des contenus,

l'interaction (au travers des comportements), le temps et l'espace. Un premier modèle, CITE, proposé dans [PR17] avait pour ambition de prendre en compte l'ensemble des dimensions énoncées précédemment. Bien qu'à l'origine, ce modèle prévoyait la prise en compte de l'espace dans la conception d'un scénario, cette problématique constitue, entre autre, une intégration future à envisager pour notre modèle mais n'entre pas dans le cadre de ces travaux de thèse.

3.1 CIT : principes

Le modèle présenté dans ces travaux vise à améliorer les conditions de production d'une scénarisation interactive (notamment pour les applications robotiques) et se doit de présenter les caractéristiques suivantes :

1. **Généricité** : l'enjeu est ici de proposer un modèle adapté à la représentation et à la supervision de tout type d'activité interactive. Ainsi, notre modèle permet la conception de différents types d'applications (application interactive sur robots, jeux-vidéo...) ;

2. **Réutilisabilité et extensibilité** : outre le fait de pouvoir utiliser cette modélisation pour tout type d'activité, il s'avère nécessaire de pouvoir étendre ou modifier une scénarisation donnée pour l'utiliser dans un autre contexte. Il peut par exemple s'agir de reproduire un mini-jeu d'un musée dans un autre lieu culturel en modifiant les textes et contenus manipulés ou éventuellement d'ajouter de nouvelles interactions ;

3. **Temporalité** : l'interaction Homme/Machine est régie par des échanges et des processus de collaboration et de rétroaction nécessitant une certaine fluidité. Cet enjeu spécifique pousse alors à contraindre temporellement les interactions pour garantir une bonne qualité. Le modèle proposé se doit donc d'intégrer les mécaniques nécessaires à la prise en compte de délais et de gardes temporelles.

4. **Modularité** : La représentation de la scénarisation peut invoquer un nombre important d'entités (qu'ils s'agissent d'entités virtuelles, de modules logiciels du robot par exemple...). Ces dernières interagissent avec un couplage qui peut être fort et complexifier la tâche de modélisation ou de modification. Le modèle proposé ici vise à permettre une modularité dans la modélisation en autorisant une description individualisée des entités et de leurs comportements. La composition de ces derniers au sein d'entités de plus haut niveau va alors simplifier la conception ;

5. **Vérifiabilité** : la scénarisation obtenue pour la représentation de l'activité peut s'avérer complexe et nécessite une étape de validation avant son exécution avec les usagers. Cela est d'autant plus vrai, dans un contexte où la modélisation a pu être facilement étendue et modifiée. Ainsi, le modèle se doit de permettre une vérification formelle augmentant la confiance en la qualité de ce scénario. CIT propose une

approche entièrement basée sur les automates temporisés à entrées/sorties pour lesquelles des approches de model-checking sont applicables.

De plus, et dans l'optique d'atteindre les avantages visés, notre modèle **CIT** (**C**ontenu, **I**nteraction et **T**emps), envisage la modélisation de l'activité interactive sous la forme d'une modélisation multi-dimensionnelle basée sur :

- **les contenus** : l'ensemble des contenus utilisables dans le scénario. Ces contenus peuvent être de différentes formes : texte, son, image, vidéo ;
- **l'interaction** : les différents échanges réalisables entre l'homme et la machine, modélisés par des réseaux d'automates temporisés à entrées/sorties ;
- **le temps** : la gestion du temps, qui est pris en compte au niveau de l'interaction ou du contexte.

Nous avons présenté dans l'état de l'art, une approche de conception de scénario à base d'automates à entrées/sorties, proposée dans le cadre de précédents travaux de thèse menés au L3i [Rem13] (section 2.2.2). Cette approche propose une modélisation modulaire de l'activité dont la phase de conception est divisée en trois couches. Chaque couche permet de créer différentes entités sur la base d'entités créées dans des couches plus basses. Si cette conception modulaire est efficace, elle nécessite de longues étapes qui peuvent être un frein.

Nous proposons donc une refonte totale de ce modèle avec la proposition du modèle CIT. A travers ce nouveau modèle, nous réduisons le nombre de couches et d'entités nécessaires à la conception d'un scénario. De plus, nous proposons une gestion externalisée des contenus et la prise en compte des contraintes temporelles au travers de l'utilisation des automates à entrées/sorties temporisés.

Comme décrit précédemment, l'un des enjeux de la modélisation de l'interaction est de tenir compte des différentes dimensions de la scénarisation de l'activité : les contenus, l'interaction et le temps.

3.1.1 Une vision modulaire de l'Interaction

Pour simplifier la tâche de conception, nous envisageons, ici, un modèle permettant d'une part une conception haut niveau de scénario (au travers du graphe de contexte) et d'autre part une description modulaire des agents (par agrégation de comportements dans des patterns et dans des agents). La figure 3.1 présente le principe de création des comportements, des patterns et des agents.

La description du système tient dans la description des comportements atomiques, regroupés en patterns. Le comportement est défini comme un automate temporisé et l'agrégation de ces comportements en agents est réalisée par le produit synchronisé des différents automates de comportements ou de patterns qu'il agrège. Cette mécanique modulaire de description des comportements facilite leur ajout ou leur retrait au sein d'un agent et réduit la tâche du concepteur puisque l'automate global du comportement de l'agent est produit automatiquement.

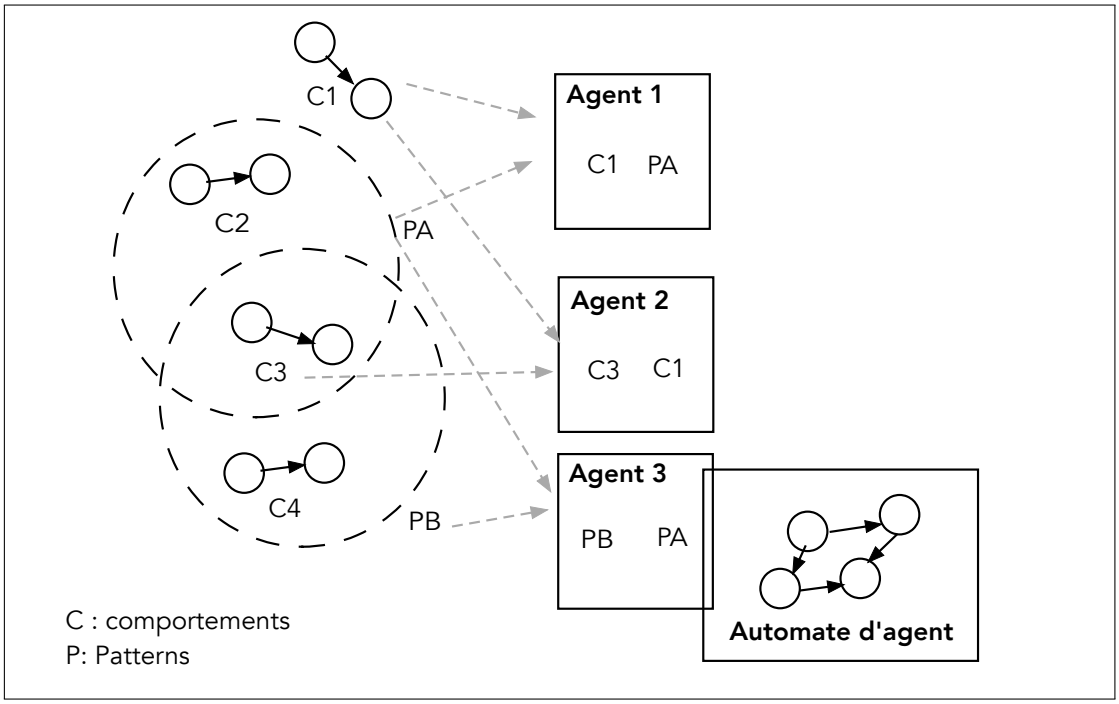


FIGURE 3.1 – Principe de création des comportements, patterns et agents

Le scénario global, quant à lui, va être produit par agrégation des agents dans des contextes d'exécution. Le contexte courant évolue à l'occurrence spécifique d'un évènement donné. Ainsi, chaque contexte résultant de la composition d'un ensemble d'agents va présenter un comportement issu du résultat de la composition du comportement des agents. L'automate d'exécution du contexte est lui aussi produit automatiquement par produit synchronisé. Le principe de création est décrit figure 3.2.

Cette approche simplifie la phase de scénarisation puisqu'il peut suffire, dans certains cas, d'ajouter un agent dans un contexte et de recalculer son automate.

3.1.2 Des Contenus externalisés

L'approche modulaire que nous venons de décrire s'appuie sur une dynamique d'externalisation des contenus permettant de découpler les données du modèle d'exécution. En effet, dans une optique de généricité et d'extensibilité, le modèle repose sur une identification des contenus et des modules exécutables décrits dans une base de données externe.

L'approche proposée consiste à stocker les textes et la référence vers le code d'exécution des comportements de manière à permettre une modification de ces derniers sans modifier la logique du modèle.

Prenons l'exemple du comportement *parler* dont l'automate est présenté figure 3.3. Nous associons à ce comportement le module exécutable portant l'identifiant 1, repré-

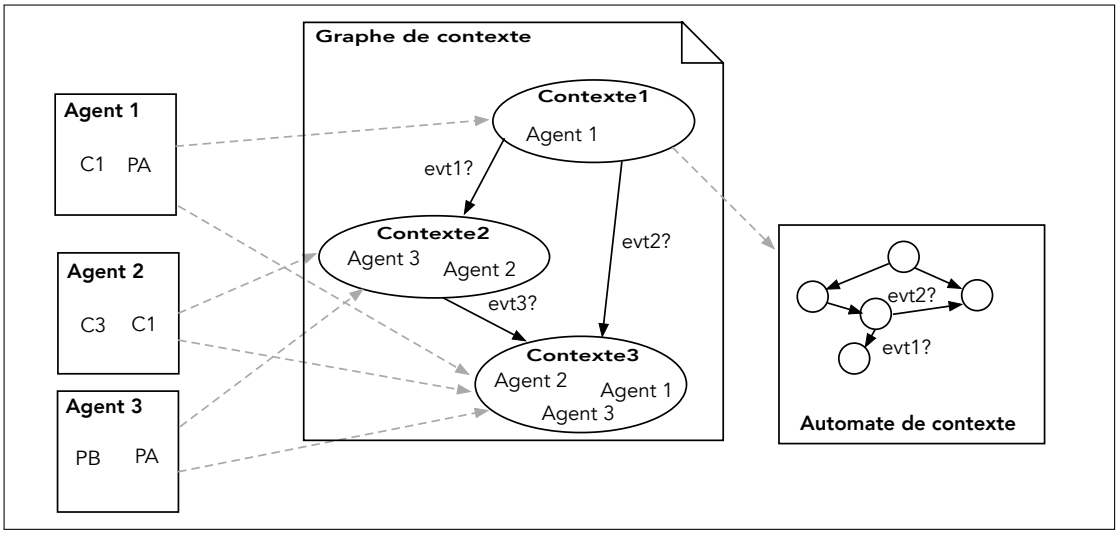


FIGURE 3.2 – Principe de création des contextes et du graphe de contexte

sentant le module *say*. Ce module nécessite un paramètre d'entrée, représentant le texte à synthétiser vocalement, dont l'identifiant en base de données est 1.

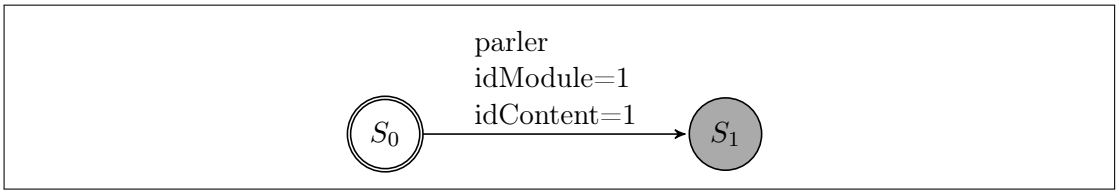


FIGURE 3.3 – Représentation du comportement *parler*

Content		
Id.	**Label**	**Nom**
1	monstrueux_question_1	Dans les caisses en bois, trouve un grand objet qui aurait pu appartenir à une licorne. En réalité à quel animal appartient-il ?

TABLE 3.1 – Extrait de la base de données des contenus

Ainsi, comme nous allons le mettre en évidence dans la suite de ce document, cette externalisation des contenus facilite grandement le processus de conception et de modélisation du scénario. La modification est alors réalisée directement en base de données.

Au delà de la facilité de modification des contenus pour la reproductibilité de l'expérience, cette approche ouvre des perspectives pour une modification dynamique des

Behavior		Parameter		Output	
Id	**Name**	**Label**	**Type**	**Label**	**Type**
1	say	sentence	text	out	boolean

TABLE 3.2 – Extrait de la base de données des modules

contenus en cours d'exécution (textes simplifiés dans le cas de jeunes utilisateurs par exemple).

3.1.3 La prise en compte des contraintes de Temps

La prise en compte de la dimension temporelle s'avère primordiale dans la conception d'une activité interactive. En effet, gérer les temporisations entre les comportements du système et dans les interactions avec l'utilisateur permet de garantir la nécessaire fluidité des échanges. Ainsi, les enjeux de l'intégration des contraintes temporelles portent sur :

1. La description des temps acceptables pour la réalisation des comportements ou les attentes au cours de l'interaction ;
2. La prise en compte d'un temps global d'exécution du système et le temps global propre au temps d'échange d'un utilisateur avec ce système.

Décrire et analyser le temps est une tâche complexe nécessitant une sémantique précise (le temps est-il discret, continu ? s'incrémente t-il ou se décrémente t-il ? comment s'interprètent les contraintes ?). Tel que nous l'avons conçu, le modèle se prête parfaitement à l'intégration d'un temps décrit par la sémantique de [GLPY97]. Il s'agit d'un temps continu, représenté par des horloges, associées à des contraintes sur les transitions (gardes) et sur les états (invariants sur les localités).

Pour répondre aux enjeux liés aux temps cités précédemment, nous avons intégré ces horloges et contraintes :

- sur les automates de comportement ;
- sur le graphe de contexte.

Ainsi, le temps présent ici est double. Un temps d'exécution du contexte et un temps global du système. L'analyse plus fine des transitions et des horloges de ces deux temps permet d'extraire des conclusions sur la temporalité liée à l'utilisateur.

3.2 Une modélisation à deux couches

La phase de modélisation est divisée en deux étapes distinctes et correspondent aux deux couches présentées en figure 3.4 :

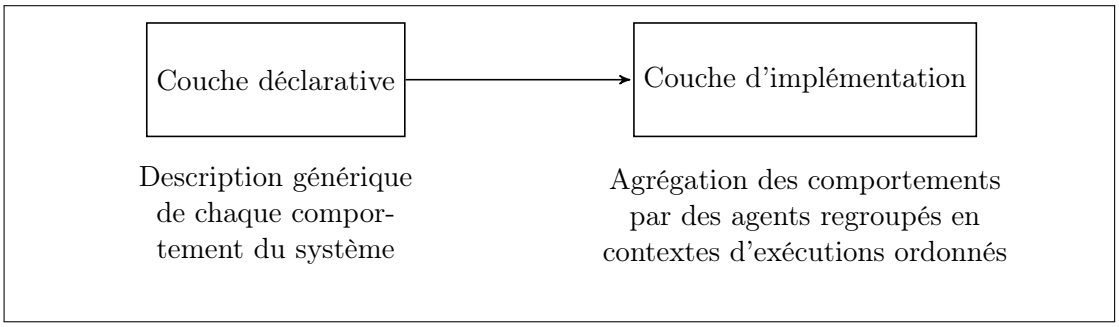


FIGURE 3.4 – CIT : Modèle à deux couches

- la **couche déclarative** permet de définir les comportements génériques du scénario regroupables en patterns ;
- la **couche d'implémentation** permet d'instancier les comportements et patterns par des agents, regroupées eux mêmes dans des contextes d'exécution ordonnés, modélisant ainsi le graphe de contextes.

Chacune de ces entités est modélisée par un automate temporisé à entrées/sorties dont nous avons donné la définition dans l'état de l'art (chapitre 2, définition 2.2.3).

3.2.1 Couche déclarative

Le défi majeur de la couche déclarative est de permettre la représentation de comportements et de modèles atomiques réutilisables. Ces modèles peuvent être composés dynamiquement au sein des agents de la couche d'implémentation pour constituer leur automate de comportement dans un contexte d'exécution.

Elle permet la définition des comportements génériques que nous souhaitons intégrer à l'expérience. Ces comportements peuvent ensuite être regroupés en patterns. Les patterns représentent une nouvelle structure dans laquelle sont regroupés les comportements susceptibles d'être exécutés à plusieurs reprises dans le scénario, les uns à la suite des autres. Les comportements et patterns sont réutilisables par héritage multiple dans la seconde couche.

Définition 3.2.1. La **couche déclarative** est définie par le tuple $\langle V_d, \Lambda, C, P, X \rangle$ tel que :

- V_d : ensemble de *variables déclaratives* utilisables lors de la définition des comportements C. Une variable déclarative peut être une variable entière (*int*), une

variable décimale (*double*), une variable booléenne (*bool*) ou une chaîne de caractères (*string*). Nous notons V_d^e la valuation des variables déclaratives, également appelée vecteur d'état ;

- Λ : ensemble de *signaux* permettant la synchronisation de plusieurs comportements ;
- C : ensemble de *comportements* génériques. Nous notons Σ, l'ensemble des labels identifiant les comportements ;
- P : ensemble de *patterns*. Un pattern regroupe un sous ensemble de comportements issus de C ;
- X : ensemble d'*horloges* définies sur $\mathbb{R}^+$. Les horloges peuvent être utilisées dans la définition même d'un comportement ou de façon plus globale, afin de temporiser l'ensemble du scénario.

Afin d'illustrer cette définition, nous allons détailler la création de ces différentes entités grâce à un exemple fil rouge tout au long de ce chapitre. Nous souhaitons modéliser trois comportements pour le pilotage d'un robot humanoïde :

- un comportement *seLever* pour mettre le robot sur ses jambes ;
- un comportement *marcher* pour que le robot puisse se déplacer ;
- un comportement *sasseoir* pour mettre le robot dans sa position de repos.

Les comportements *seLever* et *marcher* seront associés dans un pattern *promenade*. Ce couple de comportements pourra être utilisé par héritage multiple dans la seconde couche.

3.2.1.1 Comportement

Le paradigme de construction de notre modèle est basé sur la composition des comportements atomiques. Ces comportements représentent des actions entraînant la progression des entités du système provoquant une modification du vecteur d'état V_d^e. Chaque comportement peut faire l'objet de contraintes temporelles. Les comportements étant modélisés par des automates temporisés, la sémantique utilisée est celle décrite dans le chapitre précédent (section 2.2.2). Ces contraintes temporelles auront pour conséquence l'exécution du comportement entre n et m unités de temps.

Définition 3.2.2. Un **comportement** $c \in C$ est défini par le tuple $\langle \sigma_c, g_c(V_d), u_c(V_d), i_c(X), g_c(X), r_c(X), l_c, \lambda_c, f, A_c \rangle$ tel que :

- $\sigma_c \in \Sigma$: label identifiant le comportement ;
- $g_c(V_d)$: garde appliquée sur les variables déclaratives. L'exécution du comportement est conditionné par la satisfaction de cette garde ;

- $u_c(V_d)$: fonction de valuation des variables déclaratives V_d. L'exécution du comportement provoque la modification du vecteur d'état V_d^e ;
- $i_c(X)$: invariant des horloges X indiquant la durée maximale avant laquelle le comportement doit être exécuté ;
- $g_c(X)$: garde sur les horloges X indiquant la durée minimale après laquelle le comportement peut être exécuté ;
- $r_c(X)$: fonction de remise à zéro des horloges X ;
- l_c est un booléen indiquant l'aspect répétitif ou non du comportement. Une valeur à *true* modélisera ce comportement par un automate réflexif (une seule localité et une transition ayant pour source et destination cette même localité) ;
- $\lambda_c \in \Lambda$:signal permettant la synchronisation avec un autre comportement. $\lambda!$ est un signal d'émission et $\lambda?$, un signal de réception selon la sémantique UPPAAL [GLPY97] ;
- f : fonction de transition : $f : s \times \sigma_c \times g_c \times u_c(V_d) \times r_c(X) \times \lambda_c \rightarrow s'$, tel que s et s' sont respectivement les localités source et destination de l'automate temporisé A_c du comportement et g_c est la garde composée de $g_c(X)$ et de $g_c(V_d)$.

3.2.1.2 Automate temporisé du comportement

L'automate temporisé d'un comportement est composé de :

- de deux localités S_0 et S_1, qui peuvent être la même dans le cas d'un comportement répétitif (si $l_c = true$) ;
- d'un invariant $iS_0(X)$ appliquée à la localité source S_0 sur les horloges X ;
- d'une transition t définie par $f : S_0 \times \sigma \times g \times u(V_d) \times r(X) \times \lambda \rightarrow S_1$:
  - σ : le label du comportement ;
  - $g : g(V_d) \wedge g(X)$, respectivement la garde appliquée sur les variables déclaratives et sur les horloges X ;
  - $u(V_d)$: la fonction de mise à jour des variables déclaratives ;
  - $r(X)$: fonction de remise à zéro d'un sous ensemble des horloges X ;
  - λ : un signal d'émission ou de réception permettant la synchronisation de deux comportements.

Exemple : Nous souhaitons modéliser le comportement *seLever*, qui permettra à terme au robot Nao de se mettre debout (la spécification de l'action exécutée sur le robot se fera dans la seconde couche) et le comportement *marcher*.

V_d = [estDebout : *bool*, value : *false*]
$X = [x]$
c_1=[
$\sigma_{c_1} = seLever$
$g_{c_1}(V_d) = [estDebout == \text{false}]$
$u_{c_1}(V_d) = \{estDebout = \text{true}\}$
$i_{c_1}(X) = [\text{x}<5]$
$g_{c_1}(X) = [\text{x}\geq 2]$
$r_{c_1}(X) = \{\text{x} := 0\}$
$l_{c_1} = false$
$\lambda_{c_1} = \varnothing$
]

c_2=[
$\sigma_{c_2} = marcher$
$g_{c_2}(V_d) = [estDebout == \text{true}]$
$u_{c_2}(V_d) = \varnothing$
$i_{c_2}(X) = [\text{x}\leq 10]$
$g_{c_2}(X) = [\text{x}\geq 3]$
$r_{c_2}(X) = \{\text{x} := 0\}$
$l_{c_2} = true$
$\lambda_{c_2} = \text{chante}!$
]

Nous imposons une contrainte de temps au comportement *seLever* : son exécution doit se produire dans l'intervalle [2 ;5[unités de temps. Aucune synchronisation n'aura lieu à l'issu de son exécution. L'horloge x sera, quant à elle, remise à zéro.
Le comportement *marcher* devra être exécuté dans l'intervalle [3 ;10] unités de temps et pourra être répétitif.

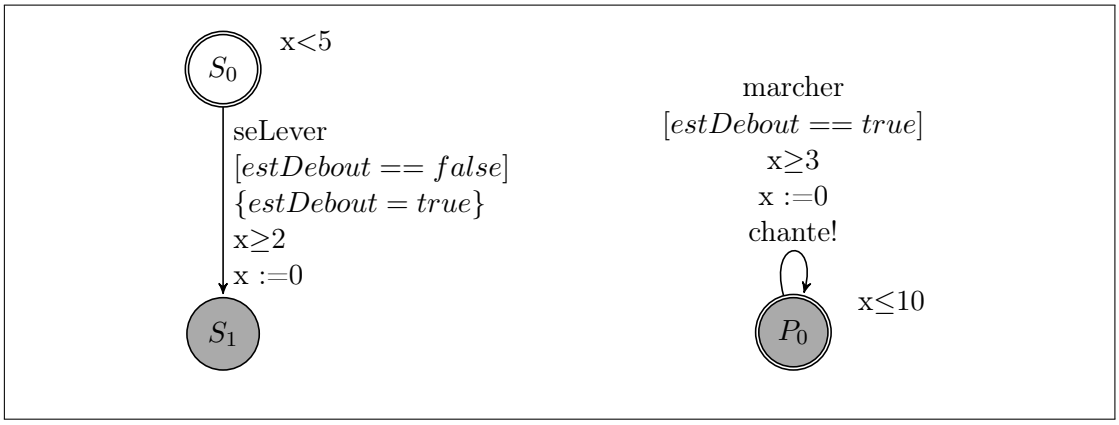


FIGURE 3.5 – Automate temporisé des comportements *seLever* et *marcher*

Les automates temporisés correspondants à ces comportements sont donnés dans la figure 3.5.

La représentation du comportement *seLever* est réalisée par un automate temporisé à deux localités. La borne maximale de l'intervalle de temps est représentée au travers d'un invariant appliqué sur la localité initiale. La borne inférieure est, quant à elle, représentée par une garde de transition. La garde appliquée sur la variable *estDebout* est schématisée par des crochets []. La fonction de valuation des variables est, pour sa part, représentée par des accolades {}, tandis que la remise à zéro des horloges est représentée par :=. Enfin, le comportement *marcher*, dont la représentation par un automate temporisé réflexif ($l_{c_2} = true$) est donnée en figure 3.5, sera synchronisé avec un autre comportement via

le signal *chante*. Les représentations des contraintes de temps, de variables, la fonction de valuation et le reset d'horloge restent inchangées.

L'algorithme permettant la conversion des comportements en automate est donné par l'algorithme 1.

Données : Comportement comportement
Résultat : AutomateTemporisé automate

automate ← nouveau AutomateTemporisé;
automate ← ajouter localité initiale S_0;

si *comportement est répétitif* **alors**
  automate ← faire de S_0 une localité finale;
  automate ← ajouter transition t de S_0 à S_0;
sinon
  automate ← ajouter la localité finale S_1;
  automate ← ajouter transition t de S_0 à S_1 ;
fin
$i_{S_0}(X) \leftarrow i(x)$;
$a_t \leftarrow \sigma_c$;
$g_t \leftarrow g_c(V_d) \cup g_c(X)$;
$r_t \leftarrow r_c(X)$;
$u_t \leftarrow u_c(V_d)$;
$\lambda_t \leftarrow \lambda_c$;

Algorithme 1 : Génération de l'automate temporisé d'un comportement

3.2.1.3 Pattern

Les patterns regroupent un ensemble de comportements susceptibles d'être exécutés conjointement à plusieurs reprises au cours du scénario. Les patterns sont des structures non interruptibles. Aucun comportement n'appartenant pas au pattern pourra être exécuté avant la fin de son exécution. Ces patterns seront ensuite instanciables dans la seconde couche par les agents. Nous utilisons ici le principe de réutilisation de composants.

Définition 3.2.3. Un **pattern** $p \in P$ est défini par le tuple $\langle C_p, A_p \rangle$ tel que :

- $C_p \subset C$: ensemble de comportements regroupés au sein du pattern. Nous ajoutons un paramètre à chaque comportement d'un pattern représentant la priorité d'exécution ;
- A_p : l'automate du pattern. Il est obtenu par le produit synchronisé de l'ensemble des automates des comportements qui le composent. Nous donnons sa définition

dans l'algorithme 2. A l'issu du produit synchronisé, une vérification du modèle est effectuée afin de supprimer les transitions non accessibles en fonction des contraintes sur les variables.

Données : Pattern pattern
Résultat : AutomateTemporisé automate

automate ← nouveau AutomateTemporisé;
automatePriorite ← nouveau AutomateTemporisé;
comportementsPriorises ← pattern.getComportmentsPriorises();
localiteCourante ← null;
localitePrioritaireFrom ← nouveau dictionnaire<int, etat>;
localitePrioritaireTo ← nouveau dictionnaire<int, etat>;
i ← 1;
pour tous *comportement c de comportementsPriorises ayant la priorité i* **faire**
  automateComportement ← toAutomaton(c);
  si *automatePriorite est vide* **alors**
    automatePriorise ← automateComportement;
    localiteCourante ← automatePriorise.localiteFinale();
    localiteInitiale ← automatePriorise.localiteInitiale();
  sinon
    localiteInitiale ← localitePrioritaireFrom.get(i);
    localiteFinale ← localitePrioritaireTo.get(i);
    si *localiteInitiale == null* **alors**
      localiteInitiale ← localiteCourante;
    fin
    localiteInitiale ← localiteInitiale.getInvariant() && c.getInvariant();
    si *localiteFinale == null* **alors**
      localiteFinale ← nouvelle localite finale;
    fin
    tr ← automateComportement.getTransition();
    Ajouter une nouvelle transition t à automatePriorite de localiteInitiale à localiteFinale avec l'ensemble des propriétés de tr;
  fin
  Ajouter à localitePrioritaireFrom le couple <i, localiteInitiale>;
  Ajouter à localitePrioritaireTo le couple <i, localiteCourante>;
  i ← i+1;
fin

```
pour tous comportement c du pattern à l'exception des comportements priorisés
 faire
  | automateComportement ← toAutomaton(c);
  | automate ← produitSynchronise(automate, automateComportement);
fin
si automate est vide alors
  | automate ← automatePriorise;
sinon
  | si automatePriorise n'est pas vide alors
  |   | Faire de localiteCourante une localite non finale;
  |   | pour tous transition tr de automate faire
  |   |   | si tr.localiteSource est la localite initiale alors
  |   |   |   | tr.localiteSource ← localiteCourante;
  |   |   | sinon
  |   |   |   | Ajouter la localite tr.localiteSource à automatePriorise;
  |   |   | fin
  |   |   | Ajouter la localite tr.localiteDestination à automatePriorise;
  |   |   | Ajouter la transtion tr à automatePriorise;
  |   | fin
  |   | automate ← automatePriorise;
  | fin
fin
```

Algorithme 2 : Génération de l'automate temporisé d'un pattern

Exemple : Nous souhaitons définir le pattern *promenade*, regroupant les comportements *seLever* et *marcher*, définis précédemment. L'automate issu du produit synchronisé, présentée dans la section suivante, de leur automate respectif est donné dans la figure 3.6.

3.2.1.4 Produit synchronisé de deux automates

Le produit synchronisé offre une approche hybride qui permet au système d'évoluer par la synchronisation de deux entités ou par leur évolution individuelle. Notre définition s'appuie sur les travaux de Bengtsson [BY03] et de Rempulski [Rem13].

Soit, A_1 et A_2, deux automates temporisés. L'automate A résultant du produit synchronisé de A_1 et A_2 est composé de :

- $S_A = S_{A_1} \times S_{A_2}$;
- $S_{0_A} = S_{0_{A_1}} \times S_{0_{A_2}}$;
- $T_A = T_{A_1} \cup T_{A_2}$;

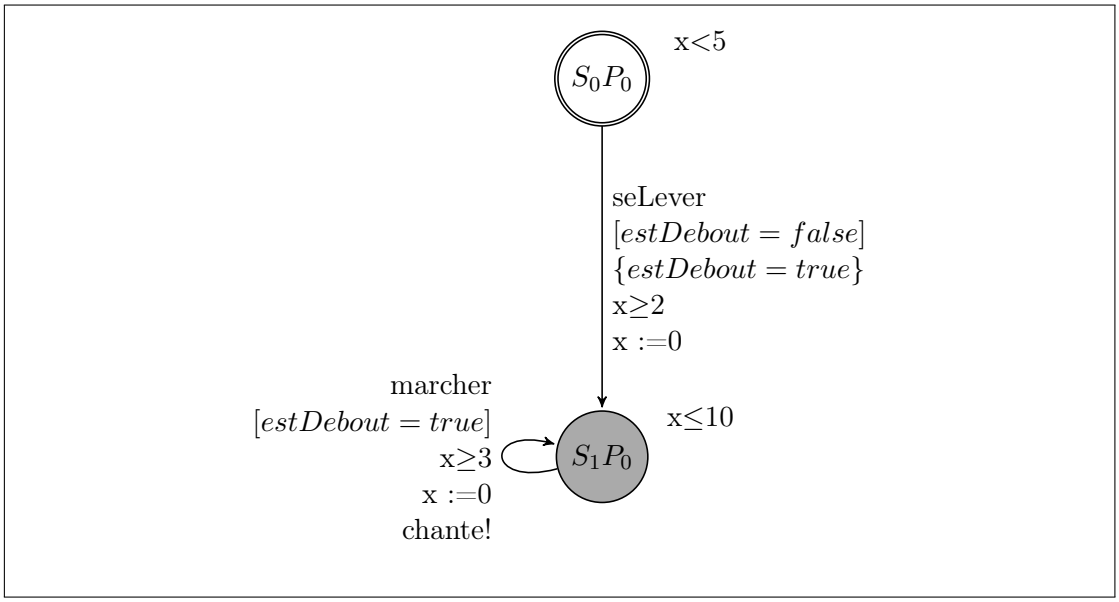


FIGURE 3.6 – Automate temporisé du pattern *promenade*

- $V_A = V_{A_1} \cup V_{A_2}$;
- $I_A = I_{A_1} \cup I_{A_2}$;
- $X_A = X_{A_1} \cup X_{A_2}$.

Nous notons $s_{A_1} \in S_{A_1}$ et $s_{A_2} \in S_{A_2}$ deux localités issues respectivement de l'automate A_1 et de l'automate A_2, $\lambda \in \Lambda$, un signal de synchronisation et a_t l'étiquette d'une transition $t \in T_A$. La fonction de transition synchronisée est :

- $f((s_{A_1}, s_{A_2}), a_t) = (s'_{A_1}, s'_{A_2})$ si $f_{A_1}(s_{A_1}, a_t!) = s'_{A_1}$ et $f_{A_2}(s_{A_2}, a_t?) = s'_{A_2}$;
- $f((s_{A_1}, s_{A_2}), a_t) = (s'_{A_1}, s_{A_2})$ si $f_{A_1}(s_{A_1}, a_t) = s'_{A_1}$ avec $a_t \in \Sigma_{A_1}$;
- $f((s_{A_1}, s_{A_2}), a_t) = (s_{A_1}, s'_{A_2})$ si $f_{A_2}(s_{A_2}, a_t) = s'_{A_2}$ avec $a_t \in \Sigma_{A_2}$.

3.2.2 Couche d'implémentation

La couche d'implémentation est la couche où le concepteur définit les agents. Les agents sont des entités permettant l'instanciation de comportements atomiques ou de patterns, précédemment définis dans la couche déclarative, par héritage multiple. Ce principe d'héritage multiple permet au concepteur de réutiliser les comportements prédéfinis dans la couche déclarative autant de fois que nécessaire sans avoir à les redéfinir. Chaque comportement de l'agent doit être spécifié. Cette spécification consiste à déterminer l'action réelle qui sera exécutée sur le processus piloté. Le concepteur utilise pour cela la base de données de contenus dans laquelle est stocké l'ensemble des contenus et

actions exécutables sur le processus (par exemple sur le robot Nao). Ces agents sont ensuite liés dans des contextes d'exécution représentant des situations dans lesquelles les agents interagissent les uns avec les autres.

Définition 3.2.4. La **couche d'implémentation** est définie par le tuple $\langle V_G, A, E, G \rangle$ tel que :

- V_G : un ensemble de *variables globales*, utilisables par toutes les entités de la couche d'implémentation, notamment lors de la spécification des actions exécutées sur le processus ;
- A : un ensemble d'*agents* implémentant des patterns et des comportements atomiques définis dans la couche déclarative ;
- E : un ensemble de *contextes* représentant des situations d'exécution particulière du système. Un contexte met en relation un ou plusieurs agents ;
- G : le *graphe de contexte* du système. Il représente, sous forme d'automate temporisé à entrées/sorties, l'ordre d'exécution des différents contextes créés.

3.2.2.1 Agent

Un agent peut implémenter le rôle d'un ou plusieurs patterns grâce au mécanisme d'héritage multiple. Il permet aussi d'implémenter des comportements atomiques définis dans la couche déclarative. Ainsi, le concepteur peut spécialiser un agent grâce aux différents comportements qu'il implémente. L'automate de comportement de l'agent sera obtenu par synchronisation de l'ensemble des comportements. L'enjeu majeur de cette couche est de garantir la réutilisabilité des comportements. Un agent spécifique à une expérience interactive donnée peut utiliser des comportements atomiques conçus pour une autre expérience.

Les comportements, agrégés par un agent, proposent des valeurs par défaut lors de leur définition dans la couche déclarative. Ceux-ci peuvent ne pas correspondre à l'attente du concepteur pour un agent particulier. Il peut alors en modifier les valeurs. Enfin, l'agent permet de spécifier, pour chacun des comportements qu'il agrège, le contenu réel qui sera exécuté sur le processus piloté, dans notre cas un robot Nao.

Définition 3.2.5. Un agent a est défini par le tuple $\langle V_a, P_a, B_a, A_a \rangle$ tel que :

- V_a : un ensemble de *variables d'agents* ;
- P_a : un ensemble de patterns implémentés par l'agent ;
- B_a : un ensemble de comportements atomiques implémentés par l'agent ;
- A_a : l'automate temporisé de l'agent.

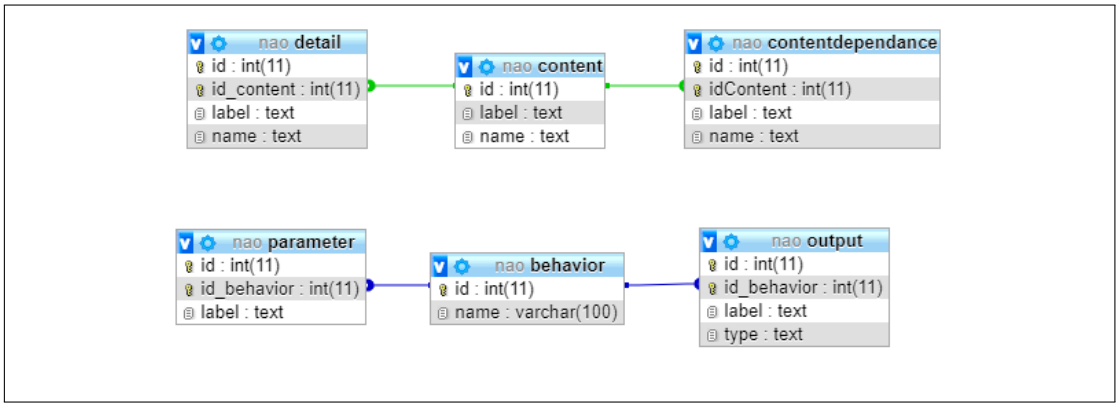


FIGURE 3.7 – Base de données des contenus

Spécification des comportements La spécification des comportements se réalise par l'utilisation d'une base de données externe (figure 3.7) intégrant l'ensemble des actions exécutables sur le processus piloté ainsi que les divers contenus, notamment textuel.

Exemple : Nous définissons un agent *promeneur* implémentant le pattern *promenade* et le comportement atomique *asseoir*, qui déclenchera le signal *assis*. Le concepteur doit donc spécifier les trois comportements que l'agent implémente. Pour cela, il utilise la base de données et en particulier la table *Behavior* contenant l'ensemble des actions exécutables sur le processus. Ainsi, pour chaque comportement, il va spécifier l'action à exécuter. Pour être exécutée, certaines actions possèdent des paramètres (un dictionnaire et un temps d'écoute pour la reconnaissance vocale par exemple) et des sorties (le mot reconnu et un taux de confiance pour la reconnaissance vocale). Ces paramètres doivent également être spécifiés par le concepteur. Enfin, si le concepteur veut utiliser des contenus, par exemple textuels lors de l'utilisation de la synthèse vocale, le concepteur utilisera la table *Content*.

L'automate temporisé de cet agent, obtenu par produit synchronisé, est donné dans la figure 3.8.

3.2.2.2 Contexte

Le contexte décrit une situation dans laquelle un certain nombre d'agents va être impliqué. Sa description se fait simplement par la liste des agents présents et les comportements possibles au sein du contexte seront calculés par synchronisation de l'ensemble des agents présents.

Définition 3.2.6. Un contexte c est défini par le tuple $\langle B_c, A_c \rangle$ tel que :

- B_c : un ensemble d'agents impliqués dans le contexte c ;
- A_c : l'automate temporisé du contexte.

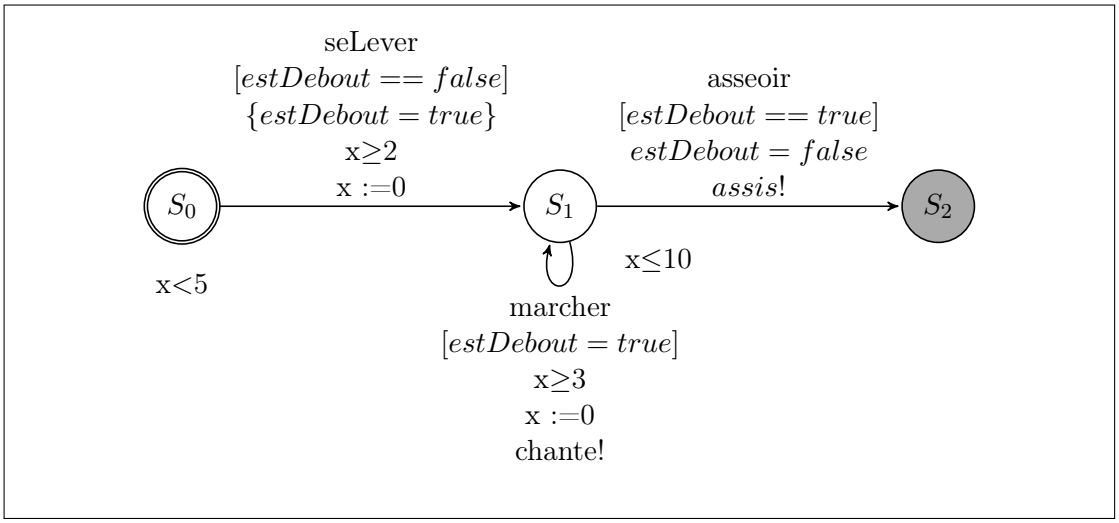


FIGURE 3.8 – Automate temporisé de l'agent *promeneur*

Reprenons notre exemple précédent. Nous souhaitons agréger dans le contexte *balade* les agents *promeneur* et *chanteur*. Ce dernier agent ne possède qu'un seul comportement qui est celui de *chanter* lorsqu'une synchronisation sur le signal *chante* se produit. L'automate de ce contexte est alors le résultat du produit synchronisé des automates des deux agents. Le résultat de ce produit est donné en figure 3.9. Les comportements *marcher* et *chanter* se sont alors synchronisés pour ne former au final qu'une seule transition exécutant deux comportements.

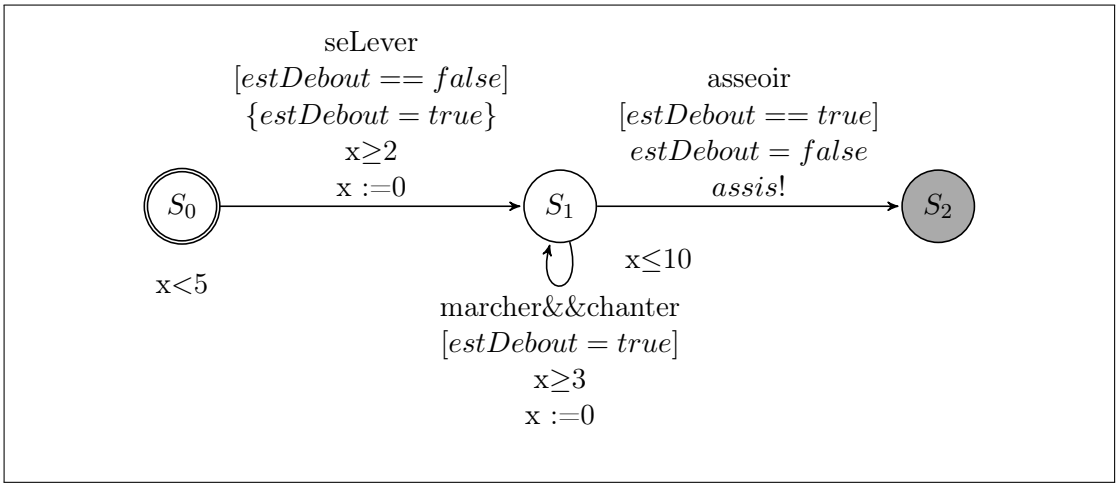


FIGURE 3.9 – Automate temporisé du contexte *balade*

3.2.2.3 Graphe de contexte

Le scénario global d'exécution est produit par le graphe de contexte. Il s'agit d'un automate temporisé à entrées/sorties de haut niveau qui représente les passages entre contextes d'exécution. L'évolution du système d'un contexte à un autre peut se réaliser par synchronisation (envoie d'un signal depuis un contexte, reçu par le graphe de contexte) ou par contrainte temporelle. Le graphe de contexte est défini par un ensemble de contextes, dont l'un est initial, auxquels sont ajoutés des successeurs. Ainsi, pour chaque successeur, le concepteur doit spécifier :

- le nom du contexte successeur ;
- le signal de synchronisation ;
- la mise à jour des variables.

Exemple : Nous souhaitons ordonner les contextes *balade, seReposer, accueil* et *jeu*. Le contexte *balade* implémente l'agent *promeneur*. Nous souhaitons que le contexte *balade* débute le scénario et qu'il lance le contexte *seReposer* lorsque le robot se sera assis, par synchronisation, via le signal *assis*. Le retour au contexte *balade* se fera également par synchronisation, via le signal *bouge*. Le passage au contexte accueil et au contexte jeu se fera par l'utilisation des canaux *accueillir* et *jouer* respectivement. Le graphe de contexte résultant est donné sur la figure 3.10.

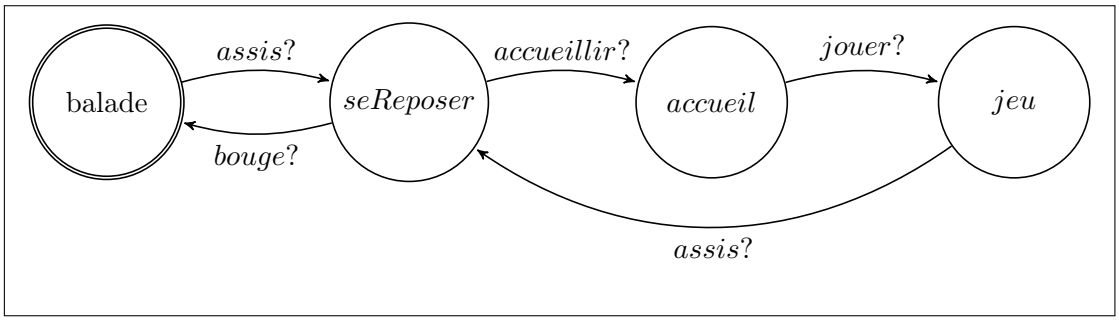


FIGURE 3.10 – Graphe de contexte

Après avoir conçu son expérience interactive, le concepteur peut vérifier que le modèle produit répond à ses attentes. Cette étape de vérification consiste à réaliser un *model-checking*, que nous allons présenter dans le chapitre suivant.

3.3 Conclusion

Dans ce chapitre, nous avons présenté notre modèle générique **CIT** (**C**ontent, **I**nteraction and **T**ime), dédié à la modélisation de scénarios interactifs. La création d'un tel scénario, modélisé par des automates temporisés à entrées/sorties, est facilité par son découpage en deux couches.

Dans la première couche, le concepteur crée **la couche déclarative**. Dans celle-ci sont définis les comportements atomiques du scénario. Ces comportements peuvent être regroupés en patterns, structures génériques réutilisables et non interruptibles.

La seconde étape consiste à créer **la couche d'implémentation** dans laquelle vont être implémentés les patterns et comportements atomiques dans des agents, par héritage multiple. Lors de la définition d'un agent, le concepteur de l'expérience interactive spécifie pour chaque comportement de l'agent, quelle action et contenu vont être exécutés sur le processus piloté. Nous utilisons ici, une base de données externe facilitant la gestion des contenus. Ainsi, un scénario dédié au Muséum de La Rochelle pourra être déployé dans un autre musée, en ne modifiant seulement que les contenus présents en base de données. Les agents ainsi créés sont ensuite mis en relation au sein de contextes ordonnés, modélisant ainsi les différentes situations exécutables du scénario.

Les dimensions CIT ont été implémentés dans notre outil d'édition **CELTIC** que nous présenterons au chapitre 6. La dimension permettant de gérer l'espace fait partie des intégrations futures que nous souhaitons apporter à notre modèle.

Le scénario obtenu avec notre modèle générique va ensuite permettre le pilotage de processus quelconques. Le chapitre suivant est donc dédié à la présentation de la sémantique dynamique du modèle et à notre technique de *model checking*.

Chapitre 4

Sémantique dynamique et vérification du modèle CIT

Sommaire

Nous avons présenté dans le chapitre précédent, notre modèle CIT pour la modélisation des comportements robotiques. Nous avons décrit, pour cela, la sémantique statique du modèle au travers des définitions formelles nécessaires. Ce modèle temporisé étant amené à être utilisé pour la supervision d'une tâche robotique, nous devons décrire sa sémantique dynamique pour présenter son comportement à l'exécution et en particulier sur l'application des contraintes temporelles. A partir de la sémantique statique et dynamique, le système peut ensuite être analysé et contrôlé afin de vérifier certaines propriétés sur la qualité de l'exécution. Nous détaillerons donc, dans un second temps, l'extension de la méthode de vérification pour le modèle CIT.

4.1 Sémantique dynamique du modèle

La sémantique dynamique d'un modèle permet de décrire le comportement du système au cours de son exécution. S'il existe plusieurs types d'automates modélisant les systèmes temps-réel et ayant une sémantique dynamique propre, nous nous intéressons dans ces travaux, à une modélisation de l'activité robotique à base de réseaux d'automates temporisés à entrées/sorties, dans lesquelles les horloges évoluent de façon synchrone (le temps s'écoule au même rythme pour l'ensemble des horloges).

4.1.1 Sémantique dynamique des automates temporisés

Soit A, un automate temporisé dont la définition a été décrite en section 2.2.2, définition 2.2.3. Nous associons à chaque localité $l \in L$ une valuation des horloges X notée v telle que $v(X)$ satisfait l'invariant $i_l(X)$. Il existe deux types de transitions : les transitions de délais et les transitions discrètes.

On parle de **transition de délai** lorsque le temps s'écoule d'une durée $d \in \mathbf{R}$. La valeur des différentes horloges est alors incrémentée. La valuation (l, v) devient alors $(l, v + d)$. Nous avons donc :

$$(l, v) \xrightarrow{d} (l, v + d) \text{ si pour tout } 0 \leq d' \leq d,\ v + d' \text{ satisfait } i_l(X).$$

On parle de **transition discrète** lorsqu'une transition $t : \langle l, a_t, g_t, r_t, u_t, l' \rangle$ du modèle est exécutée, si les contraintes sur les variables et les horloges sont respectées (soit g_t composées de $g_t(V)$, garde appliquée sur les variables V et $g_t(X)$, garde appliquée sur les horloges X). Nous avons donc :

$$(l, v) \xrightarrow{a_t} (l', v[r_t := 0])$$

avec r_t l'éventuelle ré-initialisation des horloges.

4.1.2 Sémantique dynamique du modèle CIT

Sur la base de la sémantique dynamique des automates temporisés, nous avons défini la sémantique dynamique du modèle CIT intégrant le graphe de contexte et les automates de contexte générés.

Soit A_C l' automate temporisé du contexte C et A_{gc} l'automate temporisé du graphe de contexte gc. Nous associons à chaque localité $l \in A_C$ l'identifiant c du contexte C ainsi que la valuation v des horloges X. Il existe donc, pour le modèle CIT, trois types de transitions :

1 : les transitions de délais pour lesquelles le temps s'écoule d'une durée $d \in \mathbf{R}$ à la fois dans l'automate du contexte et également dans l'automate du graphe de contexte. Nous avons donc :

$$(l, c, v) \xrightarrow{d} (l, c, v + d) \text{ si pour tout } 0 \leq d' \leq d,\ v + d' \text{ satisfait } i_l(X) \text{ et } i_c(X)$$

avec l une localité de l'automate du contexte c.

2. les transitions discrètes lorsqu'une transition $t \in A_C : \langle l, a_t, g_t, r_t, u_t, \lambda_t, l' \rangle$ est exécutée avec $\lambda_t = \emptyset$ (sans synchronisation). Nous avons donc :

$$(c, l, v) \xrightarrow{a_t} (c, l', v[r_t := 0])$$

3. les transitions de synchronisation lorsqu'une transition $t \in A_C : \langle l, a_t, g_t, r_t, u_t, \lambda_t, l' \rangle$ est exécutée avec $\lambda_t! = \emptyset$ (synchronisation avec le graphe de contexte). Nous avons donc :

$$(c, l, v) \xrightarrow{\lambda_t} (c', l_0, v[r_t := 0])$$

avec l_0 la localité initiale du contexte c'.

Reprenons l'exemple du contexte *balade* introduit en section 3.2.2.2 dont l'automate temporisé et une exécution possible de celui-ci, sont donnés en figure 4.1. Dans cet exemple :

- une transition de délai peut être :

$$((balade, S_0), x = 0) \xrightarrow{d} (balade, S_0), x = 2.7)$$

- une transition discrète :

$$((balade, S_1), x = 7.63) \xrightarrow{marcher\&\&chanter} (balade, S_1), x = 0)$$

- une transition de synchronisation peut être :

$$((balade, S_1), x = 0) \xrightarrow{\lambda_t = assis} (seReposer, S_0), x = 9.3)$$

Nous indiquons sur cette figure le couple (contexte courant/localité courante). Conformément au graphe de contexte décrit en section 3.2.2.3, l'envoie du signal $assis!$ provoque un changement de contexte. Le système atteint alors la localité initiale S_0 du contexte *seReposer*. Cette communication entre contextes est réalisée par la synchronisation du signal $assis$.

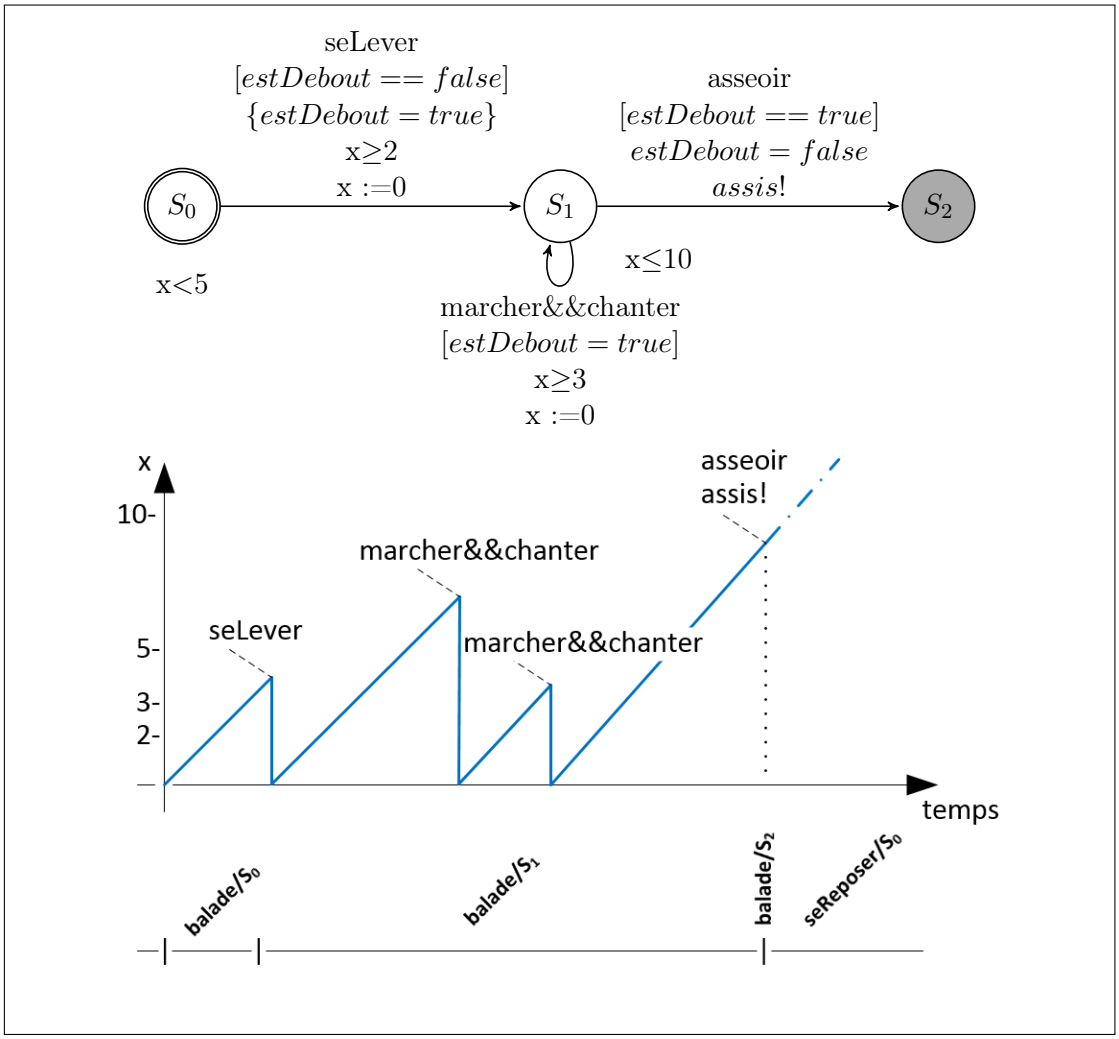


FIGURE 4.1 – Automate temporisé du contexte *Promenade* et un exemple d'exécution

4.2 Vérification formelle des automates temporisés

Lors de la conception d'un jeu ou d'une plateforme robotique, les concepteurs doivent pouvoir vérifier que le modèle proposé répond à leurs attentes en vérifiant certaines propriétés. Par exemple, dans le cas des jeux, il est intéressant de pouvoir vérifier qu'un joueur n'est pas en mesure d'accéder au niveau suivant sans avoir réalisé certaines tâches. Dans le cas de la robotique, de plus en plus orienté vers le grand public, un concepteur va pouvoir s'assurer par exemple que la vitesse d'un robot ne peut dépasser un certain seuil, pour des raisons évidentes de sécurité.

Comme présenté dans [LFD+18], il existe différentes approches pour la formalisation et la vérification dans le contexte robotique. Ainsi, les différents formalismes cités sont les machines à états finis, les algèbres de processus, les ontologies. Cet état de l'art met

en évidence un certain nombre de techniques de validation telles que la vérification à l'exécution [EGT+09, HEZ+14, KGFP07] ou des combinaisons de différentes approches [CEG+14, HAS14, HKWW17]. Parmi les 49 articles cités dans cet état de l'art, 32 portent sur l'utilisation de techniques de model-checking, 3 de theroem-proving et 14 sur des approches diverses (réseaux de Petri temporisés, Z...).

Theorem proving. Le *theorem proving* ou *preuve par le théorème* est aussi une technique de vérification de systèmes et notamment dans le domaine de la robotique. Elle permet de produire une preuve formelle sur le comportement réel d'un système. Par exemple, dans le cadre de la gestion d'un parc de véhicules autonomes en Allemagne, [RKH+17] utilise l'assistant de preuve Isabelle/HOL (Higher-Order Logic - Logiques d'ordre supérieur) [NPW02] et la logique temporelle (section 4.2.1) pour formaliser les règles de conduite des véhicules autonomes. Isabelle est un système générique pour implémenter des formalismes logiques. Sous Isabelle, une théorie est composée de déclarations de type, de définitions et de théorèmes, pour la plupart prouvés par la composition de théorèmes existants. RobotChart (section 2.1) utilise également cet outil pour la vérification des machines à états modélisant les comportements robotiques [FBC+18].

Model-checking Comme le montre [LFD+18], le model-checking est l'une des approches les plus courantes pour la robotique. Elle consiste en une technique automatique de vérification des systèmes réactifs à état finis. Les spécifications sont exprimées en une logique temporelle et le système est modélisé sous la forme d'un graphe de transitions d'états [CGP99]. De nombreuses recherches sont menées sur ces sujets et des formalismes variés ont été proposés tant pour la représentation du modèle de système que pour l'expression des propriétés. Des outils tels que SPIN [Hol03], SMV [BBF+01b], UPPAAL [GLPY97] permettent d'automatiser le processus et d'obtenir une trace contre-exemple en cas de non respect d'une propriété sur un système.

Ici, nous nous intéressons plus particulièrement aux techniques appliquées pour les automates temporisés [AD92]. Le *model-checking* permet de vérifier, pour un automate temporisé, un ensemble de propriétés sur la qualité d'exécution du système (invariants temporel ou non sur l'ensemble des exécutions possibles, accessibilité d'un état, etc.). Ces propriétés peuvent être décrites en logique temporelle temporisée.

4.2.1 Logique temporelle

Les logiques temporelles sont utilisées pour l'expression de propriétés à vérifier au travers de modèles qui peuvent être linéaires ou arborescents. Il existe différentes formes de logiques temporelles, parmi lesquelles :

- LTL : Linear-time Temporal Logic [Pnu77] introduisant les opérateurs pour une réflexion temporelle sur le système (opérateurs toujours et fatalement) ;
- CTL : Computation Tree Logic [CES86] proposant une généralisation de LTL avec les quantifieurs de chemins (il existe et pour tout) ;

Nous nous attardons ici plus particulièrement sur l'extension temporisée de CTL : TCTL, permettant la vérification de système intégrant des contraintes temporelles. En tant qu'extension de CTL, TCTL propose les deux opérateurs pour la vérification des propriétés :

- les opérateurs de persistance : $\diamond$ et $\Box$ pour exprimer une condition sur la satisfaction d'une propriété ϕ. Ainsi, $\diamond$ pour exprimer que la propriété est parfois vérifiée au cours de l'exécution et $\Box$ pour exprimer que la propriété est toujours vérifiée (il s'agit alors d'un invariant de l'exécution).
- les opérateurs de sélection arborescente : $\forall$ et $\exists$ pour exprimer un ensemble de chemins valides. Ainsi $\forall$ se réfère à tous les chemins et $\exists$ au moins un chemin ;

La syntaxe utilisée par la logique TCTL est définie par :

$$\phi ::= s_k \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \forall\psi \mid \exists\psi$$
$$\psi ::= \diamond\ \phi \mid \Box\ \phi$$

avec ϕ une expression linéaire sur les variables ou sur les localités du système.

Spécification de propriétés TCTL TCTL fournit alors quatre types de propriétés arborescentes (figure 4.2) :

- $\forall\ \Box\ \phi$: pour tous les chemins de l'arbre, ϕ est toujours vérifiée ;
- $\exists\ \diamond\ \phi$: ϕ est fatalement vérifiée dans au moins une localité ;
- $\forall\ \diamond\ \phi$: pour tous les chemins, ϕ est fatalement vérifiée ;
- $\exists\ \Box\ \phi$: il existe un chemin de l'arbre où ϕ est toujours vérifiée.

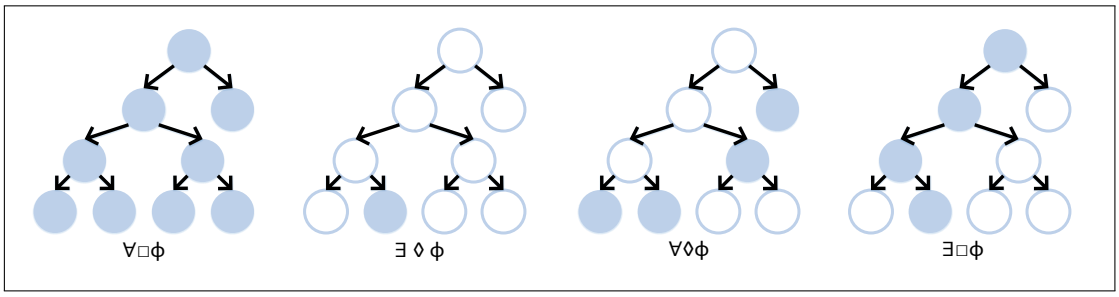


FIGURE 4.2 – Représentation des propriétés TCTL

La vérification de propriétés temporelles s'avèrent particulièrement utile dans le contexte robotique. En effet, les contraintes temporelles sont indispensables pour la spécification des comportements intégrant une dynamique interactive (par exemple des temps d'écoute pour la reconnaissance vocale). Outre les aspects temporels, la nécessité de procéder à une exploration de l'espace d'états du système se fait ressentir afin d'éviter tout problème de *deadlock*.

Trois types de propriétés peuvent ainsi être vérifiées en utilisant cette logique temporelle :

- vivacité : un événement va éventuellement se produire (et ce sous certaines conditions) ;
- accessibilité : un état particulier du système peut être atteint ;
- sûreté : un événement ne doit jamais se produire ou un état du système ne doit jamais être atteint.

Si des outils comme *PRISM* [KNP11], pour les systèmes probabilistes ou *KRONOS* [BBF+01a], qui permet de vérifier des systèmes temporisés à base de réseaux d'automates, s'avèrent efficaces, nous nous intéressons particulièrement au *model-checker* UPPAAL, sur lequel s'appuie notre formalisme.

4.2.2 L'outil UPPAAL

UPPAAL (UPPsala + AALborg) [GLPY97] est un framework utilisé pour la modélisation et la simulation de systèmes temps-réel par automates temporisés communicants par synchronisation binaire. UPPAAL comprend également un outil de model-checking pour vérifier les propriétés de sûreté et d'accessibilité. Cependant, l'une de ses limitations concerne la vérification de propriété de vivacité. En effet, pour vérifier ces propriétés, il faut passer par un automate d'observation (*testing automata* - [JLS02]). UPPAAL étant un outil particulièrement efficient pour la vérification de propriétés sur les automates temporisés, plusieurs contributions utilisent cet outil pour prouver la robustesse de l'approche (par exemple, dans les systèmes de communication hétérogène [LPPR13], dans la vérification des systèmes à déclenchement temporel [SSF+18], dans la vérification de robots autonomes (chargeuse de pneus) [GMSL18] ou dans l'analyse des documents normatifs [CHS18]).

4.2.2.1 Modélisation

UPPAAL utilise la logique temporelle TCTL pour la vérification de propriétés. Prenons,l'exemple simple d'un passage à niveau temporisé. Ce système intègre trois processus : le train, la barrière et un contrôleur permettant d'orchestrer le système. Le cahier des charges de ce système est le suivant :

- La barrière doit mettre 15 secondes maximum pour s'abaisser et au maximum 10 pour remonter ;
- Le train est proche du PN (passage à niveau) sur une durée de 20 à 30 secondes et traverse le PN au maximum en 5 secondes ;
- Lorsque que le contrôleur a connaissance de l'approche d'un train sur le PN, il dispose d'une seconde exactement pour déclencher la fermeture de la barrière et la fait remonter une seconde après que le train a libéré le PN.

La représentation sous forme d'automates temporisés communicants est donnée à la figure 4.3.

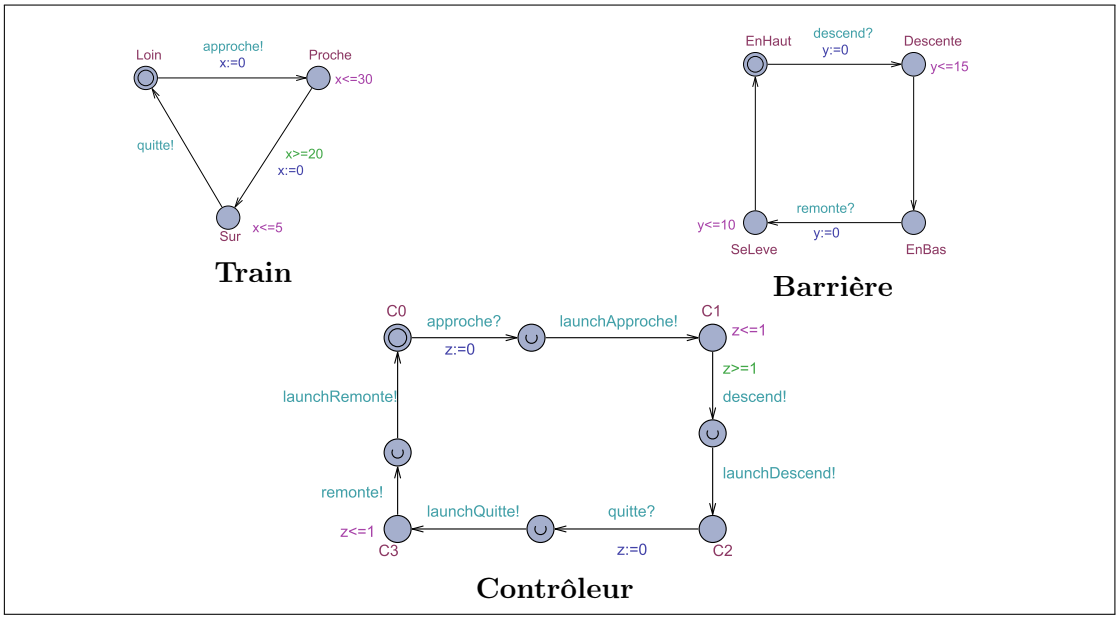


FIGURE 4.3 – Représentation du PN temporisé sous Uppaal

4.2.2.2 Expression des propriétés

Le concepteur de ce système va pouvoir ensuite vérifier :

- des propriétés de sûreté :
  - P_1 : la barrière ne peut pas être en haut lorsque le train est sur le PN ;
  - P_2 : la barrière ne peut pas être en bas lorsqu'aucun train n'est proche du PN.
- des propriétés de vivacité (avec l'automate observateur de la figure 4.4) :
  - P_3 : l'approche d'un train entraînera la descente de la barrière ;
  - P_4 : l'éloignement d'un train entraînera la levée de la barrière ;
- une propriété d'accessibilité :
  - P_5 : l'absence de deadlock.

Le tableau 4.5 présente le formalisme UPPAAL des propriétés que l'on souhaite vérifier et leur résultat. L'une des propriétés n'est pas vérifiée. En effet, le processus modélisant la barrière peut être dans la localité *EnBas* lorsque le processus du train est dans la localité *Sur*.

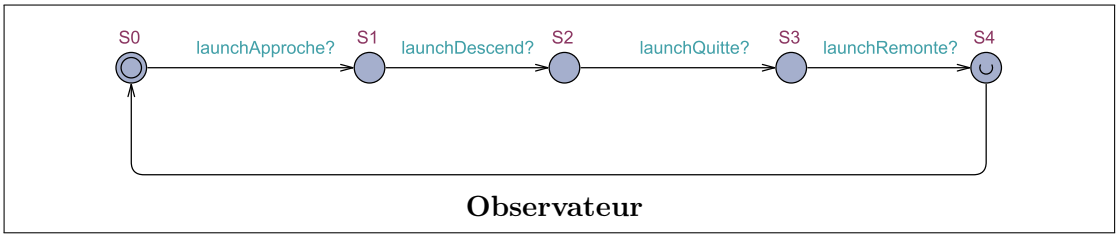


FIGURE 4.4 – Représentation de l'automate observateur pour les propriétés de vivacité

Propriété	formulation UPPAAL	Resultat
P_1	A[] not (Barriere.EnHaut && Train.Sur)	
P_2	A[] not (Barriere.EnBas && not Train.Proche)	
P_3	E<> Observateur.S2	
P_4	E<> Observateur.S4	
P_5	A[] not deadlock	

FIGURE 4.5 – Résultats des analyses d'accessibilité, de sûreté et de vivacité sur l'exemple du PN.

4.2.2.3 Vérification et algorithme d'analyse d'accessibilité

La mécanique de vérification UPPAAL tient dans une analyse d'accessibilité des états (en se basant sur le principe qu'une propriété de sûreté est la négation d'une propriété d'accessibilité). Les propriétés de vivacité sont alors vérifiées par test de l'accessibilité d'un état de l'automate observateur. L'algorithme calcule "à la volée" les états accessibles du système et les compare avec ceux exprimés dans la propriété en cours de vérification (algorithme 3)

4.3 Extension du model-checking pour le modèle CIT

Comme nous l'avons présenté dans l'état de l'art, toute modélisation formelle d'activité nécessite une phase de vérification. UPPAAL [GLPY97] étant l'un des principaux outils de vérification pour automates temporisés, nous présentons dans cette section les règles permettant de convertir le modèle CIT en modèle UPPAAL, afin d'appliquer une analyse par model checking.

Un modèle UPPAAL est représenté par un fichier XML, dans lequel le comportement global du système est divisé en processus (*template* dans le fichier XML). Chaque processus possède son propre automate ainsi que des variables, canaux de communication (channels) et fonctions pouvant être utilisées pour la spécification de contraintes sur les transitions. Ces variables et fonctions peuvent être déclarées localement (c.a.d. propre à chaque processus) ou de manière globale, pour l'ensemble du système.

Données : Analyse d'accessibilité UPPAAL
Résultat : Ensemble des états accessibles

passed := {} ;
waiting := $\{l_0\}$;
tant que *waiting != {}* **faire**
  waiting := waiting - {S};
  si $S \notin$ *passed* **alors**
    passed := passed $\cup$ S ;
    pour tous *S' tel que* $S \longrightarrow S'$ **faire**
      waiting := waiting $\cup$ S;
    fin
  fin
fin

Algorithme 3 : Analyse d'accessibilité UPPAAL

4.3.1 Conversion des variables du modèle CIT

Le modèle *CIT* propose trois différents types de variables :

- les variables déclaratives qui peuvent être utilisées dans la définition des comportements atomiques. Ces variables sont converties dans le modèle UPPAAL par des variables globales ;
- les variables d'agents utilisées pour la spécification des comportements au sein des agents. Elles sont converties en variables de processus, spécifiques aux processus impliquant les agents ;
- les variables globales utilisables par l'ensemble des contextes, convertis en variables globales sous UPPAAL.

Dans le modèle UPPAAL, ces différentes variables peuvent être des entiers (*int*) ou des valeurs booléennes (*bool*). Cependant, les variables du modèle CIT peuvent également prendre en charge les chaînes de caractères. Une conversion des chaînes de caractères en tableau d'entiers de taille fixe est alors nécessaire. Les opérations traditionnelles d'égalité et de différence doivent également être définies. Un exemple de conversion d'une variable de type string et des fonctions de comparaison associées est donné dans la figure 4.6. Dans cet exemple, la variable *nom* a été convertie en *nomTab* (ligne 37) sous forme de tableaux d'entiers.

4.3.2 Conversion des contextes du modèle CIT

Un contexte dans le modèle CIT représente une entité faisant intervenir un certain nombre d'agents. Son automate temporisé de comportements est alors obtenu par le

```
<nta>
    <declaration>
        /* constante utilisée pour la taille max d un tableau d entiers */
        const int nmax = 50;

        /* Retourne la taille d un tableau */
        int len(int tab[nmax])
        {
            int length=0;
            int i;
            for( i=0;i<nmax;i=i+1)
            {
                if(tab[i]!=0)
                    length +=1;
            }
            return length;
        }

        /* Fonction équivalente a compareTo en JAVA */
        int distance(int t1[nmax], int t2[nmax])
        {
            if(len(t1)==len(t2))
            {
                int i;
                int diff = 0;
                for( i=0;i<len(t1);i=i+1)
                {
                    if(t1[i] != t2[i])
                        return 1;
                }
                return 0;
            }
            return -1;
        }

        /* Variable "nom" convertie en int[] */
        int nomTab[nmax];

        /* Assigne l équivalent d une chaîne vide dans un tableau d entiers */
        void reset(int& tab[nmax])
        {
            int i;
            for(i = 0; i < nmax; i=i+1)
            {
                tab[i] = 0;
            }
        }
        ...
    </declaration>
    ...
<nta>
```

FIGURE 4.6 – Exemple de conversion d'une variable de type *string* dans le modèle UPPAAL.

produit synchronisé de l'ensemble des automates des agents le composant. Un contexte représente donc une situation d'exécution particulière pour le scénario désiré. Ainsi, sa

conversion sous UPPAAL peut être réalisée par la création d'un processus, possédant son propre automate et ses propres variables. Comme présenté ci-dessus, UPPAAL n'intègre pas la gestion des chaînes de caractères. L'ensemble des transitions faisant intervenir des contraintes sur des chaînes de caractères doivent être converties pour satisfaire au modèle UPPAAL. Par exemple, la modélisation du comportement permettant d'exécuter une reconnaissance vocale sur le robot nécessite l'utilisation de chaînes de caractères pour la gestion de l'expression reconnue par le robot. L'exemple de la figure 4.7 illustre cette représentation, via la transition *demanderPrenom*. L'exécution de ce comportement renvoie une chaîne de caractères dans la variable *nom*. Cette chaîne de caractères est convertie sous forme de tableau d'entiers dans la variable *nomTab*. Cette conversion est réalisée en code ASCII. Ainsi une majuscule est convertie dans l'intervalle [65 ;90] et les minuscules dans l'intervalle [97 ;122]. La vérification de contraintes sur les transitions *erreur* et *direBonjour* se fait alors grâce aux fonctions de comparaison décrites dans la figure 4.6.

Une deuxième modification de l'automate de contexte est nécessaire pour la conversion de notre modèle sous UPPAAL. A partir du graphe de contexte, l'algorithme de supervision du modèle CIT prévoit le chargement automatique du contexte initial (*promenade* dans l'exemple de la figure 3.10). Or le modèle UPPAAL ne permet pas d'exécuter un processus automatiquement à partir d'un autre. Pour simuler ce chargement automatique, nous ajoutons pour chaque contexte un canal de communication (chan) portant le nom du contexte préfixé de "lancer". Ainsi, la première transition de chaque contexte nécessitera la réception de ce message et permettra son exécution. Par exemple, dans la figure 4.7, la transition de l'état initial S_0 à l'état S_1 propose une synchronisation avec le graphe de contexte par l'attente de réception du signal *lancerAccueil*.

Enfin, la troisième et dernière modification consiste à réaliser un bouclage entre l'état final et l'état initial. Sous UPPAAL, l'algorithme de supervision garde en mémoire l'état courant du processus à la fin de son exécution et ne permet pas sa remise à zéro automatiquement (c'est à dire une nouvelle exécution du contexte à partir de l'état initial). Nous créons donc un bouclage artificiel, par la création d'une transition entre l'état final (à l'origine) et l'état initial sur laquelle est réalisée une synchronisation avec le graphe de contexte par l'envoie d'un signal.

Dans l'exemple de la figure 4.7, la transition entre les états S_3 et S_0 est ajoutée et permet l'envoie du signal *jouer*. Ce message sera alors reçu par le graphe de contexte qui est en attente de réception.

4.3.3 Conversion du graphe de contexte du modèle CIT

La conversion de notre graphe de contexte au format UPPAAL nécessite des adaptations et des transformations. En effet, UPPAAL permet une exécution parallèle des différents processus, ce que notre plateforme ne permet pas. Chaque contexte est exécuté de façon séquentielle, par échange de messages. Comme nous l'avons vu précédemment, l'exécution d'un contexte par l'échange de message a nécessité certaines modification dans l'automate du contexte (attente d'un message pour son exécution et envoie d'un message à la fin de l'exécution). Le graphe de contexte doit donc également être converti

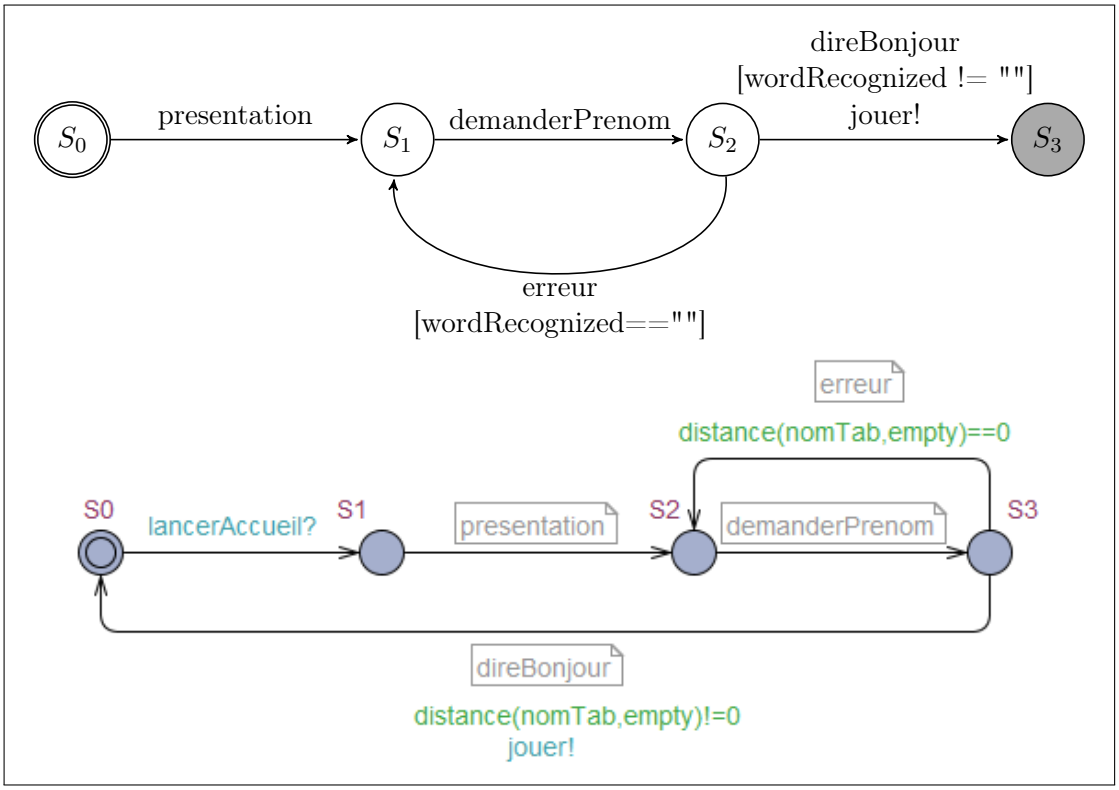


FIGURE 4.7 – Contexte "Accueil" dans notre modèle (en haut) et sous UPPAAL (en bas)

pour être en capacité d'intercepter les échanges de messages pouvant avoir lieu avec les différents contextes. Ainsi, l'exécution d'un contexte sera réalisée par l'envoie d'un message à partir d'un état "urgent" (état qui ne laisse pas le temps s'écouler). Le passage d'un contexte à un autre reste inchangé par rapport au modèle CIT et est donc réalisé par synchronisation (exemple figure 4.8).

Nous présentons en détail, dans l'algorithme 4, les différentes étapes de conversions de notre modèle.

Données : Modèle CIT *input*
Résultat : Modèle UPPAAL *output*

pour tous *Channel c* ∈ *input.channels()* **faire**
  Créer un canal de communication (chan) dans output;
fin
pour tous *Clock c* ∈ *input.clocks()* **faire**
  Créer une horloge dans output;
fin
pour tous *variableDeclarative dv* ∈ *input.declarativeVariables()* **faire**
  si *dv.type == "string"* **alors**
    Créer une variable globale de type int[] dans output avec la valeur de *dv* convertie en code ASCII;
  sinon
    si *dv.type == "double"* **alors**
      Créer une variable globale de type int avec la valeur de *dv* * 100 dans output;
    sinon
      Créer une variable globale de type int avec la valeur de *dv* dans output;
    fin
  fin
fin
pour tous *VariableGlobale gv* ∈ *input.globalVariables()* **faire**
  si *gv.type == "string"* **alors**
    Créer une variable globale de type int[] dans output avec la valeur de *gv* convertie en code ASCII;
  sinon
    si *gv.type == "double"* **alors**
      Créer une variable globale de type int avec la valeur de *gv* * 100 dans output;
    sinon
      Créer une variable globale de type int avec la valeur de *gv*' dans output;
    fin
  fin
fin
Ajouter les fonctions manipulant les tableaux d'entiers (voir figure 4.6);
Créer un template (processus) pour simuler le robot;

pour tous *Action a pouvant être exécutée sur le robot* **faire**
  Créer un canal de communication pour simuler l'action;
  Créer une localité et une transition pour la simulation de a;
  Spécifier les contraintes et paramètres de a en utilisant la base de données;
fin
pour tous *Context* $c \in$ *input* **faire**
  templateContexte ← nouveau Template;
  Créer une localité initiale S_0 et une localité S_1 dans templateContexte;
  Créer une transition de S_0 à S_1 permettant une synchronisation avec le graphe de contexte en utilisant le canal "lancer" + nom du contexte c;
  pour tous *Localite* $s \in$ *automate de* c **faire**
    Créer une nouvelle localité dans templateContexte;
    Spécifier l'invariant;
  fin
  pour tous *Transition* $t \in$ *automate de* c **faire**
    Créer une nouvelle transition avec les paramètres de t;
    Convertir les contraintes de chaînes de caractères en tableau d'entiers;
    Utiliser le canal de communication approprié pour simuler une action avec le robot;
  fin
  Réaliser le bouclage vers la localité initiale ;
fin
templateGrapheContexte ← nouveau Template;
pour tous *Localite* $s \in$ *automate du graphe de contexte* **faire**
  Créer une nouvelle localité $stUrgent$ ayant pour nom le nom de s suffixé de "urgent" avec la propriété "Urgent" sous UPPAAL et la rendre initiale si s l'est;
  Créer une nouvelle localité st possédant le nom de s;
  Créer une transition de $stUrgent$ à st avec la synchronisation portant le nom de "lancer"suffixé du nom de s;
fin
pour tous *Transition* $t \in$ *automate du graphe de contexte* **faire**
  Créer une transition avec les localités adéquates;
fin
Créer le système UPPAAL contenant le processus de simulation du robot, les contextes et le graphe de contexte;

Algorithme 4 : Algorithme de conversion du modèle CIT au modèle UPPAAL

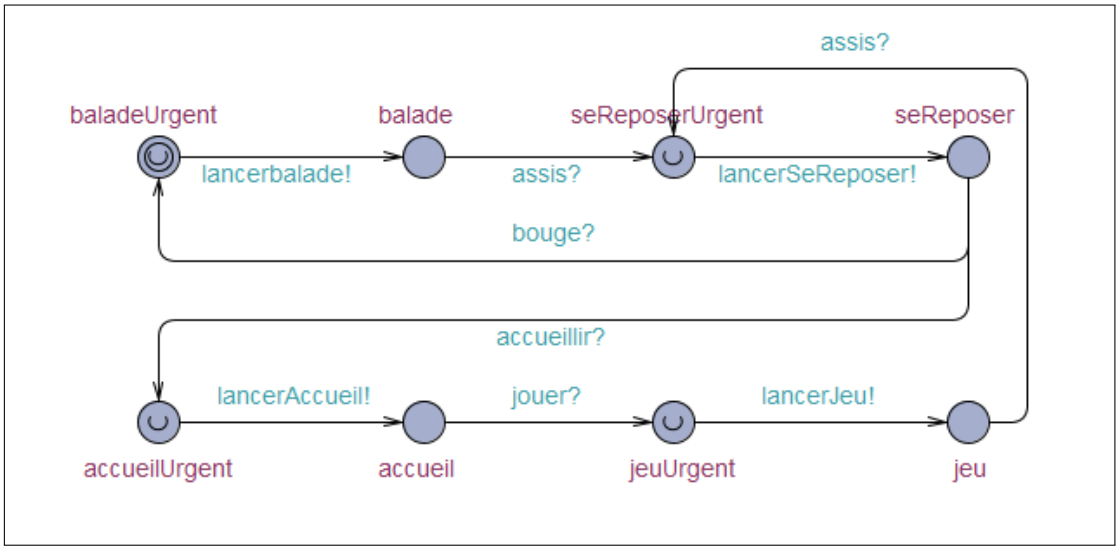


FIGURE 4.8 – Conversion du graphe de contexte présenté en figure 3.10 sous UPPAAL

4.4 Conclusion

Dans ce chapitre, nous avons présenté la sémantique dynamique du modèle CIT. Celle ci est basée sur les contraintes temporelles associées à chaque localité et chaque transition du modèle. Ce modèle, comme tout modèle, nécessite une phase de vérification. Cette étape permet de garantir, pour le concepteur, que le modèle produit répond en tout point à ses attentes. Pour cela, le modèle CIT est converti dans le formalisme de l'outil UPPAAL. Le concepteur peut ainsi vérifier trois types de propriétés : vivacité, accessibilité et sûreté.

Lors de la conception d'un modèle de pilotage de tâches robotiques, le concepteur va spécifier certains paramètres pilotant l'expérience. Par exemple, lors d'une reconnaissance vocale, il va pouvoir spécifier qu'en dessous d'un certain seuil de reconnaissance, la réponse donnée par l'utilisateur ne sera pas considérée comme correcte. Ces seuils peuvent, dans certaines conditions, présenter un obstacle pour la bonne réalisation de la tâche. Nous présentons donc, dans le chapitre suivant, notre méthode d'adaptation de paramètres, basée sur un algorithme de renforcement.

Chapitre 5

Apprentissage automatique au cours de l'interaction

Sommaire

Comme nous avons pu le voir dans les chapitres précédents, l'exécution d'une expérience interactive destinée à la robotique est conditionnée et contrainte par un ensemble de paramètres, de temps ou représentant une contrainte de variables.

Prenons un exemple simple, qui est celui décrit en figure 5.1. Dans cet exemple, le concepteur a modélisé une interaction verbale entre un humain et un robot. Le robot pose une question à l'utilisateur et se met en écoute de la réponse pendant D unité de temps. Le robot renvoie alors un couple (mot reconnu (non représenté dans l'automate), taux de confiance t_c) si l'utilisateur a répondu dans le temps imparti et renvoie un message d'erreur dans le cas contraire. A la fin de l'exécution de ce comportement, trois cas possibles émergent de cette interaction :

- CU_1 : l'utilisateur répond dans le temps imparti ($\leq D$) et la valeur du taux de confiance t_c renvoyée par le robot est supérieure au seuil C : la réponse va être prise en compte (S_1, S_2) ;
- CU_2 : l'utilisateur répond dans le temps imparti ($\leq D$) mais la valeur du taux de confiance t_c est inférieure au seuil C : la réponse va être considérée comme

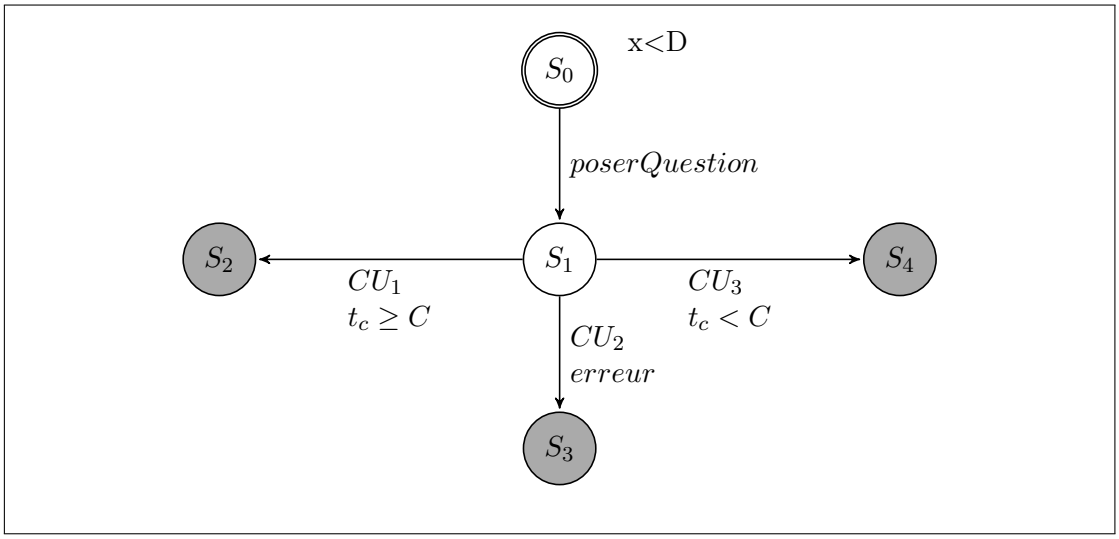


FIGURE 5.1 – Automate temporisé représentant une interaction verbale

incertaine (S_1, S_4) ;

- CU_3 : l'utilisateur ne répond pas dans le temps imparti ($> D$) : le robot renvoie alors un message d'erreur (S_1, S_3).

A travers cet exemple simple, nous remarquons que la bonne réalisation de cette interaction est conditionnée par une valuation correcte des paramètres C et D. Si cette tâche peut être difficile pour le concepteur, nous souhaitons proposer une méthode d'adaptation dynamique de ces paramètres pour soulager le concepteur de cette tâche difficile. Cette adaptation dynamique aura pour conséquence d'éviter une certaine déception de l'utilisateur, dont l'expérience ne serait pas adaptée à ses attentes.

Nous présentons donc, dans ce chapitre, un rapide état de l'art sur les systèmes adaptatifs et plus particulièrement en robotique. Cet état de l'art nous permettra, dans un second temps, d'aboutir à notre proposition d'adaptation dynamique de l'activité.

5.1 Systèmes adaptatifs

L'utilisation des systèmes adaptatifs est répandue dans la vie quotidienne de chacun. Qu'il s'agisse de smartphones ou tablettes, de site marchands ou encore de plateformes de streaming vidéo, la nécessité de disposer d'outils permettant de s'adapter à l'expérience utilisateur est primordiale (notamment pour les ventes).

L'adaptation des comportements d'un système lors de son exécution est une tâche difficile et largement développée ces dernières années dans l'état de l'art [MSD19, KRV+15]. La difficulté principale est due à l'absence de méthodes précises pour la description et la modélisation des systèmes.

Dans le domaine de la robotique, de nombreuses approches utilisent l'apprentissage par renforcement pour cette problématique (section 5.1.1) et notamment pour les robots assistants/domestiques. C'est le cas du robot Paro [WS06]. Conçu à l'image d'un phoque, ce robot thérapeutique est destiné aux personnes atteintes de troubles du comportement et de la communication. L'utilisation de l'apprentissage par renforcement permet d'adapter les intonations et mouvements du robot. [TŢM08] propose eux aussi un robot thérapeutique destiné aux patients ayant subi un AVC. Le rôle de ce robot est de surveiller, assister, encourager les patients en proposant des exercices de réhabilitation. L'adaptation par renforcement est réalisée par l'observation de trois paramètres : la distance entre le robot et le patient, la vitesse des mouvements et la communication verbale (vitesse et volume).

5.1.1 Apprentissage par renforcement

L'objectif de l'apprentissage par renforcement est de trouver, par essais et erreurs, l'action optimale à effectuer. Chaque action du système est associée à une récompense (positive, négative ou nulle). Cette récompense permet d'évaluer la qualité d'une action particulière. L'objectif pour un agent est de maximiser la récompense accumulée au cours de l'exécution du système.

Trois éléments formalisent l'apprentissage par renforcement :

- S : un ensemble d'états correspondant à l'environnement dans lequel évolue l'agent ;
- A : un ensemble d'actions réalisables par l'agent ;
- $R : \{S, A\} \rightarrow \mathbb{R}$: une fonction de récompense associant un état et une action.

La figure 5.2 représente ce formalisme. L'agent, ici un robot, va interagir avec son environnement en percevant l'état $s \in S$ dans lequel il se trouve. Il va alors effectuer une action $a \in A$ et percevoir la récompense r associée.

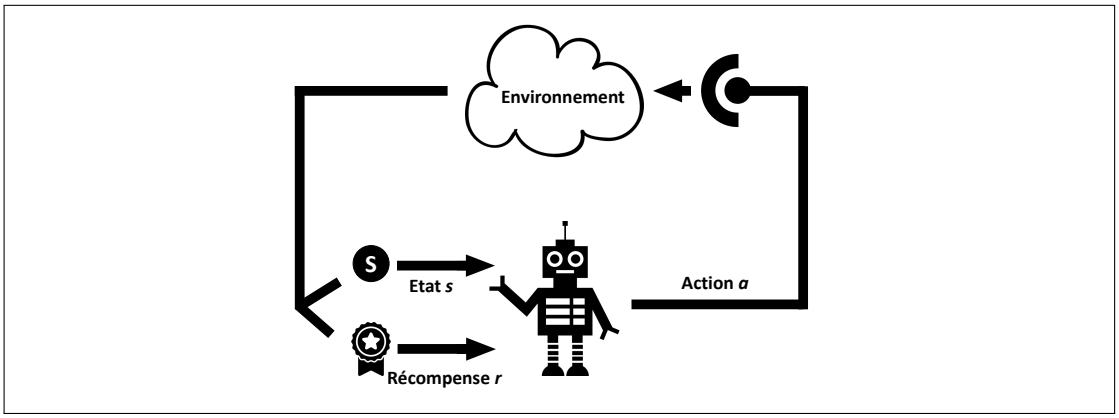


FIGURE 5.2 – Formalisation du problème d'apprentissage par renforcement

Le comportement de l'agent est défini par une stratégie $\pi : \{S, A\} \rightarrow [0, 1]$ de manière probabiliste par la spécification pour chaque état s de la probabilité d'exécution de l'action a. L'évolution de l'agent dans l'environnement est géré par un processus de décision markovien (toute l'information nécessaire au choix de la prochaine action se trouve dans l'état courant). L'objectif pour les algorithmes d'apprentissage par renforcement est de trouver la stratégie optimale π^* permettant de maximiser la récompense à long terme.

Pour évaluer et donc estimer le gain moyen d'une stratégie π pour un état donné, les algorithmes d'apprentissage par renforcement utilisent une fonction de valeur. Cette fonction de valeur diffère selon les trois classes d'algorithme d'optimisation que sont :

a. la programmation dynamique ;

b. les méthodes de Monte-Carlo ;

c. les méthodes de différence temporelle.

a. Programmation dynamique Lorsque l'agent connaît l'environnement dans lequel il va évoluer, les algorithmes de programmation dynamique sont applicables. L'évaluation de la stratégie π repose dans ce cas sur l'équation de Bellman et est définie par :

$$V(s) = \sum_a \pi(s, a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')] \tag{5.1}$$

b. Méthodes de Monte-Carlo Contrairement aux méthodes de programmation dynamique, les méthodes de Monte-Carlo sont destinées aux applications pour lesquelles l'environnement n'est a priori pas connu. Ces méthodes sont donc particulièrement adaptées en robotique. La fonction de valeur est définie par :

$$V(s) = V(s) + \alpha[R(s) - V(s)] \tag{5.2}$$

c. Méthodes de différence temporelle Les méthodes de différence temporelle [Sut88] constituent les méthodes les plus utilisées en robotique. Elles combinent les deux avantages des méthodes précédentes, à savoir :

- apprendre à partir d'expériences sans connaissance a priori de l'environnement (Monte-Carlo) ;
- utilisent les estimations des états successeurs pour estimer la valeur d'un état (programmation dynamique).

La principale méthode est celle du **Q-Learning** [WD92] et utilise le maximum de la fonction de valeur sur les actions suivantes et ne repose donc plus sur l'action effectivement réalisée. La fonction de valeur ne repose pas seulement sur l'état mais sur le couple état/action et est définie par :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (5.3)$$

avec $0 \leq \alpha \leq 1$, le facteur d'apprentissage ; $0 \leq \gamma \leq 1$, le facteur d'actualisation ; s, l'état précédent ; s_{t+1}, le nouvel état dans lequel le système se trouve après l'exécution de l'action a_t, associée à la récompense r_{a_t}.

Le choix des valeurs de α et de γ n'est pas anodin. En effet, initialiser le facteur d'apprentissage α à la valeur 0 ne permettra pas à l'agent d'apprendre de ses actions, tandis qu'une valeur de 1 ne permettrait à l'agent de prendre en compte que les actions futures et non celles s'étant déjà réalisées. De même, le facteur d'actualisation γ initialisé à 0 ne permettrait pas à l'agent de prendre en considération les récompenses futures et inversement, une valeur de 1 lui permettrait de le faire.

Les méthodes de Q-Learning étant très adaptées à la robotique, nous pouvons les retrouver notamment dans les robots "footballeurs" [SLH+18, HGA15, HCL07] où l'adaptation consiste à l'attribution du meilleur rôle pour chaque robot joueur.

Comme le souligne Martins dans [MSD19], la grande majorité des adaptations réalisées en robotique consiste à ajuster certains paramètres tels que la vitesse du robot, le volume, les gestes ou encore la couleur des LED. Nous nous intéressons également, dans ces travaux de thèse, à l'ajustement de certains paramètres. Nous présentons donc dans la suite de la section les techniques d'adaptation associées.

5.1.2 Adaptation des comportements robotiques

Nous avons présenté dans le chapitre 2, différentes approches de modélisation de l'activité robotique. Afin de répondre le plus précisément aux attentes des utilisateurs, les plateformes, quelles qu'elles soient, doivent s'adapter à leurs comportements. Deux questions se posent alors :

- comment adapter en prenant en compte le contexte dans lequel l'interaction homme-robot a lieu (bruit, luminosité, heure, ...) et les caractéristiques de l'utilisateur (préférences, âge, ...) ?
- comment apprendre et faire évoluer le modèle de pilotage ?

Le principe général des architectures adaptatives est représenté à la figure 5.3 : (1) un moteur décisionnel choisi l'action à effectuer en fonction du contexte, des précédentes interactions avec l'utilisateur etc. (2) les paramètres contrôlant l'action sont adaptés pour répondre aux besoins de l'utilisateur ; (3) l'action est exécutée sur l'interface/robot ; (4) une interaction est réalisée avec l'utilisateur ; (5) un feedback est propagé pour reprendre une nouvelle décision.

La plupart des approches proposant d'adapter les comportements robotiques est basée sur un modèle formel. Par exemple, [KJM09] propose TAMER (Training an Agent Manually Via Evaluative Reinforcement) dans lequel une évaluation manuelle de l'action réalisée par le robot est effectuée par un observateur qui affecte une récompense. La phase

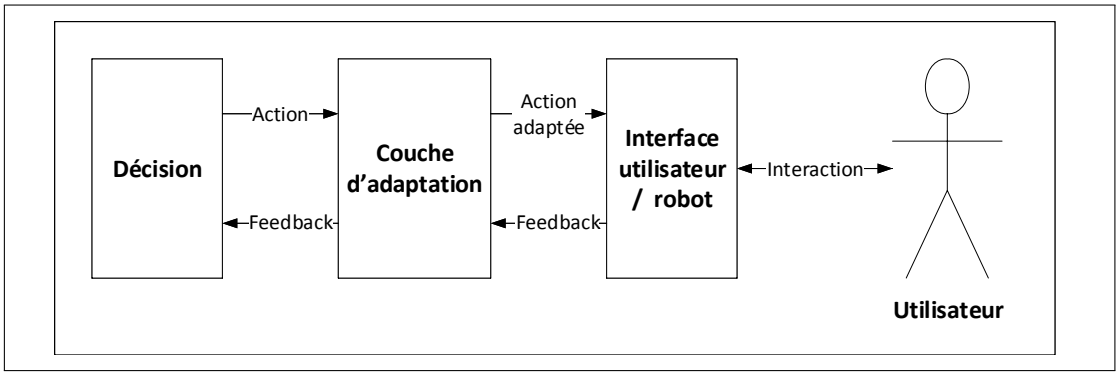


FIGURE 5.3 – Principe général des approches adaptatives

d'apprentissage consiste à déterminer les actions optimales pour maximiser la récompense. Toujours sur la base d'un *feedback*, [KSE13] propose une approche dans laquelle l'ensemble des informations concernant le profil de l'utilisateur et les différentes activités sont intégrées dans un Processus de Décision Markovien (MDP). La phase d'apprentissage est quant à elle réalisée grâce au *feedback* de l'utilisateur après chaque interaction. Ce *feedback* permet de déterminer quelles relations peuvent exister entre les actions du robot et les différents paramètres caractérisant un utilisateur ou une activité. Une fonction de récompense personnalisée est alors générée pour privilégier, dans les interactions futures, les actions optimales.

Enfin, les approches à base de règles sont aussi utilisées en robotique. [KI05] propose ainsi une plateforme robotique *Roovie* destinée aux élèves d'écoles élémentaires. Utilisé notamment pour l'apprentissage des langues étrangères, le robot adapte son comportement en fonction des interactions passées pour chaque élève. L'ordre d'exécution des activités et les adaptations sont prédéfinies par un système de règles. Le robot n'est ici pas capable d'optimiser automatiquement ses comportements à partir des interactions mais simplement par l'utilisation d'une base de règles.

5.2 Approche proposée

Comme nous l'avons vu dans la section précédente, les algorithmes d'apprentissage par renforcement adressent les problématiques de prise de décision et d'optimisation. L'objectif est d'optimiser les actions d'un agent pour maximiser une récompense à long terme.

Nous détaillons à présent, dans la section suivante, comment nous avons intégré cette phase d'apprentissage dans le modèle CIT.

5.2.1 Principe

Appliqué au modèle CIT, l'apprentissage par renforcement doit permettre d'optimiser certains paramètres de contrôle de l'activité. Ainsi, si le concepteur paramètre son activité

avec certaines contraintes sur les comportements du modèle, il est très probable que les valeurs données ne permettent pas de proposer une activité en adéquation avec les comportements réels de l'utilisateur final. Nous intégrons donc, un module d'apprentissage par renforcement basé sur l'algorithme du *Q-Learning*. Le principe général d'adaptation est donné en figure 5.4.

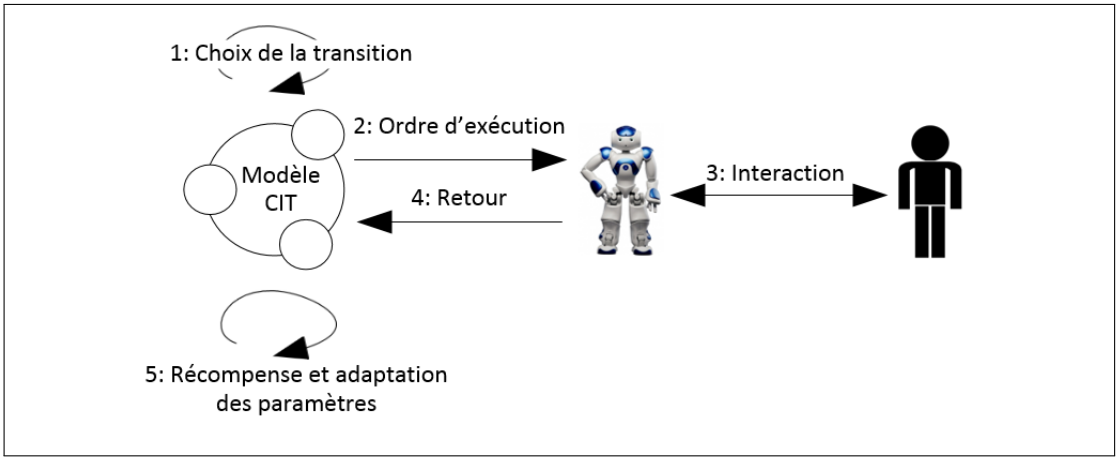


FIGURE 5.4 – Principe général d'adaptation du modèle CIT

Nous proposons d'adapter, dans un premier temps, les paramètres contrôlant les échanges vocaux entre l'utilisateur et le robot. Deux paramètres contrôlent l'exécution de ces échanges :

- la durée, notée D, pendant laquelle le robot est en capacité d'écouter l'utilisateur ;
- un seuil de reconnaissance, noté C, permettant de prendre en compte ou non la réponse donnée par l'utilisateur (à l'issue de l'exécution du module de reconnaissance vocale, le robot retourne le couple (mot reconnu, taux de confiance)).

Reprenons notre exemple présenté dans l'introduction de ce chapitre. Pour rappel, le robot demande à l'utilisateur son prénom et se met en écoute pendant D secondes (D est donc ici une contrainte d'invariant). Des trois cas d'utilisation qui découlent naturellement de cette interaction, nous souhaitons proposer des adaptations différentes des paramètres C et D :

- CU_1 : la valuation des paramètres C et D a permis de réaliser l'interaction correctement. La valeur de C doit être augmentée et celle de D diminuée ;
- CU_2 : la valuation de D était correcte mais celle de C n'a pas permis d'aboutir à une bonne interaction. Les valeurs de C et D doivent être diminuées ;
- CU_3 : la valuation de D n'a pas permis à l'utilisateur de répondre. la valeur du paramètre D doit être augmentée.

5.2.2 Modification dynamique du modèle

Pour répondre aux contraintes précédentes, nous associons, à chacune des transitions du modèle, une récompense r positive pour les transitions de succès ($r = 1$ pour la transition (S_1, S_2) dans l'exemple précédent) ou négative pour les transitions non désirables ($r = -1$ pour les transitions $((S_1, S_3)$ et $(S_1, S_4))$).

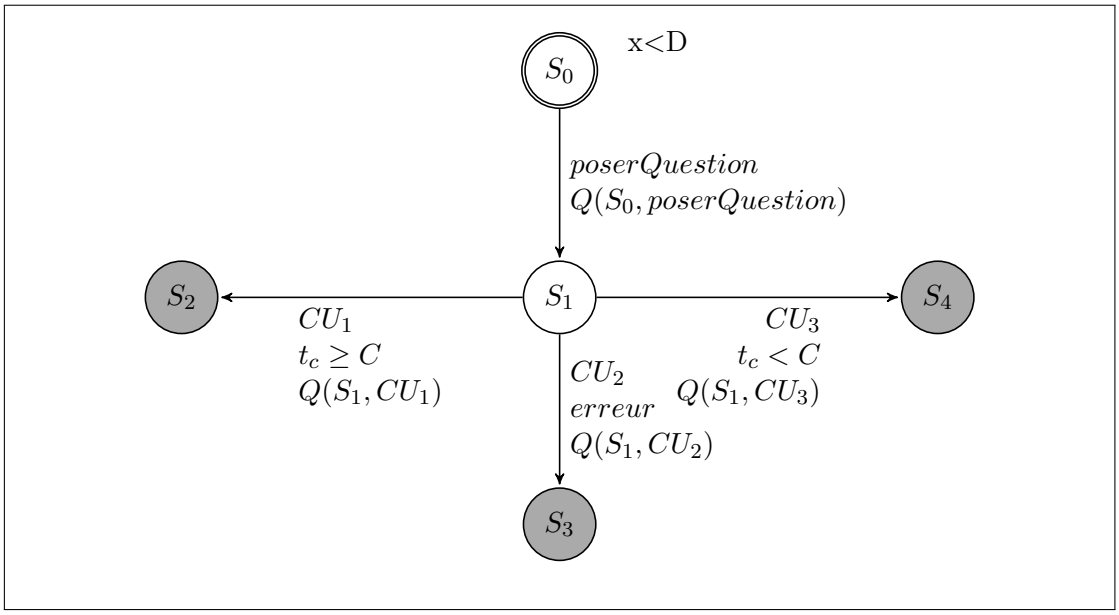


FIGURE 5.5 – Application du *Q-Learning* sur une interaction simple

Ces récompenses vont nous permettre d'appliquer le calcul de la fonction de valeur Q du *Q-Learning*. Nous obtenons ainsi, pour chaque couple (localité, transition) une valeur Q représentant la qualité de l'action réalisée au travers d'une transition. Cette valeur va être intégrée dans la fonction de mise à jour des paramètres contrôlant l'activité.

Nous proposons deux fonctions de mise à jour des paramètres de temps et de seuils de reconnaissance vocale :

$$D = D - r\left(\frac{D * |Q_p|}{|Q_p| + \epsilon}\right) \tag{5.4}$$

$$C = C + r\left(\frac{C * |Q_p|}{|Q_p| + \epsilon}\right) \tag{5.5}$$

avec Q_p la Q-valeur de l'action la plus défavorable.

Par exemple, dans l'exemple présenté en introduction de ce chapitre, l'action la plus défavorable est celle représentée par la transition (S_0, CU_2). La $Q - valeur$ qui lui est affectée sera donc la plus faible comparée aux autres Q-valeur de cet exemple. Nous avons choisi d'utiliser cette valeur pour éviter que l'écart entre l'ancienne valeur des paramètres et la nouvelle ne soit trop grand.

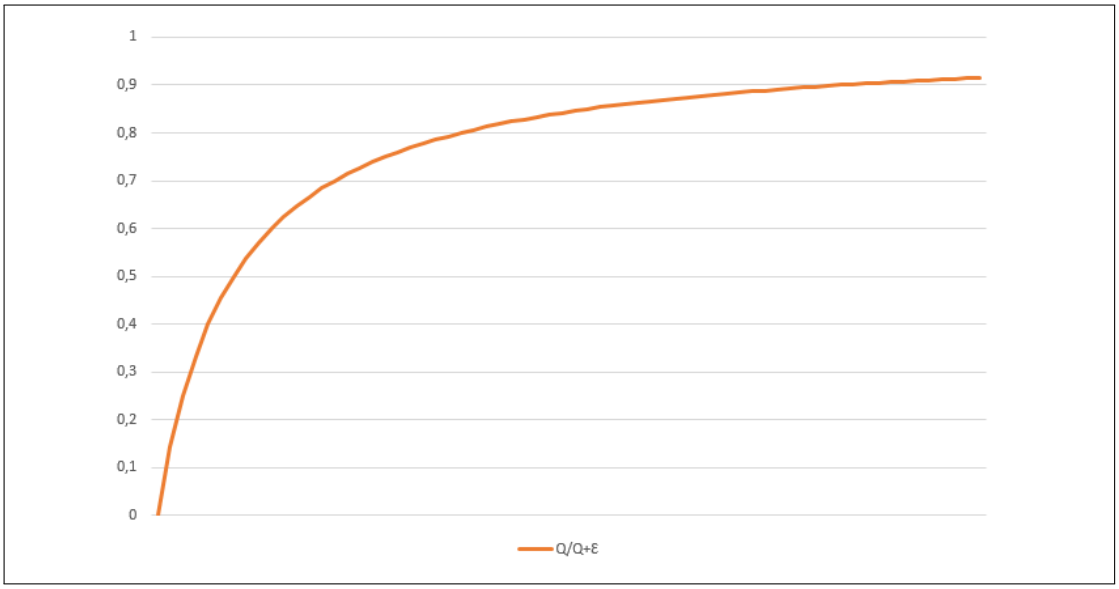


FIGURE 5.6 – Allure de $\frac{|Q|}{|Q|+\epsilon}$

La fonction $\frac{|Q|}{|Q|+\epsilon}$, dont la représentation est donnée en figure 5.6, varie entre [0 ;1]. Ainsi, pour chaque transition ayant des contraintes temporelles et/ou des contraintes liées au taux de confiance issu de la reconnaissance vocale, les nouvelles valeurs des paramètres varieront du simple au double. Le choix de ϵ n'est également pas anodin. Plus ϵ sera grand, plus la fonction $\frac{|Q|}{|Q|+\epsilon}$ prendra du temps pour converger vers 1.

Nous donnons l'algorithme d'adaptation des paramètres, appliqué au modèle CIT, ci-dessous. Nous rappelons que L représente l'ensemble des localités du modèle à base d'automates et T l'ensemble des transitions.

Initialiser Q(l,t) pour tout $l \in L, t \in T$ à 0;
Déterminer Q_p pour chaque paramètre de temps ou de seuil de reconnaissance;
pour tous *transition t exécutée* **faire**
  Observer r et l';
  $Q(l,t) = Q(l,t) + \alpha[r + \gamma max_a Q(l',a) - Q(l,t)]$;
  si *l est contraint par un invariant D* **alors**
    $D = D - r * (D * |Q_p|/(|Q_p| + \epsilon))$
  fin
  si *t est contraint par une garde sur le taux de confiance* t_c **alors**
    $C = C + r * (C * |Q_p|/(|Q_p| + \epsilon))$
  fin
fin

Algorithme 5 : Algorithme du *Q-Learning* appliqué au modèle CIT

5.3 Conclusion

Nous avons présenté dans ce chapitre notre algorithme d'adaptation des paramètres contrôlant l'activité. Nous utilisons pour cela le *Q-Learning* qui permet d'obtenir pour chaque couple (localité, transition) une valeur d'intérêt, prenant en compte à la fois l'expérience passée mais également l'espérance de gain pouvant être obtenue dans le futur. Cette valeur permet ensuite de mettre à jour les paramètres de temps et des seuils de confiance associés à la reconnaissance vocale.

Cette thèse a permis de mettre en oeuvre les différents modèles et algorithmes que nous avons présenté dans les chapitres précédents. Deux plateformes ont été développées permettant d'aboutir à une chaîne complète allant de la conception à l'exécution adaptative des activités. Nous présentons donc, dans le chapitre suivant, ces deux plateformes.

Troisième partie

Mise en œuvre et résultats

Chapitre 6

Mise en oeuvre de l'architecture CIT et supervision

Sommaire

L'ensemble des algorithmes que nous avons présentés dans les chapitres précédents ont fait l'objet d'une implémentation dans une plateforme que nous avons baptisée **CELTIC** pour **C**ommon **E**ditor for **L**ocation **T**ime **I**nteraction and **C**ontents. CELTIC est une plateforme d'édition et de spécification d'expériences interactives. Elle est composée d'un éditeur web (développé en partie par des étudiants de l'IUT de La Rochelle dans le cadre de projets tuteurés), utilisant une base de données MySQL pour la gestion externe des contenus, permettant de concevoir l'expérience de façon modulaire, grâce à la modélisation en deux couches (cf chapitre 3). Elle embarque également un générateur qui, à partir de la spécification, conçoit l'expérience sous forme d'automates temporisés communicants (chapitre 3).

Outre le fait de développer cette plateforme pour valider notre approche, nous souhaitons à terme la mettre à disposition de la communauté scientifique (mise à disposition des sources). Nous détaillons ici les possibilités offertes par notre plateforme pour la conception d'expériences interactives.

Enfin, notre outil **EDAIN** pour **E**xecution **D**river based on **A**rtificial **IN**telligence permettant l'exécution et la supervision du modèle sera présenté.

6.1 CELTIC

La figure 6.1 présente l'architecture de CELTIC et le processus de création d'un scénario. L'édition et la génération d'un scénario tiennent en six étapes :

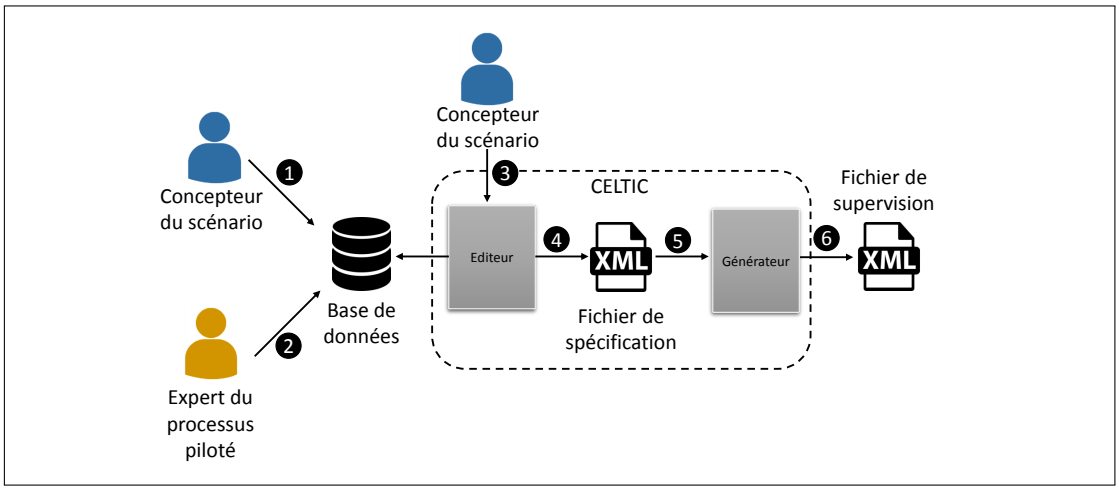


FIGURE 6.1 – Architecture de l'éditeur et du générateur de CELTIC

❶ Le concepteur du scénario édite l'ensemble des contenus utilisables lors de l'exécution du scénario (dans notre cas, les textes prononcés par Nao, mais ce pourrait être des sons ou des vidéos) ;

❷ L'expert du processus piloté insère en base de données l'ensemble des modules exécutables par le processus (reconnaissance vocale, reconnaissance faciale, mouvements...) ainsi que les paramètres d'entrées et de sorties nécessaires à leur exécution ;

❸ Le concepteur du scénario conçoit l'expérience grâce à l'éditeur, implémentant notre modèle CIT à deux couches ;

❹ Un fichier XML de spécification est généré ;

❺ Le fichier de spécification est utilisé pour la génération du scénario à base d'automates temporisés communicants. Le générateur intègre les algorithmes de composition présentés dans le chapitre 3. Il permet également la génération du modèle au format UPPAAL pour la vérification de propriétés ;

❻ Un fichier de supervision est enfin généré, contenant la modélisation XML sous forme d'automates temporisés du scénario interactif.

Nous commençons par présenter l'éditeur web de CELTIC et la base de données qu'il utilise lors de la phase de conception. Cet éditeur produit une spécification de l'expérience, sérialisée dans un fichier XML, qui sera ensuite utilisé par notre générateur

produisant les automates temporisés communicants par lesquels seront pilotés un processus.

6.1.1 Base de données

L'une des problématiques adressées dans ces travaux de recherche porte sur la gestion des contenus proposés (textuels, actions exécutables par le processus piloté...) dans un scénario. Nous avons proposé d'externaliser l'ensemble des contenus du scénario en base de données pour faciliter à la fois la conception d'une expérience mais surtout son maintien et sa reproductibilité. En effet, l'objectif est de pouvoir concevoir un scénario sans avoir la contrainte de la gestion des contenus et ainsi pouvoir proposer le même modèle de scénario pour un autre lieu (dans le cadre de nos expérimentations, introduites dans le chapitre 1, pour un autre musée) en ne modifiant seulement que les contenus en base de données.

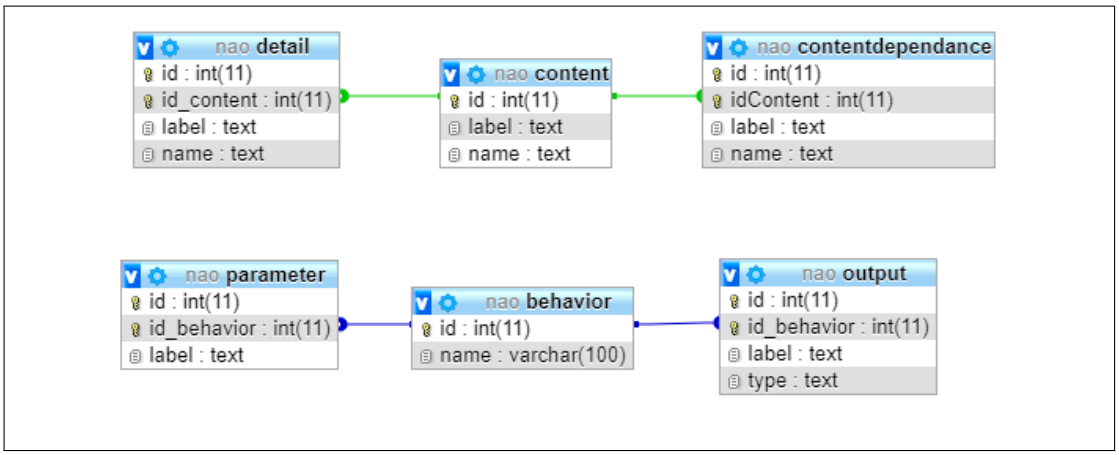


FIGURE 6.2 – Base de données utilisée lors des expériences

Nous rappelons dans la figure 6.2 l'architecture de notre base de données déjà présentée dans le chapitre 3. Elle se compose de six tables : trois concernant la gestion des contenus et trois concernant les actions exécutables sur le processus piloté.

Le scénario, que nous modélisons pour le Muséum d'Histoire Naturelle de La Rochelle, propose un jeu sérieux sous forme de quiz. L'ensemble des questions posées par le robot aux joueurs, sont présentes dans la table *Content*. Le label représente l'identifiant de la question et le champ *name* la question en elle même. La réponse à la question est présente dans la table *contentDependance*. Lorsqu'un joueur répond correctement à une question, le robot propose des explications supplémentaires à celui-ci. Ces explications sont présentes dans la table *Detail*.

Nous avons développé, pour le pilotage du robot, un ensemble de modules Python utilisant la librairie *NAOqi* fournie par le constructeur de Nao : Aldebaran. Chaque module correspond à une action particulière (déplacement, reconnaissance faciale, reconnaissance vocale, etc.). Le nom de chaque module ainsi développé est sauvegardé en base

de données dans la table *Behavior*. Ces modules comportent un ensemble de paramètres d'entrées stockés dans la table *Parameter* et des sorties contenues dans la table *Output*.

Behavior		**Parameter**		**Output**	
Id	**Name**	**Label**	**Type**	**Label**	**Type**
1	walk	gesture	text	out	boolean
		speed	int		
2	vocalRecognition	dictionary	string	wordRecognized	string
		time	int	confidence	double
3	faceDetection	time	int	faceDetected	boolean

TABLE 6.1 – Extrait de la base de données des modules

Dans l'exemple du tableau 6.1, nous présentons le module *walk* permettant au robot de marcher. Ce module nécessite deux paramètres d'entrées : un mouvement particulier du robot (gesture) et une vitesse. A la fin de son exécution, le module retourne une valeur booléenne (*out*) indiquant la réussite ou non de l'exécution de ce module sur le robot.

La base de données ainsi peuplée, le concepteur du scénario peut commencer sa modélisation en utilisant l'éditeur de CELTIC que nous présentons dans la section suivante.

6.1.2 Éditeur de CELTIC

Notre plateforme se compose d'un éditeur web, utilisant le framework *Symfony*, pour la conception de scénario interactif. Développée en partie par les étudiants de l'IUT, nous avons souhaité rendre facilement accessible notre plateforme à la communauté scientifique. Le choix de la technologie web était donc évident.

Cet éditeur se compose de deux interfaces principales :

- La première est dédiée à l'édition du scénario. Dans cette interface, le concepteur du scénario va construire son expérience pas à pas grâce à notre modélisation à deux couches ;
- La seconde est dédiée à la gestion de la base de données et notamment aux différents contenus proposés lors de l'expérience. Le développement de cette fonctionnalité n'est pas encore terminé, mais n'interfère pas dans le bon fonctionnement de la première partie (nous utilisons, en attendant, l'interface *phpMyAdmin*).

Reprenons l'exemple introduit dans le chapitre 3, dans lequel nous souhaitons modéliser le comportement d'un agent *promeneur*. La vue de l'interface de CELTIC de la couche déclarative est donnée en figure 6.3. Sur celle-ci, nous retrouvons l'ensemble des éléments nécessaires à la réalisation de notre activité : channel (canaux de communications pour la synchronisation), variables déclaratives, comportements, patterns et horloges.

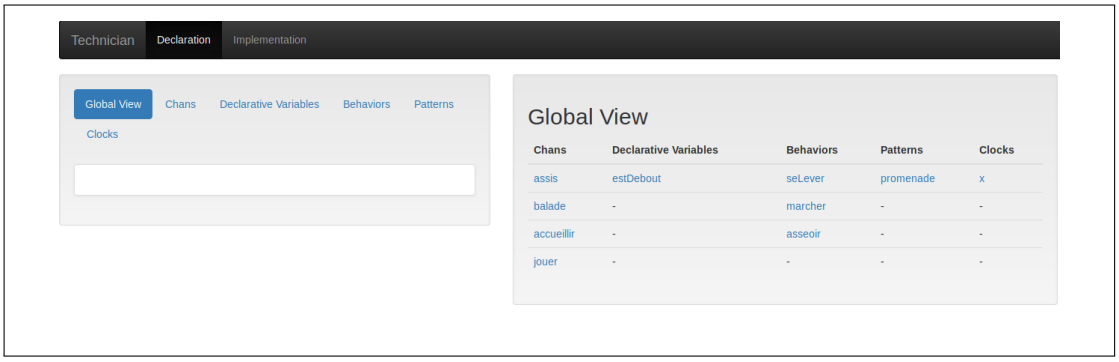


FIGURE 6.3 – Vue globale des éléments de la couche déclarative

Lorsque l'ensemble de ces éléments est défini, le concepteur peut utiliser la seconde partie de l'éditeur pour déclarer ses agents regroupant des comportements. Chaque comportement est alors spécifié afin d'établir un lien entre le modèle formel et l'application cible, à savoir dans notre cas le robot Nao. Pour cela, il dispose ici aussi d'une interface. La figure 6.4 présente l'interface où le concepteur va pouvoir créer et éditer les paramètres d'un agent.

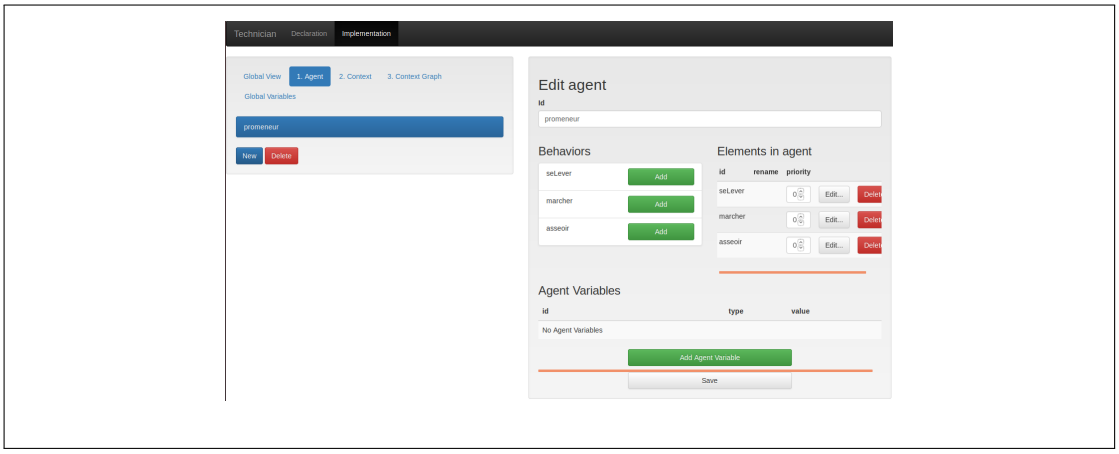


FIGURE 6.4 – Vue d'édition d'un agent

Pour spécifier les paramètres d'un comportement, le concepteur utilise la vue présentée en figure 6.5. En sélectionnant un module, l'application recherche en base de données l'ensemble des paramètres nécessaires à l'exécution. Dans l'exemple, pour le comportement *marcher*, le concepteur a sélectionné le module *walk*. Les paramètres *gesture* et *speed* sont donc à spécifier. Le concepteur peut alors utiliser deux types de variables pour valuer les paramètres. Soit une variable locale au modèle soit une valeur présente en base de données (lorsque nous souhaitons faire parler le robot, son texte peut être spécifié par l'utilisation d'une valeur présente en BDD).

La dernière étape consiste à créer et ordonner les différents contextes mettant en

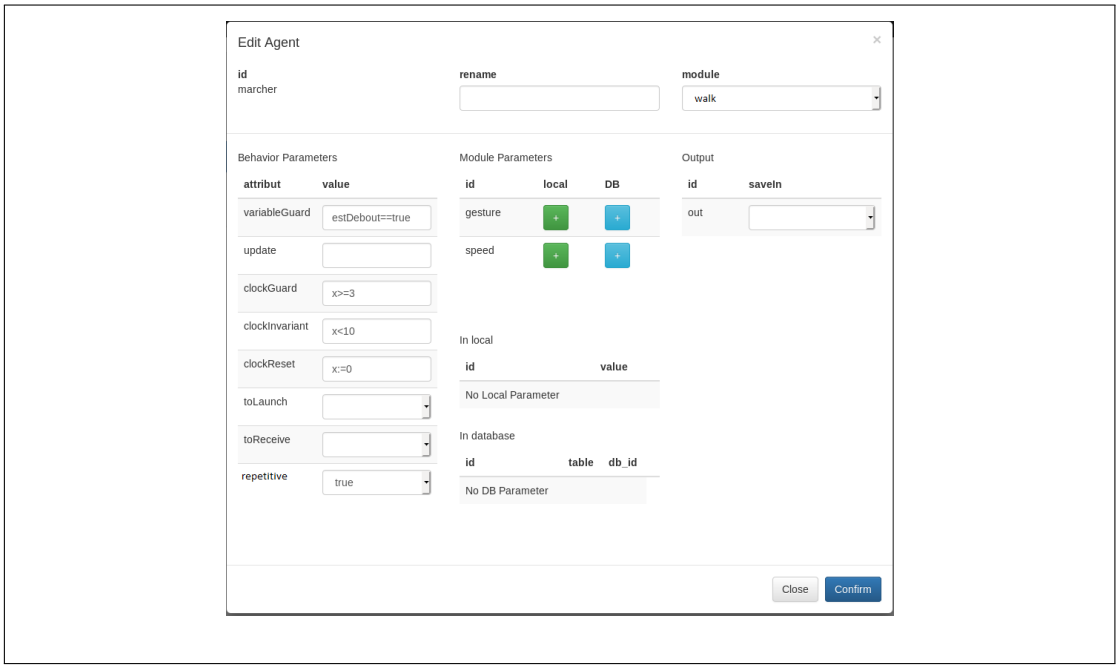


FIGURE 6.5 – Vue d'édition d'un comportement au sein d'un agent

relation des agents. L'interface dédiée à cette étape est présentée en figure 6.6. Lorsque cette dernière étape est réalisée, nous obtenons un fichier XML dans lequel l'expérience est décrite. Celui-ci sera alors utilisé pour générer les automates temporisés correspondant et le modèle au format UPPAAL.

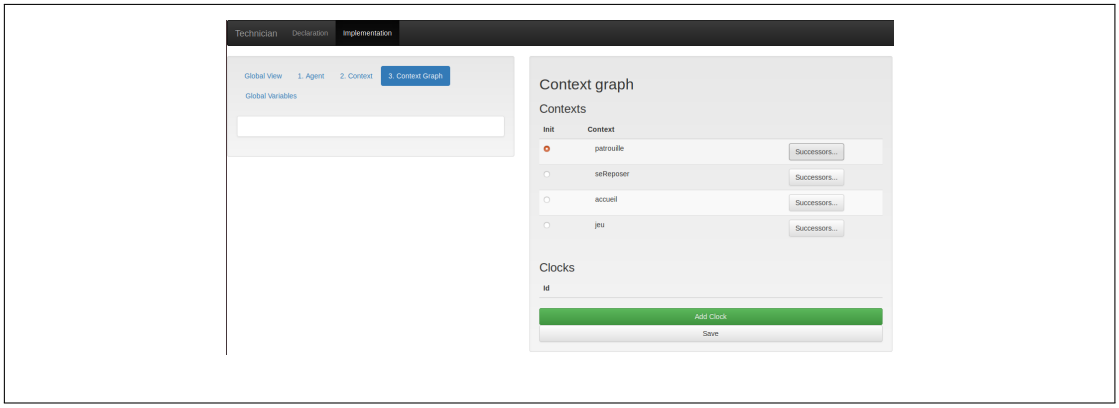


FIGURE 6.6 – Vue d'édition du graphe de contexte

6.1.3 Générateur d'automates

Le générateur d'automates de la plateforme CELTIC a été développé en JAVA. Sur la base du modèle décrit par l'éditeur et sérialisé au format XML, le générateur est en charge de générer l'automate temporisé communicant de chaque entité du système. Ces automates sont ensuite combinés pour répondre aux spécifications du concepteur grâce aux différents algorithmes de composition qu'il intègre et présentés au chapitre 3. A l'issu de la génération de chaque entité, le générateur sérialise à nouveau au format XML le scénario et produit ainsi le fichier de supervision. Ce fichier intègre l'ensemble des canaux de communication, les variables, l'automate temporisé de chaque contexte ainsi que le graphe de contextes de haut niveau.

Reprenons l'exemple de la figure 4.7. Sa représentation au format XML serait celle présentée figure 6.7. Le fichier de supervision ainsi obtenu peut alors être utilisé par notre superviseur EDAIN (un exemple de fichier de supervision est donné en annexe). L'algorithme 4, a également été implémenté en JAVA dans ce générateur. Nous pouvons ainsi produire un second fichier XML au format UPPAAL pour la vérification de certaines propriétés.

6.2 EDAIN

La supervision d'une tâche en robotique consiste à prendre les décisions adéquates en fonction des différents événements qui peuvent se produire. Lorsque les robots interagissent avec un humain, les tâches deviennent alors collaboratives et nécessitent la prise en compte de spécificités particulières. En effet, le succès de l'exécution de la tâche n'est alors plus seulement due au robot mais également à la réalisation de l'activité attendue de l'humain. Par exemple, lors d'un échange vocale entre le robot et l'humain, la tâche ne peut être réussie que si le robot est en phase d'écoute mais également si l'humain lui même décide d'interagir avec le robot en lui parlant. Il est donc nécessaire de prendre en compte toutes les issues possibles lorsque nous souhaitons superviser une activité robotique. Nous détaillons donc, dans la suite de ce chapitre, comment, à partir du modèle à base d'automates, est supervisée et pilotée l'activité dans notre approche.

EDAIN est l'outil de supervision associé au modèle CIT. Il permet d'établir une connexion sur les robots Nao et Pepper. Le mécanisme de communication permettant le dialogue entre les robots et EDAIN repose sur une communication client-serveur. Un serveur développé en Python a été embarqué sur les robots tandis qu'un client JAVA est en charge d'établir la connexion depuis l'outil EDAIN.

Après avoir renseigné les adresses de la base de données et du robot ainsi qu'un fichier où seront sauvegardés tous les logs, l'interface du superviseur EDAIN demande à l'utilisateur de charger le fichier de supervision produit. Les différents automates sont alors représentés graphiquement et l'outil charge le contexte initial (figure 6.8). L'ensemble des variables et horloges est alors affiché ainsi que l'automate du contexte actuel. EDAIN va alors exécuter, parmi les transitions possibles, celle répondant aux contraintes de temps et de variables, selon l'algorithme d'analyse d'accessibilité décrit au chapitre 4.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE supervision SYSTEM "supervision.dtd">
<supervision id="Nao">
  <chan id="jouer"/>
  <chan id="assis"/>
  <chan id="bouge"/>
  <chan id="accueillir"/>
  <variable id="wordRecognized" type="string" value=""/>
  <context id="accueil">
    <automaton id="auto_accueil">
      <state id="S0" isInitial="true" isFinal="false" invariant=""/>
      <state id="S1" isInitial="false" isFinal="false" invariant=""/>
      <state id="S2" isInitial="false" isFinal="true" invariant=""/>
      <state id="S3" isInitial="false" isFinal="true" invariant=""/>
      <transition id="presentation" from="S0" to="S1" guard="" update=""
          clockGuard="" reset="" synchronization="">
        <module id="2">
          <parameter id="gesture" table="Behavior" id_bdd="21" saveIn=""/>
          <parameter id="speed" value="90"/>
          <parameter id="text" table="Content" id_bdd="1" saveIn=""/>
        </module>
      </transition>
      <transition id="demanderPrenom" from="S1" to="S2" guard="" update=""
          clockGuard="" reset="" synchronization="">
        <module id="3">
          <parameter id="gesture" table="Behavior" id_bdd="21" saveIn=""/>
          <parameter id="speed" value="90"/>
          <parameter id="text" table="Content" id_bdd="1" saveIn=""/>
          <parameter id="dictionnary" value="damien,armelle,arnaud"/>
        </module>
      </transition>
      <transition id="direBonjour" from="S2" to="S3" guard="wordRecognized!="""
          update="" clockGuard="" reset="" synchronization="jouer!">
        <module id="2">
          <parameter id="gesture" table="Behavior" id_bdd="21" saveIn=""/>
          <parameter id="speed" value="90"/>
          <parameter id="text" table="Content" id_bdd="3" saveIn=""/>
        </module>
      </transition>
      <transition id="erreur" from="S2" to="S1" guard="wordRecognized==""" update=
          "" clockGuard="" reset="" synchronization=""/>
    </automaton>
  </context>
  ...
```

FIGURE 6.7 – XML de supervision du contexte Accueil décrit au chapitre 3

Lorsqu'une action doit être exécutée sur le robot, EDAIN récupère en base de données le nom du module à exécuter et éventuellement les valeurs des contenus à proposer à l'utilisateur et va se connecter au serveur du robot pour lui transmettre les paramètres de pilotage. L'échange d'information se réalise au travers d'un socket et utilise le format JSon. Par exemple, lors de l'exécution du module de reconnaissance vocale, EDAIN envoie l'information ci dessous au robot :

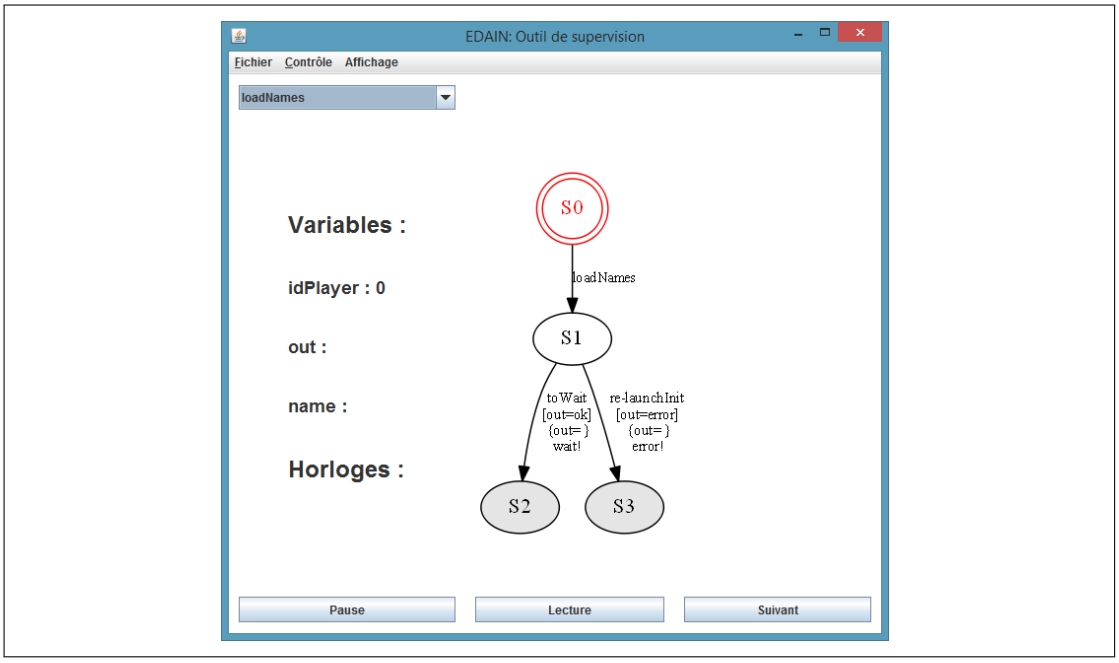


FIGURE 6.8 – Interface du superviseur EDAIN

{"speechReco" :{"parameters" :{"time" :"15","dictionnary" :"oui,non"}}}

Du côté du robot, à la réception de ces informations, le système va exécuter le script *speechReco.py* en lui passant les paramètres associés. A l'issue de l'exécution de ce script, le robot renverra à EDAIN le mot reconnu ainsi qu'un taux de confiance associé dans le même format :

{"out" :{"wordRecognized" :"oui","confidence" :"0.3216"}}

Grâce aux informations reçues, EDAIN met à jour le vecteur d'états des variables du modèle et peut ainsi choisir la prochaine transition à exécuter. Lorsqu'une synchronisation entre contexte est réalisée, le contexte cible est chargé sur l'interface et le même procédé est reproduit.

6.3 Conclusion

Nous avons présenté dans ce chapitre les deux logiciels développés pendant ces travaux de recherche : CELTIC et EDAIN. Ces derniers constituent l'implémentation des

différents concepts adressés dans cette thèse et permettent d'aboutir à une chaîne complète allant de la conception à la supervision de scénario interactifs déployés sur des robots.

Nous présentons dans le chapitre suivant les expérimentations réalisées grâce à notre plateforme dans les deux musées partenaires ainsi que les résultats associés.

Chapitre 7

Expérimentations face au public et résultats

Sommaire

Dans ce chapitre, nous allons présenter les différentes expérimentations que nous avons menées au Muséum d'Histoire Naturelle de La Rochelle et au Musée Sainte-Croix de Poitiers. Ces expérimentations nous ont permis d'éprouver la méthode de modélisation proposée et de vérifier la généricité de représentation d'une expérience interactive, la réutilisabilité des éléments de la couche déclarative d'une expérience à une autre et la facilité avec laquelle les contenus sont décorrelés du modèle (au travers de l'architecture basée sur la base de données présentée à la figure 6.2 du chapitre précédent).

La première expérimentation a eu lieu lors de la fête de la science 2017 au muséum d'Histoire Naturelle de La Rochelle. Le scénario créé pour l'occasion avait pour objectif

FIGURE 7.1 – Expériences au Muséum d'Histoire Naturelle de La Rochelle

de faire découvrir certaines œuvres exposées, sous forme de quiz proposés par les robots Nao (figure 7.1).

La seconde expérimentation, dédiée au musée Sainte Croix de Poitiers, s'est déroulée lors de la Gamers Assembly 2018. Nous proposions, à cette occasion, un nouveau scénario interactif, faisant intervenir les robots Nao et des jeux développés sur tablettes.

Nous détaillons dans les sections 7.1 et 7.2 ces deux expérimentations.

7.1 Muséum d'Histoire Naturelle de La Rochelle

La première expérimentation que nous avons réalisée s'est déroulée au Muséum d'Histoire Naturelle de La Rochelle, pendant trois après midi, lors de la fête de la Science. L'objectif était de présenter trois oeuvres exposées au public sous forme de quiz. A chaque question, le joueur était invité à parcourir le musée pour trouver la réponse et revenir vers le robot. Les joueurs pouvaient interagir avec Nao à n'importe quel moment, par simple reconnaissance visuelle du badge qu'ils portaient. Nao continuait alors la séquence du jeu en interrogeant sa mémoire interne.

Nous avons proposé deux expérimentations, basées sur le même modèle (seuls les contenus textuels en base de données ont été modifiés) :

- la première, destinée aux enfants leur faisait découvrir l'exposition consacrée aux monstres, imaginaires ou réels ;
- la seconde, destinée aux adolescents et adultes, proposait de découvrir l'exposition permanente consacrée à l'archéologie.

Nous détaillons l'ensemble de l'expérimentation dans la section suivante.

7.1.1 Présentation du scénario

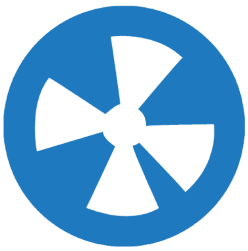


FIGURE 7.2 – Exemple de NaoMark utilisée pour identifier les joueurs

L'expérimentation, présentée sur la figure 7.3, débutait par la mise en attente de Nao. Lorsqu'un joueur voulait interagir avec lui, un contact tactile, sur le capteur situé sur la tête de Nao, lui permettait de s'éveiller. Nao est capable de détecter la présence de personnes grâce à ses caméras. Mais, nous n'avons pas souhaité utiliser cette fonctionnalité du fait du grand nombre de visiteurs pouvant passer devant lui. Après avoir déclenché l'exécution du scénario, le visiteur était invité à présenter son badge à Nao pour s'identifier. Cette procédure était à renouveler à chacune des phases du jeu. Au début de la partie, Nao ne connaissait pas le joueur et lui présentait donc le déroulement de la partie. Il enregistrait alors en mémoire l'identifiant de son badge ainsi que l'identifiant de la question à laquelle il devait répondre (dans ce cas, la première). Lorsque Nao reconnaissait le joueur, il répétait la question du quiz et attendait la réponse. Un indice était donné au joueur en cas de mauvaise réponse. Dans le cas contraire, des détails sur l'oeuvre étaient énoncés et la question suivante était posée, avec sauvegarde en mémoire. A la fin du quiz, le joueur obtenait son temps de jeu par Nao ainsi que son classement provisoire.

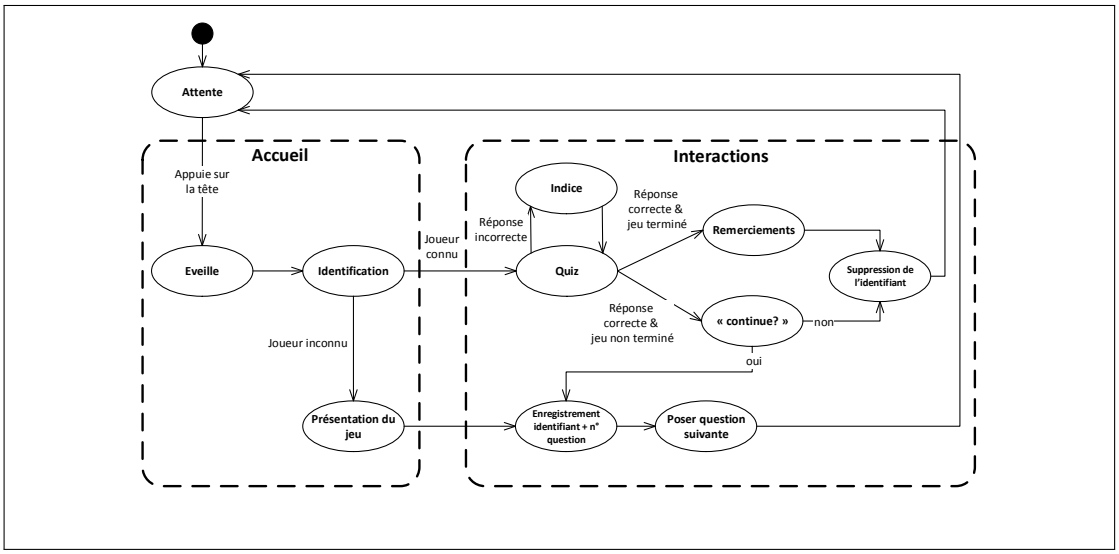


FIGURE 7.3 – Scénario de l'expérience interactive

Ce scénario a été modélisé par notre outil **CELTIC** et est composé de huit contextes, comme indiqué sur la figure 7.4 :

- **Init** : le robot effaçait les anciennes parties de sa mémoire s'il y en avait. En cas d'erreur, il recommençait ;

- **Attente** : le robot se mettait en attente d'un joueur, qui lui appuyait sur la tête pour lancer le début du jeu. Il y avait un bouclage sur ce contexte toutes les quinze secondes, en cas d'inactivité ;
- **Accueil** : dans ce contexte, le robot accueillait le joueur par un petit mot de bienvenue et se présentait ;
- **Reconnaissance** : après avoir été accueilli par Nao, le joueur devait s'identifier en présentant, au robot, le badge qu'il avait précédemment récupéré à l'accueil du musée. Sur celui-ci, un NaoMark (figure 7.2) unique lui permettait de s'identifier (chaque NaoMark possède un identifiant, représenté par un entier). Des tests de reconnaissance faciale ont été réalisés, mais l'absence d'une bonne luminosité dans le musée nous a contraint à abandonner cette approche ;
- **Nouveau** : lorsque Nao ne reconnaissait pas le joueur, le contexte *Nouveau* était exécuté. Celui-ci permettait d'expliquer le jeu et son déroulement. Si aucun NaoMark n'était détecté, le contexte était exécuté une nouvelle fois ;
- **Ancien** : dans ce contexte, Nao avait reconnu le joueur. Il lui reposait alors la question à laquelle il devait répondre et attendait la réponse. Si la réponse n'était pas correcte ou si un problème de reconnaissance vocale se produisait, le contexte était à nouveau exécuté. Dans le cas d'une réponse correcte, le robot donnait des détails supplémentaires sur l'oeuvre répondant à l'énigme. Enfin, si le joueur répondait à la dernière question du quiz, le robot le félicitait, effaçait de sa mémoire l'identifiant du joueur (le badge pouvait alors servir à un nouveau joueur) et retournait en *Attente*. Dans le cas contraire, le couple <identifiant du joueur, numéro de la question suivante> était sauvegardé dans la mémoire du robot ;
- **Jouer** : cette étape consistait à demander au joueur s'il voulait continuer ou non la partie. Si oui, le contexte *PoserQuestion* était exécuté sinon, le robot se remettait en *Attente* ;
- **PoserQuestion** : une nouvelle question était posée au joueur puis le contexte *Attente* était à nouveau chargé.

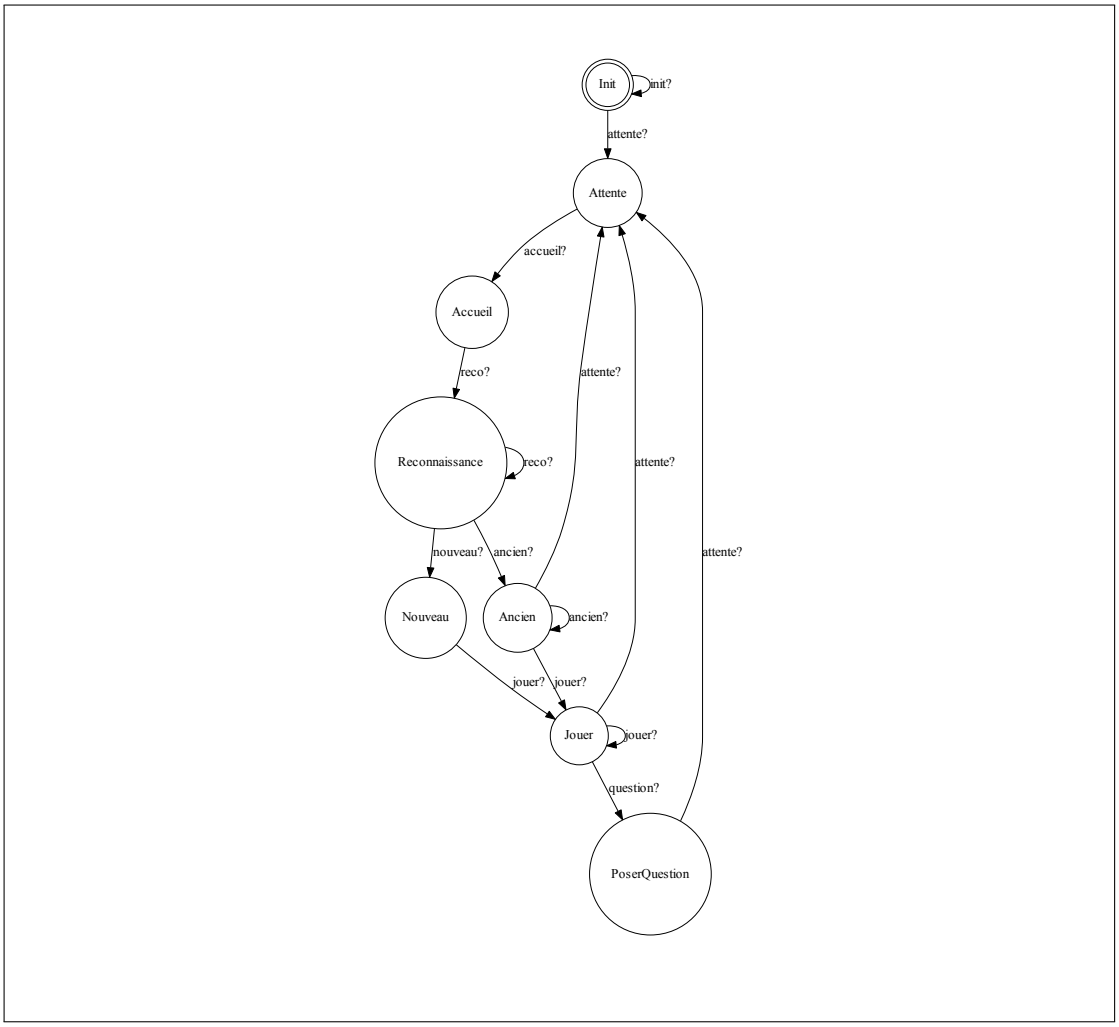


FIGURE 7.4 – Graphe de contexte des expériences du Muséum d'Histoire Naturelle de La Rochelle

7.1.2 Limites de l'expérimentation

FIGURE 7.5 – Expérimentation au Muséum d'Histoire Naturelle

Lors de ces trois jours d'expérimentations, nous avons rencontré des problèmes à la fois sur le plan technique et sur le plan de la conception même du scénario. Du fait de la configuration du Muséum, il était impossible d'isoler Nao dans une pièce. Cet inconvénient a eu pour conséquences des difficultés de reconnaissance vocale. En effet, Nao était placé au milieu des oeuvres dans un bruit ambiant élevé. La qualité de l'interaction entre le joueur et le robot a été impactée. De plus, l'exécution du scénario est dépendante

de certains paramètres contrôlant l'activité. La valuation de ces paramètres (de temps ou seuils de reconnaissance vocale) est réalisée par le concepteur et peut, dans certains cas, ne pas être adaptée à l'environnement ou au joueur.

Notre outil de supervision **EDAIN**, présenté dans la section 6.2 du chapitre 6, intègre un système d'observation des comportements de l'utilisateur et du scénario, afin de générer des traces d'exécution. Les logs obtenus lors de ces expérimentations font apparaître les problématiques que nous avons présentées précédemment. Chaque événement produit par l'outil de supervision était sauvegardé dans un fichier plat ainsi que l'ensemble des retours issus de Nao (Figure 7.6) Pour l'expérimentation réalisée au Muséum de La Rochelle, nous avons obtenu 6500 lignes de logs.

```
31 mar. 2018 14:50:47; joueur:4 nouveauJoueur; parameters: [gesture: animations/
    Stand/BodyTalk/Speaking/BodyTalk_11, speed: 90, text: Je ne pense pas te
    connaitre. Afin de mieux faire connaissance, peux tu me donner ton prenom.
    Attends bien le petit son que je vais emettre avant de me repondre.]
31 mar. 2018 14:51:02; joueur:4 nouveauJoueurBack; parameters: [out: ok]
31 mar. 2018 14:51:03; joueur:4 ecouterPrenom; parameters: [time: 15]
31 mar. 2018 14:51:12; joueur:4 ecouterPrenomBack; parameters: [confidence:0
    .500299990177, name:theophile]
31 mar. 2018 14:51:13; joueur:4 donnerPrenomNouveau; parameters: [gesture:
    animations/Stand/BodyTalk/Speaking/BodyTalk_11, speed: 90, text: theophile]
31 mar. 2018 14:51:18; joueur:4 donnerPrenomNouveauBack; parameters: [out: ok]
31 mar. 2018 14:51:19; joueur:4 sauvegarderPrenom; parameters: [id: 4, name:
    theophile]
31 mar. 2018 14:51:20; joueur:4 sauvegarderPrenomBack; parameters: [out: ok]
```

FIGURE 7.6 – Exemple de traces d'exécution enregistrées lors des interactions entre un joueur et Nao

Nous pouvons ainsi, a posteriori, reconstruire les automates temporisés de chaque joueur. Les problèmes liés à la reconnaissance vocale apparaissent alors par de nombreuses boucles (exemple figure 8.2 en annexes).

7.2 Musée Sainte-Croix de Poitiers

La ville de Poitiers organise chaque année, depuis 2014, la Gamers Assembly. Ce festival de jeux vidéo voit défiler les meilleures équipes au monde de sports électroniques. A cette occasion, des *offs* du festival sont organisés par la ville. Dans ce cadre et en collaboration avec le musée Sainte-Croix, nous avons organisé une session de jeu permettant au public de venir découvrir le musée de manière ludique [1], le 31 mars 2018.

Le jeu se déroule en quatre étapes, comme indiqué sur la figure 7.7.

1. http://l3i.univ-larochelle.fr/Nao-a-la-Gamers-Assembly-de

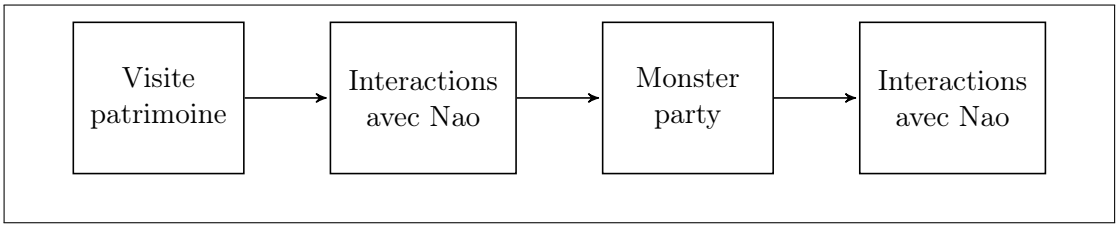


FIGURE 7.7 – Étapes du jeu réalisé au musée Sainte Croix

7.2.1 Visite patrimoine

La première partie du jeu se déroule à l'extérieur du musée, en utilisant l'application *Visite patrimoine Poitiers*. Cette application, développée par des étudiants de La Rochelle Université, propose de découvrir certains monuments de la ville. Pour notre expérience, une version personnalisée de l'application a été développée par des étudiants de l'IUT de La Rochelle.

Les joueurs se sont vus proposer un parcours à l'extérieur du musée, leur faisant découvrir des monuments comme la cathédrale Saint Pierre, le Baptistère et l'extérieur du musée. A chaque étape du parcours, une énigme leur était posée dont la réponse leur servait d'indice. A l'issu du parcours extérieur, les joueurs devaient reformer un mot code à partir des différents indices récupérés et ainsi passer à la deuxième étape du jeu. Le mot code ainsi obtenu, les joueurs se dirigeaient vers le musée où Nao les attendait.

FIGURE 7.8 – Badge identifiant les joueurs

7.2.2 Lancement du jeu : premières interactions avec Nao

Nao était chargé d'accueillir les joueurs au musée. L'objectif de ces premières interactions étaient de vérifier le mot code des joueurs ainsi que de leur expliquer le déroulement de la seconde partie du jeu. Le détail de cette partie du scénario est présenté sur la figure 7.9.

La mécanique d'identification du joueur est identique à celle utilisée lors des expériences au Muséum. Le badge, semblable à celui présenté sur la figure 7.8, permettait également de suivre le parcours du joueur dans le musée. Lorsqu'aucun joueur n'est présent devant Nao, il se met en attente et s'assoit. La première action à effectuer est donc de lui appuyer sur la tête pour l'éveiller et lui permettre d'accueillir le visiteur. Nao demandait alors au joueur de présenter son badge (Figure 7.10 [2]). Chaque badge ayant un *NaoMark* unique, il était alors possible de déterminer quel joueur était en train d'interagir avec Nao et à quelle étape du jeu il en était.

2. Crédit photo : ©Musées de Poitiers / S. Coussay

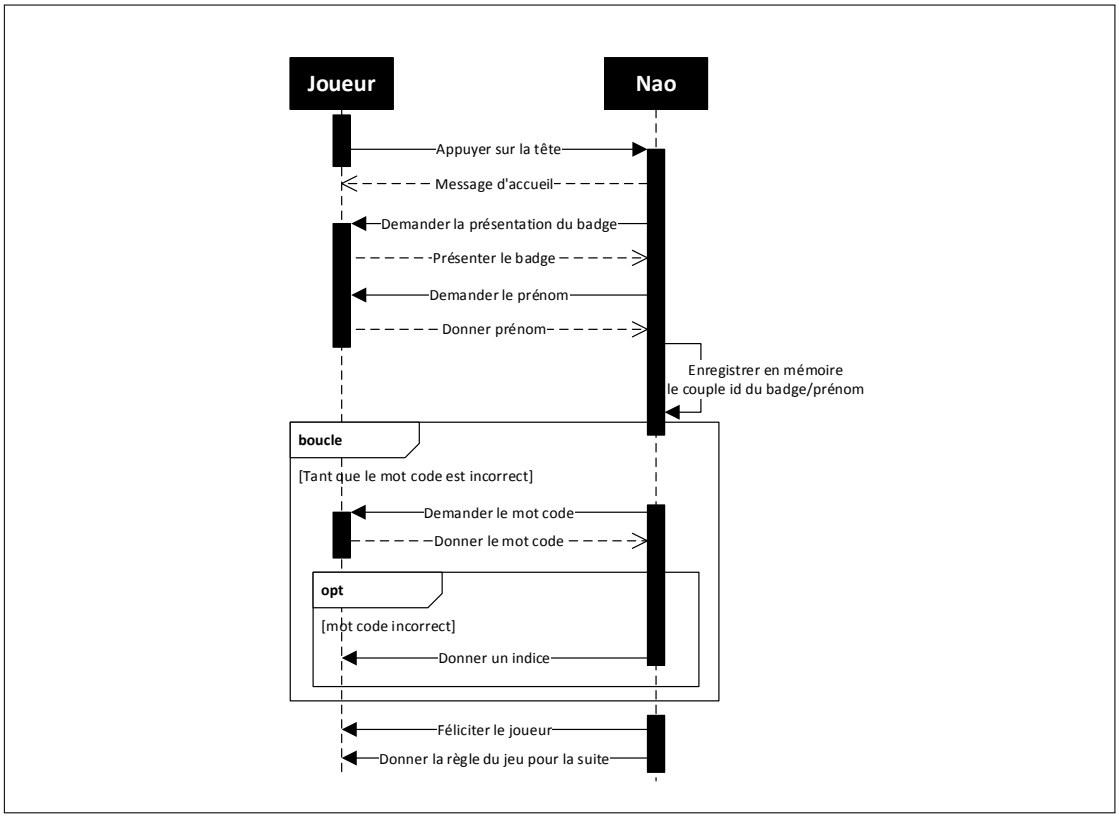


FIGURE 7.9 – Premières interactions entre le joueur et Nao

Contrairement à l'expérimentation effectuée au Muséum de La Rochelle, et dans la mesure ou un nombre trop important de joueurs pouvaient diminuer la qualité de l'expérience, une gestion des flux a été mise en place. En effet, dans le cadre de l'expérimentation du musée Sainte-Croix, une inscription préalable auprès des services de la mairie était nécessaire. Sur la base, de la liste des joueurs ainsi obtenue, nous avons donc ajouté une interaction, au cours de laquelle Nao demandait le prénom du joueur. Le dictionnaire des prénoms a permis une référence symbolique vers le prénom ayant le taux de confiance le plus élevé.

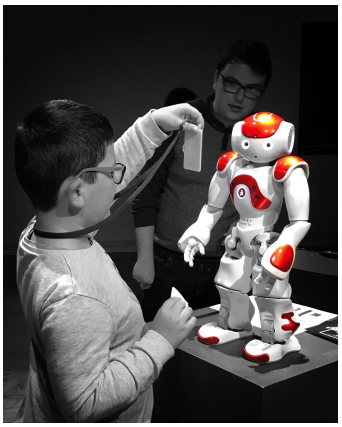


FIGURE 7.10 – Un joueur s'identifiant auprès de Nao

A l'issu de cette interaction, une sauvegarde du couple identifiant du badge/prénom était effectuée en mémoire de Nao. Cette information sera utilisable dans la suite du scénario.

Enfin, après cette sauvegarde, Nao vérifiait le mot code proposé par le joueur. Si le mot code était incorrect, le robot donnait un nouvel indice jusqu'à obtenir la bonne réponse. Lorsque la réponse était correcte, la seconde partie du jeu était expliquée au joueur. La *monster party* pouvait donc débuter.

7.2.3 Monster party

Ce jeu sur tablette, développé par des étudiants de l'IUT, avait pour objectif de faire découvrir l'étage consacré à l'archéologie. Pour ce faire, le jeu proposait de collecter des armes pour capturer des monstres, associés à des oeuvres. Grâce à la technologie de réalité augmentée, les joueurs devaient retrouver 3 armes et capturer 3 monstres (Figure 7.11). A chaque monstre capturé, un indice leur était proposé afin de déterminer le mot code final. A chaque événement produit dans le jeu (découverte d'un outil, capture d'un monstre ou récupération d'indice), une sauvegarde sur un serveur était réalisée. Dans le cadre d'un projet de recherche sur l'analyse des déplacements des joueurs [BFB18], chaque badge porté par les joueurs permettait d'enregistrer son parcours dans le musée, grâce à la technologie des iBeacons. Des Raspberry (mini ordinateur) ont été déployés sur l'étage où se déroulait ce jeu et sur lesquels les badges iBeacons se connectaient lorsqu'ils se trouvaient à proximité. Ainsi, un timestamp et l'identifiant du Raspberry étaient enregistrés sur un serveur afin d'identifier le parcours du joueur. A la fin de ce jeu, le joueur était invité à se rendre devant Nao pour vérifier le mot code final qu'il avait obtenu grâce aux différents indices récupérés.

FIGURE 7.11 – Extrait du jeu *Monster party*

7.2.4 Fin du jeu : dernières interactions avec Nao

La dernière étape du scénario est présentée sur la figure 7.13. Pour terminer la partie, le joueur devait retourner auprès de Nao pour lui donner le mot code final. Après s'être identifié avec leur badge, Nao récupérait sur le serveur l'ensemble des indices obtenus

par le joueur, via son identifiant. Ces indices ainsi récupérés étaient redonnés au joueur. Celui-ci propose alors le mot code final au robot.

FIGURE 7.12 – Deux joueurs en fin de partie

Si la réponse est incorrecte, le robot donne un indice supplémentaire jusqu'à ce que le joueur donne la bonne réponse. Enfin, lorsque la bonne réponse était proposée, Nao invitait les joueurs à se diriger vers l'accueil pour obtenir un lot (Figure 7.12 [3]).

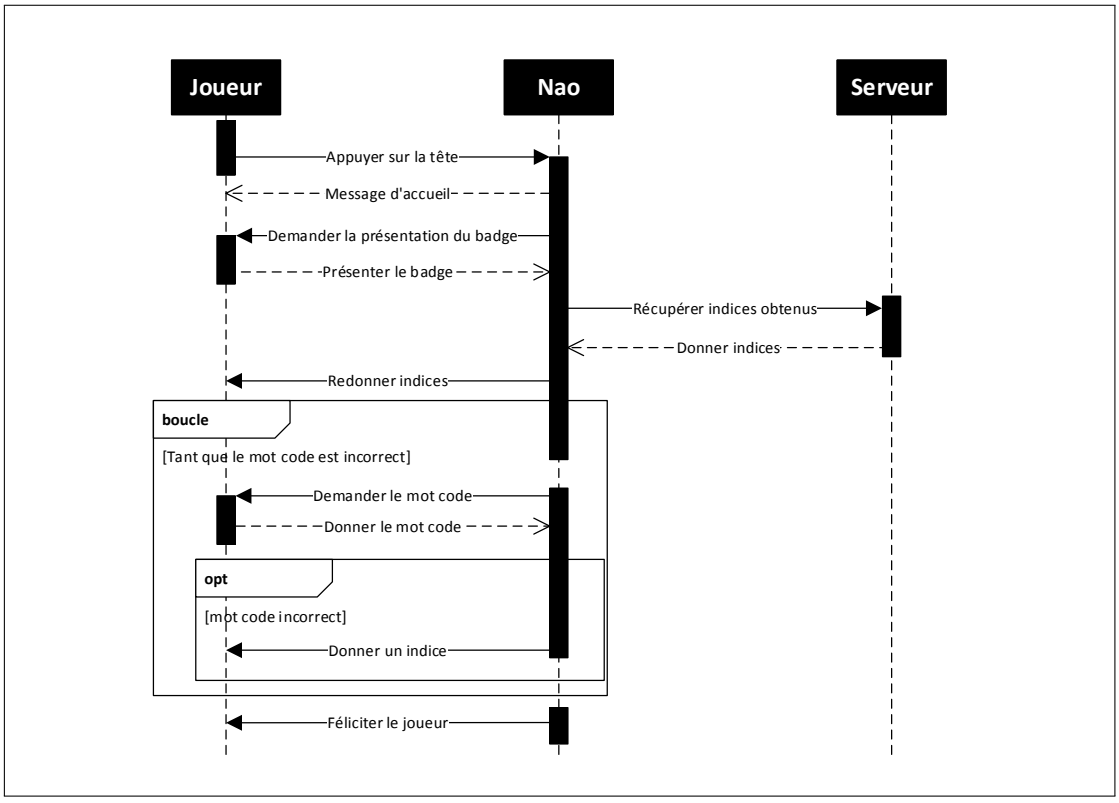


FIGURE 7.13 – Dernières interactions entre le joueur et Nao

3. Crédit photo : ©Musées de Poitiers / S. Coussay

7.2.5 Résultats

Pour pallier les difficultés rencontrées lors de l'expérimentation effectuée au Muséum de La Rochelle, liées au flux important de joueurs, des inscriptions ont été mises en place par les services de la mairie de Poitiers. Ainsi, nous avions un contrôle du flux de joueurs voulant interagir avec Nao, qui pour cette occasion, était isolé du reste de l'exposition. Au total, nous avons comptabilisé 13 joueurs (ou équipes). Bien que le nombre de participants fût faible, cette expérimentation nous a permis d'étendre et de proposer un jeu sérieux plus riche que celui proposé au Muséum de La Rochelle. En effet, dans cette expérience, nous avons couplé trois jeux, sur tablettes et sur le robot Nao. Contrairement à l'expérimentation de La Rochelle, aucun joueur n'a abandonné la partie. Des erreurs de reconnaissance vocale persistent cependant, mais leur nombre a été réduit par rapport à l'expérimentation de La Rochelle. Avoir isolé Nao et contrôlé le flux des joueurs a permis de limiter le bruit ambiant, améliorant ainsi la qualité de l'interaction. Un questionnaire évaluant la qualité du jeu a été proposé à chacun des joueurs à l'issue de leur partie. Les résultats sont présentés sur les figures 7.14, 7.15, 7.16, 7.17 et 7.1.

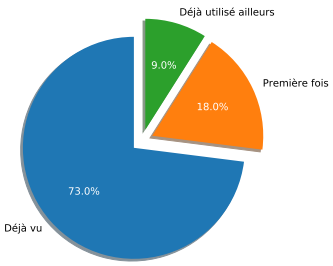


FIGURE 7.14 – Connaissiez-vous Nao avant le début du jeu ?

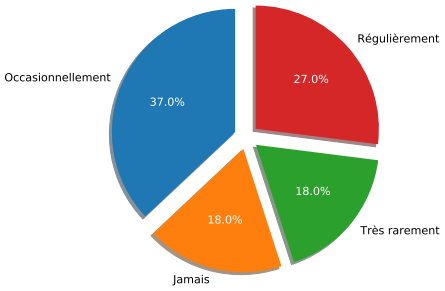


FIGURE 7.15 – Etiez-vous déjà venu au musée Sainte Croix ?

	Simple	Ennuyeuse	Originale	Positive
Tout à fait d'accord	9%	0	82%	64%
D'accord	28%	0	9	36%
Pas d'accord	45%	18	0	0
Pas du tout d'accord	0	82%	0	0
Sans opinion	18%	0	9%	0
Total	100%	100%	100%	100%

TABLE 7.1 – Comment qualifieriez-vous cette expérience ?

Cette expérience a été qualifiée d'originale et de positive par une grande majorité des joueurs mais reste complexe à appréhender pour les plus jeunes, notamment avec le

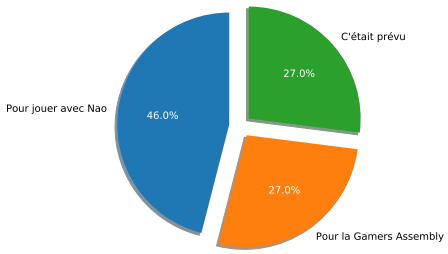


FIGURE 7.16 – Pourquoi êtes-vous venu ?

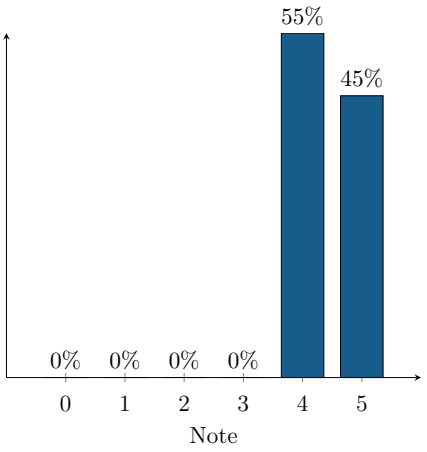


FIGURE 7.17 – Quel note donneriez-vous à cette expérience ?

jeu en réalité augmentée sur tablette. Cependant, ces résultats montrent également que les joueurs sont satisfaits de leur expérience au musée Sainte-Croix. Plus du tiers des joueurs n'ont pas pour habitude de visiter des musées et près de la moitié sont venus pour jouer avec Nao. Bien que ces résultats ne portent que sur 13 joueurs ou équipes, ils démontrent que l'usage des nouvelles technologies peut inciter le jeune public à découvrir le patrimoine exposé dans les lieux de culture.

7.3 Analyse des logs

A l'issue des deux expériences, nous avons souhaité utiliser les traces d'exécution obtenues pour analyser le comportement de l'utilisateur et observer son évolution dans le jeu. En effet, lors de la conception d'une expérience interactive, il est difficile pour le concepteur de prévoir le comportement réel de l'utilisateur final.

Nous disposons de 4000 traces représentant les parties de 46 joueurs pour l'expérience de La Rochelle et de 2200 traces pour Poitiers représentant les parties de 13 joueurs. Pour chacun des joueurs, nous reconstruisons donc le graphe modélisant leur partie. Nous présentons, en annexes, sur les figures 8.1, 8.2 et 8.3, un exemple de graphe d'une partie s'étant déroulée sans problème, un exemple de graphe d'une partie présentant des problèmes de reconnaissance vocale et la dernière d'un joueur ayant abandonné sa partie.

7.3.1 Muséum d'Histoire Naturelle

Victime de son succès, le robot Nao a suscité un grand intérêt, provoquant des cas d'utilisation non désirés. Si la première après midi, consacré à une classe de collégiens de St Savinien s'est déroulée dans de bonnes conditions (7 groupes de 2 ou 3 élèves), les phases de jeux ouvertes au grand public n'ont pas permis d'aboutir à une expérience

Partie terminée	Problème de reco. vocale	Erreur d'identification	Abandon
18	13	3	25

TABLE 7.2 – Vérité terrain de l'expérience réalisée au Muséum de La Rochelle

de jeu optimale pour les joueurs. En effet, si les deux dernières après midi ont permis à 36 joueurs de venir découvrir le Muséum et jouer avec Nao, seulement 42% d'entre eux ont pu terminer leur partie. Durant les différentes parties, nous pouvons remarquer grâce aux graphes de chaque joueur que beaucoup d'erreurs de reconnaissance vocale ont eu lieu. Ce nombre élevé a peut-être conduit certains joueurs à abandonner en cours de partie. Deuxième explication, le nombre de joueurs simultanés a fait augmenter le temps d'attente pour jouer avec Nao. Certains parents n'ont pas voulu rester davantage de temps pour terminer la partie. Ce phénomène a été particulièrement noté lors des sessions du dimanche. Les erreurs de reconnaissance vocale sont dues, en partie, au bruit ambiant présent dans le musée et à la résonance de la pièce. Le seuil de reconnaissance était de 40% pour les expériences de vendredi et samedi. Ce seuil a été abaissé à 30% pour les sessions du dimanche, mais n'a pas permis d'augmenter significativement la qualité de la partie.

7.3.2 Musée Sainte Croix de Poitiers

Partie terminée	Problème de reco. vocale	Erreur d'identification	Abandon
13	5	0	0

TABLE 7.3 – Vérité terrain de l'expérience réalisée au musée Sainte Croix de Poitiers

Pour pallier les difficultés rencontrées lors des expérimentations du Muséum de La Rochelle (bruit ambiant et nombreux joueurs), le robot Nao a été placé dans une pièce qui lui était dédiée. Ainsi, le bruit ambiant de la pièce était plus faible. La seconde modification au Musée Sainte-Croix concerne la mise en place des inscriptions pour chaque participant. En effet, cette session a été couplée à une session de jeu en réalité augmentée sur tablette. Le nombre de joueurs a été limité par le nombre de tablettes. Grâce à cela et comme le montre les résultats présentés dans le tableau 7.3, les 13 parties ont pu être terminées. Trois interactions vocales par joueur étaient prévues dans le scénario (pour donner son prénom, le mot code de début de partie et celui de fin de partie) soit 39 sur l'ensemble des parties. Dans seulement 5 cas, l'interaction ne s'est pas correctement déroulée (12.8% des cas, pour lesquels soit les joueurs ne parlaient pas assez fort, soit ne répondaient pas dans le temps imparti).

7.3.3 Optimisation des paramètres

Nous avons présenté dans le chapitre 5 notre algorithme d'apprentissage par renforcement, utilisant l'approche du *Q-Learning* pour adapter les paramètres de temps et les différents paramètres contrôlant la reconnaissance vocale. Nous utilisons cet algorithme sur les traces récoltées lors de l'expérimentation réalisée au Musée d'Histoire Naturelle de La Rochelle. L'intérêt d'utiliser ce jeu de données repose principalement sur le fait qu'il contient beaucoup d'exemples de cas d'erreurs susceptibles d'être évités par l'utilisation de notre technique de renforcement. Sur la base de ces logs, nous avons entraîné notre modèle pour déterminer les différents Q-valeur associées à chaque couple (localité, transition).

Cette phase d'entraînement s'est déroulée selon le protocole suivant :

- 1 : reconstruction des automates représentant les parties de chaque joueur ;
- 2 : à partir de l'automate des possibles, la valuation des récompenses associée à chaque transition (par défaut, les récompenses sont nulles. Seules les transitions d'erreurs ont une récompense négative (-1) et les transitions de succès (bonne reconnaissance vocale par exemple) ont une récompense positive (+1)) ;
- 3 : par tirage au sort, application du *Q-Learning* sur l'automate (un million de fois).

Nous rappelons la formule du *Q-Learning* utilisée :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (7.1)$$

avec $\alpha = 0.03$ et $\gamma = 0.9$.

La valuation obtenue a été utilisée dans un second temps afin de tester l'adaptation des paramètres au cours de l'exécution. Nous n'avons pas pu refaire d'expérience au musée pour tester cette adaptation dynamique. Nous avons donc sollicité nos étudiants de Master ICONE pour cette dernière phase de tests. Pour cela, nous avons extrait une partie du scénario du Muséum de La Rochelle où nous avons demandé à dix étudiants d'interagir avec le robot Nao vocalement. Dans ce petit scénario, Nao prononçait un mot et demandait au testeur de le répéter dans un temps donné. Trois possibilités s'offraient alors aux testeurs :

- le testeur pouvait répéter le mot prononcé par Nao ;
- le testeur pouvait dire autre chose ;
- le testeur pouvait ne pas répondre.

Nous avons appliqué pendant l'exécution de ce test, notre algorithme d'adaptation sur le paramètre contrôlant l'exécution de la reconnaissance vocale (adaptation du paramètre

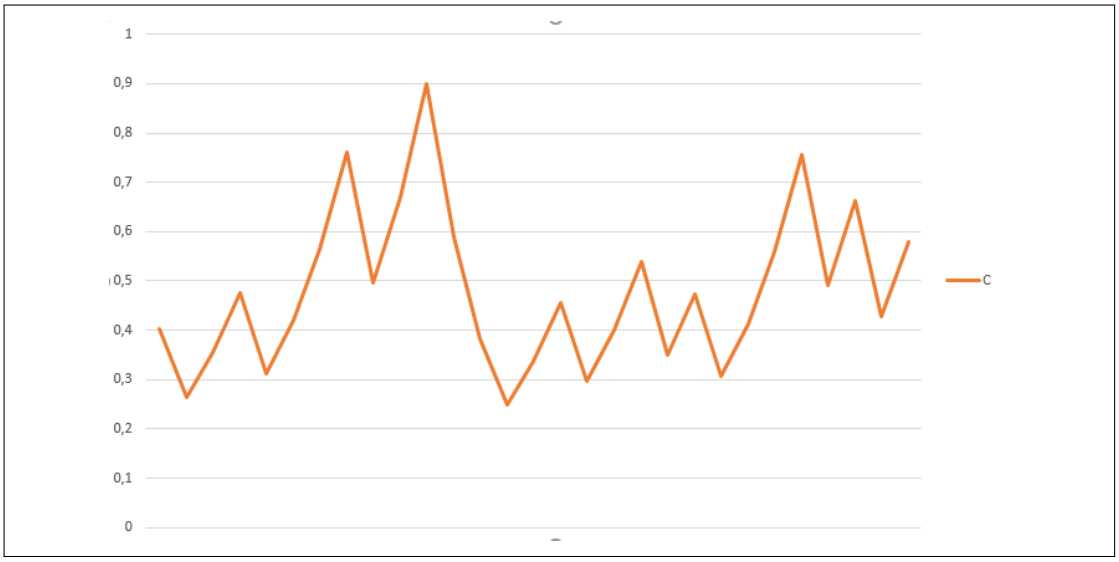


FIGURE 7.18 – Évolution des paramètres de temps et de seuils de reconnaissance

C). L'adaptation des paramètres contrôlant la reconnaissance vocale permet d'obtenir de bons résultats. De prochains tests seront à mener pour vérifier que notre approche permet d'adapter les paramètres de temps. Les résultats de cette expérimentation sont donnés en figure 7.18.

7.4 Conclusion

Nous avons présenté dans ce chapitre les deux expérimentations effectuées dans les musées partenaires :

- le Muséum d'Histoire Naturelle de La Rochelle ;
- le musée Sainte Croix de Poitiers.

Nous avons, à travers ces deux expériences, validé notre modèle de conception d'expériences interactives à base d'automates temporisés et la mécanique de supervision permettant de l'utiliser dans le pilotage d'un robot Nao. Ces expérimentations nous ont notamment prouvé que la prise en compte du temps dans les interactions est une chose très importante. En effet, nous avons remarqué que la rapidité de réponse du joueur dépend, la plupart du temps, de son âge. Ainsi, un enfant à tendance à répondre plus vite, voire même avant le début de la phase d'écoute du robot. Un adulte, quant à lui, prend davantage de temps pour répondre. De plus, le choix des délais d'attente du robot devant interagir avec le joueur et sa vitesse de parole doivent être paramétrés avec précaution. Ces paramètres ont, en effet, une forte conséquence sur la qualité de l'interaction et sur la perception du joueur sur le jeu.

Nous avons donc proposé un système d'adaptation par bouclage de pertinence à base de *Q-Learning*. Si l'adaptation des paramètres de reconnaissance vocales ont permis d'obtenir une adaptation en adéquation avec les comportements de l'utilisateur, ce n'est pas le cas pour les paramètres de temps.

Nous allons maintenant présenter, dans le chapitre suivant, une conclusion finale de ces travaux ainsi que les perspectives en découlant.

Chapitre 8

Conclusion et perspectives

Sommaire

Pour conclure ces travaux de thèse, nous allons rappeler les problématiques qu'ils adressent et que nous avions présentés en introduction de ce document. Ceux-ci ont permis de déboucher sur plusieurs propositions et contributions. Nous en proposerons une discussion afin de présenter leurs intérêts, limites et perspectives.

8.1 Rappel des problématiques adressées

La modélisation de l'interaction est une tache complexe. Par exemple, lors de la première expérimentation réalisée au Muséum d'Histoire Naturelle de La Rochelle, le scénario de l'activité de jeu, déployé sur le robot Nao, a été conçu par l'outil propriétaire Choregraphe. Bien que cet outil soit efficace pour des interactions simples, il ne permet pas de concevoir une expérience complexe, avec plusieurs interactions et de nombreux contenus. La modélisation y est linéaire et source d'erreurs (liée aux nombreux embranchements entre les comportements). Les contenus associés aux interactions sont intégrés au modèle rendant difficile la reproductibilité. Il est donc évident, à travers cette expérimentation, que l'utilisation d'un outil permettant une gestion efficace des contenus et une modélisation modulaire est nécessaire.

La première problématique concerne donc le fait de disposer d'un modèle capable de prendre en compte les dimensions de temps, de contenus et d'interactions. Nous voulions ici pouvoir proposer un modèle facilitant la phase de conception d'un scénario interactif et répondant aux critères de :

- **généricité :** la plupart des outils de conception adresse une problématique précise et ne propose pas la possibilité de concevoir d'autres types d'activités (pour des plateformes différentes). Nous souhaitions proposer un modèle suffisamment générique pour la conception d'activités interactives variées (HRI, jeux...) ;
- **modularité et reproductibilité :** la conception d'un système robotique complexe par une approche monolithique (type Choregraphe) peut difficilement être utilisée. La nécessité de diviser le modèle se fait alors ressentir. Les approches réductionnistes (également dites par composants) sont alors pertinentes et adaptées à cette problématique. La reproductibilité sera en conséquence améliorée par la mise à disposition d'un ensemble d'entités réutilisables pour de futures modélisations ;
- **extensibilité :** un scénario efficace et vérifié peut être amené à être modifié pour étendre son périmètre, modifier ses contenus (les dialogues par exemple). Là encore, les outils monolithiques et dédiés à une activité cible proposent rarement des solutions à ces problématiques. Nous souhaitions donc faciliter l'ajout et la modification de contenus ou de comportements ;
- **temporalité :** la notion de temps est importante quand on se place dans le domaine de l'interaction homme robot. La nécessité de contraindre temporellement les interactions se fait ressentir notamment dans les interactions vocales. Nous souhaitions donc pouvoir proposer une gestion efficace du temps sur les différentes entités du modèle pour garantir une fluidité des échanges.

Les premières expériences menées nous ont montré, à travers la diversité du public et de son appréhension de l'interaction avec le robot, que le paramétrage des interactions conditionne la bonne réalisation de celles-ci. En effet, les personnes âgées vont avoir tendance à hésiter à répondre au robot ou en dehors des temps d'écoute (alors que le jeune public a tendance à anticiper sur la phase d'écoute du robot). A l'issu de ces expériences, nous avons mené une réflexion sur la nécessaire adaptation de l'activité à l'usage.

La seconde problématique de cette thèse est donc de proposer un système adaptatif permettant de modifier dynamiquement le modèle, par bouclage de pertinence. En effet, nous avons vu précédemment qu'il était difficile de prévoir a priori le comportement final de l'utilisateur. Ainsi, le paramétrage du modèle contrôlant l'activité peut faire percevoir à l'utilisateur une certaine frustration s'il n'est pas adapté à ses attentes. Nous souhaitions donc ici proposer une méthode d'adaptation dynamique du modèle par l'analyse à la fois des expériences passées mais également du comportement de l'utilisateur courant.

8.2 Contributions

Ces travaux de thèse ont permis d'aboutir à un modèle de conception d'expériences interactives. La modélisation a été divisée en deux étapes. La première étape permet de définir les comportements atomiques de l'activité. Dans un second temps, des agents implémentent par héritage multiple ces comportements. Les agents spécifient, pour chacun

d'eux, le contenu et les actions à exécuter sur le processus contrôlé (dans notre cas, le robot Nao). Cette spécification est réalisée par l'utilisation d'une base de données dans laquelle sont stockés l'ensemble des contenus utilisables pour la réalisation de l'activité robotique. Les agents sont ensuite regroupés au sein de contextes d'exécution eux même ordonnés.

Notre contribution repose sur deux points principaux :

- **une construction simplifiée du scénario** basée sur un modèle à deux couches facilitant la modularité, la réutilisation et la construction automatique des entités. Nous simplifions la tâche de modélisation par une méthode en deux étapes :
  - tout d'abord, la description de l'expérience est réalisée de manière abstraite, en définissant les entités réutilisables (les comportements et les patterns), modélisées par des automates temporisés. Cette étape crée la *couche déclarative* ;
  - la *couche d'implémentation* est, quant à elle, définie par l'instanciation des entités génériques dans des agents, regroupés dans des contextes d'exécution ordonnés et agrégés dans un automate temporisé.
- **la gestion de contenus externes**. Les outils de conception conventionnels (comme Chorégraphe) ne permettent pas cette gestion. La modélisation et l'adaptation d'une expérience d'un musée à un autre se serait avérée être une tâche laborieuse. Nous proposons donc une architecture complète permettant d'externaliser les contenus. De plus, la notion de temporalité est intégrée à notre modèle, que ce soit au niveau du comportement ou du contexte. Notre outil de supervision est alors chargé d'extraire de la base de données, le contenu nécessaire au contrôle du processus.

Ce modèle est vérifiable pour des propriétés de sûreté et de vivacité. Nous proposons donc une conversion du modèle CIT vers le modèle UPPAAL afin d'utiliser le module de model-checking asocié.

Toutes ces fonctionnalités ont été mises en oeuvre dans deux outils :

- CELTIC : l'éditeur web permettant de concevoir l'expérience de façon modulaire en implémentant les algorithmes du modèle CIT ;
- EDAIN : l'outil de supervision qui, à partir d'un modèle généré par la plateforme CELTIC, permet de contrôler l'activité et de piloter les robots Aldebaran (Nao et Pepper).

Pour tester cette approche, nous avons modélisé deux jeux sérieux avec le robot Nao à destination du Muséum d'Histoire Naturelle de La Rochelle et du musée Sainte-Croix de Poitiers. La validation de l'efficience de notre modèle et de la plateforme de supervision nous permet aujourd'hui d'envisager la conception d'un jeu plus complexe.

Enfin, la dernière contribution vise à combiner cette approche *top-down* avec une approche *bottom-up* afin d'intégrer une boucle de pertinence. Ainsi, en raison des observations faites lors de l'exécution de l'activité et du processus d'apprentissage automatique, notre objectif est de pouvoir modifier le modèle et en particulier certains paramètres

contrôlant l'activité. Le processus d'apprentissage automatique permet d'adapter l'expérience au comportement réel de l'utilisateur final, ce qui est difficilement prévisible lors de la conception. Notre proposition tient dans l'intégration d'un algorithme d'apprentissage par renforcement et plus particulièrement celui du *Q-Learning*. Si ce module d'apprentissage a été intégré à l'outil EDAIN pour la supervision de l'activité et l'adaptation des paramètres, de nouveaux tests permettant de valider l'adaptation des paramètres temporels restent à mener.

8.3 Limites

Si ces travaux de thèse ont permis d'aboutir à une chaîne complète de conception et de supervision d'activités robotiques par la réalisation de deux plateformes logicielles, nous identifions aujourd'hui un certain nombre de limites. La première concerne notre outil d'édition CELTIC. Si sa plateforme web lui permet d'être à disposition de la communauté scientifique, celle-ci ne gère actuellement pas un travail collaboratif de modélisation. De plus, une fonctionnalité importante n'est pas encore disponible : la spécification des contenus par l'utilisation de ceux disponibles dans la base de données. Celle ci s'est, jusqu'ici, réalisée à la main après génération du modèle final (indication des différents identifiants des contenus dans le fichier XML manuellement). Cette fonctionnalité, relevant du développement logiciel, apportera une simplification marquée dans la phase de conception.

Enfin, l'adaptation des paramètres est encore en phase de tests et nécessite d'être consolidée par des expériences de plus grande échelle.

8.4 Perspectives

❶ Évolution du modèle : Nous avons proposé un modèle à deux couches permettant de concevoir une activité interactive de façon modulaire, dans lequel les différentes interactions ne sont pas liées à un espace particulier. Les robots et les joueurs étant mobiles, l'intégration de l'espace dans le modèle représente une intégration possible dans le modèle. L'objectif ici est de pouvoir spécifier le lieu dans lequel l'action doit se dérouler. Cette future intégration lève plusieurs problématiques :

- la cohérence et la représentation de l'information de localisation dans le modèle. Doit-on l'externaliser ou l'intégrer à la définition du comportement, de l'agent ou du contexte ?
- la vérification d'un tel modèle. Comment peut-on s'assurer de la présence d'un utilisateur dans un lieu donné pour effectuer une action particulière ?

❷ Tests de temporisation du graphe de contexte : Les expérimentations que nous avons menées au cours de ces travaux de recherche n'intègrent pas la temporisation de l'enchaînement des contextes. Si la sémantique statique et dynamique l'autorisent, des tests

seront prochainement menés sur cet aspect. De même, la réalisation d'un model-checking temporisé sur le graphe de contexte sera effectuée pour valider cette intégration.

❸ Supervision de nouvelles activités : Nous avons jusqu'ici expérimenté notre modèle à travers la conception de deux jeux sérieux pilotés sur des robots Nao. Si la supervision d'activité sur le robot Pepper (même famille que Nao) a montré son efficacité, qu'en est-il sur d'autres robots ? Une perspective possible serait de tester notre modèle et notre outil de supervision sur le bras robotisé du laboratoire. Une seconde expérimentation pourrait également consister à superviser un jeu vidéo en s'interfaçant, par exemple, avec le moteur UnrealEngine.

❹ Évolution de l'outil de conception CELTIC : Afin de pallier les difficultés rencontrées par l'utilisation de CELTIC dans sa version web, nous souhaitons, comme plusieurs plateformes telle que RobotML [DKS+12], concevoir un plugin Eclipse [1] pour la conception d'expériences interactives sur le modèle CIT. Ainsi, la mise à disposition de la communauté sera facilitée. Chaque utilisateur disposera d'un environnement de conception stable et pourra se connecter sur sa propre base de données.

❺ Ajout ou inhibition de comportements : Enfin, la mécanique modulaire favorise l'ajout et le retrait de comportements aux agents. Cette action pourrait être réalisée dynamiquement et permettre un bouclage de pertinence encore plus efficace. Outre l'adaptation dynamique des paramètres par apprentissage, une des perspectives des travaux tient dans l'adaptation dynamique des comportements autorisés ou inhibés pour un agent dans un contexte.

1. Eclipse est Environnement de Développement Intégré `https://www.eclipse.org`

Publications issues de ces travaux de recherche

2019 Damien Mondou, Armelle Prigent, Arnaud Revel. CELTIC/EDAIN : une approche de modélisation et de supervision d'expériences interactives. *Conférence Nationale en Intelligence Artificielle - CNIA*, Jérôme Lang, Juillet 2019, Toulouse, France. pp. 77-85

2018 Damien Mondou, Armelle Prigent, Arnaud Revel. A Dynamic Scenario by Remote Supervision : A Serious Game in the Museum with a Nao Robot. *Advances in Computer Entertainment Technology - ACE*, Décembre 2017. Lecture Notes in Computer Science, vol 10714. Springer, Cham

2016 Damien Mondou, Armelle Prigent, Arnaud Revel. Gestion adaptative des contenus numériques : Proposition d'un framework générique par apprentissage et scénarisation dynamique. *INFORSID*, Mai 2016, Grenoble, France, pp. 25-28

Damien Mondou, Armelle Prigent, Arnaud Revel, Nicolas Rempulski. Towards a hybrid approach for supervising interactive adaptive systems, *MOVEP*, Juin 2016, Gênes, Italie

Bibliographie

[ABG14] Elise Aspord, Joffrey Becker, and Emmanuelle Grangier. *Link human/robot*. Van Dieren eds, 2014.

Voir page 15.

[ACF+98] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab, and Félix Ingrand. An architecture for autonomy. *The International Journal of Robotics Research*, 17(4) :315–337, 1998.

Voir page 50.

[AD92] Rajeev Alur and David Dill. *The theory of timed automata*, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

Voir pages 29, 39 et 81.

[ANSG+06] Issa A.D. Nesnas, Reid Simmons, Daniel Gaines, Clayton Kunz, Antonio Diaz-Calderon, Tara Estlin, Richard Madison, John Guineau, Michael McHenry, and I-Hsiang Shu. Claraty : Challenges and steps toward reusable robotic software. *International Journal of Advanced Robotic Systems*, 3, 03 2006.

Voir page 50.

[Ara09] Manuel Araújo. Modeling games with petri nets. *Digital Games Research Association (DiGRA)*, 2009.

Voir page 32.

[BBD+12] Marcello Bonfè, Fabrizio Boriero, Riccardo Dodi, Paolo Fiorini, Angelica Morandi, Riccardo Muradore, Liliana Pasquale, Alberto Sanna, and Cristian Secchi. Towards automated surgical robotics : A requirements engineering approach. In *2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 56–61, Juin 2012.

Voir pages 5, 31 et 48.

[BBF+01a] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen, and Pierre Mckenzie. *KRONOS — Model Checking of Real-time Systems*, pages 161–168. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

Voir page 83.

[BBF+01b] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen, and Pierre Mckenzie. *SMV — Symbolic Model Checking*, pages 131–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

Voir page 81.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM '06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

Voir pages 6, 50 et 51.

[Ben07] Pierre-jean Benghozi. Aproche générationnelle des pratiques culturelles et médiatiques. *Culture Prospective*, 3 :1–32, 2007.

Voir page 14.

[BFB18] Guillaume Bernard, Cyril Faucher, and Karell Bertet. Towards reconstruction of human trajectories in indoor environments. In *Proceedings of the EKAW 2018 Posters and Demonstrations Session co-located with 21st International Conference on Knowledge Engineering and Knowledge Management (EKAW 2018), Nancy, France*, pages 37–40, Novembre 2018.

Voir page 123.

[BGAH14] Sofiane Boucenna, Philippe Gaussier, Pierre Andry, and Laurence Hafemeister. A robot learns the facial expressions recognition and face/non-face discrimination through an imitation game. *International Journal of Social Robotics*, 6(4) :633–652, Nov 2014.

Voir page 16.

[BHM15] Eric Badouel, Loïc Hélouët, and Christophe Morvan. Petri nets with structured data. In Raymond Devillers and Antti Valmari, editors, *Application and Theory of Petri Nets and Concurrency*, pages 212–233, Cham, 2015. Springer International Publishing.

Voir page 32.

[BJ14] Franciny M. Barreto and Stéphane Julia. Modeling and analysis of video games based on workflow nets and state graphs. In *Proceedings of 24th*

Annual International Conference on Computer Science and Software Engineering, CASCON '14, pages 106–119, Riverton, NJ, USA, 2014. IBM Corp.

Voir pages 5, 32, 36, 37 et 48.

[BJJ18] Franciny Barreto, Joslaine Jeske, and Stéphane Julia. *A Timed Petri Net Model to Specify Scenarios of Video Games*, pages 467–473. Springer International Publishing, 01 2018.

Voir page 32.

[BK13] Falko Bause and Pieter Kritzinger. *Stochastic Petri Nets -An Introduction to the Theory*. Novembre 2013.

Voir page 38.

[BY03] Johan Bengtsson and Wang Yi. Timed automata : Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, 2003.

Voir page 69.

[Cav17] Ana Cavalcanti. Formal methods for robotics : Robochart, robosim, and more. In *Formal Methods : Foundations and Applications - 20th Brazilian Symposium, SBMF 2017, Recife, Brazil, November 29 - December 1, 2017, Proceedings*, pages 3–6, 2017.

Voir page 19.

[CB10] Marcello Carrozzino and Massimo Bergamasco. Beyond virtual museums : Experiencing immersive virtual reality in real museums. *Journal of Cultural Heritage*, 11(4) :452 – 458, 2010.

Voir page 17.

[CEG+14] Betty H. C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pelliccione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, and Norha M. Villegas. *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*, pages 101–136. Springer International Publishing, Cham, 2014.

Voir page 81.

[CES86] Edmund Melson Clarke, E. Allen Emerson, and Aravinda Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, April 1986.

Voir page 81.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

Voir page 81.

[CHS18] John J. Camilleri, Mohammad Reza Haghshenas, and Gerardo Schneider. A web-based tool for analysing normative documents in english. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1865–1872, New York, NY, USA, 2018. ACM.

Voir page 83.

[CL12] Hugo Costelha and Pedro Lima. Robot task plan representation by petri nets : modelling, identification, analysis and execution. *Autonomous Robots*, 33(4) :337–360, Nov 2012.

Voir pages 5, 37, 38 et 48.

[Clo07] Aurélie Clodic. *Supervision pour un robot interactif : action et interaction pour un robot autonome en environnement humain*. Theses, Université Paul Sabatier - Toulouse III, June 2007.

Voir pages 5 et 51.

[CMN13] Tanguy Coenen, Lien Mostmans, and Kris Naessens. Museus : Case study of a pervasive cultural heritage serious game. *J. Comput. Cult. Herit.*, 6(2) :8 :1–8 :19, May 2013.

Voir page 17.

[Die11] Diane Dietze. *Playing and learning in early childhood education*. Wadsworth Publishing Company, 2011.

Voir page 16.

[DKS+12] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In Itsuki Noda, Noriaki Ando, Davide Brugali, and James J. Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Voir pages 27, 48 et 135.

[DOJSP11] Guilherme Willian De Oliveira, Stéphane Julia, and Ligia Maria Soares Passos. Game modeling using workflow nets. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 838–843, Oct 2011.

Voir pages 5, 32, 35, 37 et 48.

[EGT+09] George Edwards, Joshua Garcia, Hossein Tajalli, Daniel Popescu, Nenad Medvidovic, Gaurav Sukhatme, and Brad Petrus. Architecture-driven self-adaptation and self-management in robotics systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 142–151, Mai 2009.

Voir page 81.

[FBC+18] Simon Foster, James Baxter, Ana Cavalcanti, Alvaro Miyazawa, and Jim Woodcock. Automating verification of state machines with reactive designs and isabelle/utp. *CoRR*, abs/1807.08588, Juillet 2018.

Voir page 81.

[FHC97] Sara Fleury, Matthieu Herrb, and Raja Chatila. Genom : A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *In International Conference on Intelligent Robots and Systems*, pages 842–848, 1997.

Voir page 49.

[Fou18] Mohammed Foughali. *Formal Verification of the Functional Layer of Robotic and Autonomous Systems*. PhD thesis, Institut national des sciences appliquées de Toulouse, 2018.

Voir pages 5 et 49.

[Gat92] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 809–815. AAAI Press, 1992.

Voir page 50.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1) :1 – 101, 1987.

Voir page 35.

[GLPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1 :134–152, décembre 1997.

Voir pages 39, 62, 65, 81, 83 et 85.

[GMSL18] Rong Gu, Raluca Marinescu, Cristina Seceleanu, and Kristina Lundqvist. Formal verification of an autonomous wheel loader by model checking. In *Proceedings of the 6th Conference on Formal Methods in Software Engineering*, FormaliSE '18, pages 74–83, New York, NY, USA, 2018. ACM.

Voir page 83.

[GRABR14] Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and Andrew W. Roscoe. Fdr3 — a modern refinement checker for csp. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 187–201, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

Voir page 31.

[Gro99] The Object Management Group. Omg unified modeling language specification. ., 1999.

Voir page 27.

[HAS14] Daniel Heß, Matthias Althoff, and Thomas Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1474–1481, Septembre 2014.

Voir page 81.

[HCL07] Kao Shing Hwang, Yu Jen Chen, and Ching Huang Lee. Reinforcement learning in strategy selection for a coordinated multirobot system. *IEEE Transactions on Systems, Man, and Cybernetics - Part A : Systems and Humans*, 37(6) :1151–1157, Novembre 2007.

Voir page 97.

[HEZ+14] Jeff Huang, Cansu Erdogan, Yi Zhang, Brandon Moore, Qingzhou Luo, Aravind Sundaresan, and Grigore Rosu. Rosrv : Runtime verification for robots. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, pages 247–254, Cham, 2014. Springer International Publishing.

Voir page 81.

[HGA15] Yujing Hu, Yang Gao, and Bo An. Accelerating multiagent reinforcement learning by equilibrium transfer. *IEEE Transactions on Cybernetics*, 45(7) :1289–1302, Juillet 2015.

Voir page 97.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 3–29, Cham, 2017. Springer International Publishing.

Voir page 81.

[HNSY94] Ta Henzinger, X Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111 :193–244, 1994.

Voir page 39.

[Hol03] Gerard Holzmann. *Spin Model Checker, the : Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.

Voir page 81.

[JCC18] Sébastien Jacquot, Gaël Chareyron, and Saskia Cousin. Le tourisme de mémoire au prisme du big data. cartographier les circulations touristiques pour observer les pratiques mémorielles. *Mondes du Tourisme [En ligne]*, 14, 2018.

Voir page 14.

[Jen87] Kurt Jensen. *Coloured Petri nets*, pages 248–299. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987.

Voir page 34.

[Jen98] Kurt Jensen. *An introduction to the practical use of coloured Petri Nets*, pages 237–292. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

Voir pages 34 et 37.

[JLS02] Henrik Jensen, Kim Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. *BRICS Report Series*, 3, 01 2002.

Voir page 83.

[JRR15] Marcin Jamro, Dariusz Rzonca, and Wojciech Rząsa. Testing communication tasks in distributed control systems with sysml and timed colored petri nets model. *Computers in Industry*, 71 :77 – 87, 2015.

Voir page 32.

[KGFP07] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where's waldo ? sensor-based temporal logic motion planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3116–3121, Avril 2007.

Voir page 81.

[KI05] Takayuki Kanda and Hiroshi Ishiguro. Communication robots for elementary schools. *Proceedings of AISB'05 Symposium Robot Companions : Hard Problems and Open Challenges in Robot-Human Interaction (Hatfield Hertfordshire*, pages 54–63, 2005.

Voir page 98.

[KJM09] Abir Beatrice Karami, Laurent Jeanpierre, and Abdel Illah Mouaddib. Partially observable markov decision process for managing robot collaboration with human. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 518–521, Nov 2009.

Voir page 97.

[KM98] Kurt Konolige and Karen Myers. The saphira architecture for autonomous mobile robots. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 211–242. MIT Press, Cambridge, MA, USA, 1998.

Voir page 50.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0 : Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

Voir page 83.

[KNTU02] K. Kobayashi, A. Nakatani, H. Takahashi, and T. Ushio. Motion planning for humanoid robots using timed petri net and modular state net. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 6, pages 6 pp. vol.6–, Oct 2002.

Voir page 37.

[KRV+15] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17 :184 – 206, 2015. 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

Voir page 94.

[KS10] Markus Klotzbücher and Peter Soetens. Orocos rtt-lua : an execution environment for building real-time robotic domain specific languages. 01 2010.

Voir page 31.

[KSB16] David Kortenkamp, Reid Simmons, and Davide Brugali. *Robotic Systems Architectures and Programming*, pages 283–306. Springer International Publishing, Cham, 2016.

Voir page 26.

[KSE13] Abir Beatrice Karami, Karim Sehaba, and Benoît Encelle. Adaptive and personalised robots - learning from users' feedback. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 626–632, Nov 2013.

Voir page 98.

[LFD+18] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems : A survey. *CoRR*, abs/1807.00048, 2018.

Voir pages 26, 80 et 81.

[LMA06] Arne Lehmann, Ralf Mikut, and Tamim Asfour. Petri nets for task supervision in humanoid robots. In *in Proc., 37th International Symposium on Robotics (ISR 2006*, pages 71–73, 2006.

Voir page 37.

[LP18] Charles Lesire and Franck Pommereau. Aspic : an acting system based on skill petri net composition. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*, Madrid, Spain, October 2018.

Voir pages 38 et 48.

[LPPR13] Thi Thieu Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems : A case study. *Int. J. Softw. Tools Technol. Transf.*, 15(3) :211–228, June 2013.

Voir page 83.

[LRO13] Matthieu Lapeyre, Pierre Rouanet, and Pierre-Yves Oudeyer. Poppy : a New Bio-Inspired Humanoid Robot Platform for Biped Locomotion and Physical Human-Robot Interaction. In *Proceedings of the 6th International Symposium on Adaptive Motion in Animals and Machines (AMAM)*, Darmstadt, Germany, March 2013.

Voir page 15.

[MMT+08] T. Miyashita, P. Meier, T. Tachikawa, S. Orlic, T. Eble, V. Scholz, A. Gapel, O. Gerl, S. Arnaudov, and S. Lieberknecht. An augmented reality museum guide. *Proceedings - 7th IEEE International Symposium on Mixed and Augmented Reality 2008, ISMAR 2008*, pages 103–106, 2008.

Voir page 15.

[MNPW98] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent : To boldly go where no ai system has gone before. *Artif. Intell.*, 103 :5–47, 1998.

Voir page 50.

[MR17] Marco Montali and Andrey Rivkin. *DB-Nets : On the Marriage of Colored Petri Nets and Relational Databases*, pages 91–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

Voir page 32.

[MRL+16] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. Robochart : a state-machine notation for modelling and verification of mobile and autonomous robots. *Tech. Rep.*, 2016.

Voir pages 5, 29, 30 et 48.

[MRL+19] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. Robochart : modelling and verification of the functional behaviour of robotic applications. *Software & Systems Modeling*, Jan 2019.

Voir pages 29 et 48.

[MRMBR19] Pablo Muñoz, María D. R-Moreno, David F. Barrero, and Fernando Ropero. Mobar : a hierarchical action-oriented autonomous control architecture. *Journal of Intelligent & Robotic Systems*, 94(3) :745–760, Jun 2019.

Voir page 50.

[MRO09] Conor Mcgann, Kanna Rajan, and Angel Garcia Olaya. Integrated planning and execution for robotic exploration. In *in International Workshop on Hybrid Control of Autonomous Systems*, 2009.

Voir page 26.

[MSD19] Gonçalo S. Martins, Luís Santos, and Jorge Dias. User-adaptive interaction in social robots : A survey focusing on non-physical interaction. *International Journal of Social Robotics*, 11(1) :185–205, Jan 2019.

Voir pages 94 et 97.

[Mur89] Tadao Murata. Petri nets : Properties, analysis and applications. *Proceedings of the IEEE*, 77(4) :541–580, Avril 1989.

Voir page 38.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL : A Proof Assistant for Higher-Order Logic*, volume 2283. Springer-Verlag Berlin Heidelberg, 2002.

Voir page 81.

[Oct09] Sylvie Octobre. Pratiques culturelles chez les jeunes et institutions de transmission : un choc de cultures ? *Culture prospective*, 1(1) :1, 2009.

Voir page 15.

[Pet66] Carl Adam Petri. *Communication with automata*. PhD thesis, Université d'Hamburg, 1966.

Voir page 32.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

Voir page 81.

[PR17] Armelle Prigent and Arnaud Revel. Cite – content interaction time and space : a hybrid approach to model man-robot interaction for deployment in museums. *EAI Endorsed Transactions on Creative Technologies*, 4(13), 11 2017.

Voir page 58.

[Pre01] Marc Prensky. *Digital Natives, Digital Immigrants.* Marc Prensky, October 2001.

Voir page 14.

[PYK16] Daniel Ponsini, Yilin Yang, and Seung-Yun Kim. Analysis of soccer robot behaviors using time petri nets. In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pages 270–274, Juillet 2016.

Voir page 39.

[QCPG+09] Morgan Quigley, Ken Conley, Brian P Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros : an open-source robot operating system. *ICRA Workshop on Open Source Software*, 3, 01 2009.

Voir pages 6 et 52.

[Ram74] Chander Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

Voir page 33.

[Rem13] Nicolas Rempulski. *Synthèse dynamique de superviseur pour l'exécution adaptative d'applications interactives.* PhD thesis, Université de La Rochelle, 2013.

Voir pages 5, 41, 48, 59 et 69.

[RKH+17] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf, and Tobias Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol. In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods*, pages 50–66, Cham, 2017. Springer International Publishing.

Voir page 81.

[RPCV01] N. Riviere, B. Pradin-Chezalviel, and R. Valette. Reachability and temporal conflicts in t-time petri nets. In *Proceedings 9th International Workshop on Petri Nets and Performance Models*, pages 229–238, Sept 2001.

Voir page 35.

[RPE+09] Nicolas Rempulski, Armelle Prigent, Pascal Estraillier, Vincent Courboulay, and Matthieu Perreira. Adaptive Storytelling Based On Model-Checking Approaches. *International Journal of Intelligent Games \& Simulation (IJIGS)*, 5, 2009.

Voir pages 41, 44 et 48.

[Sch00] Steve Schneider. *Concurrent and Real-time systems.* John Wiley and Sons, 2000.

Voir page 29.

[SF08] Ingrid Pramling. Samuelsson and Marilyn. Fleer. *Play and learning in early childhood settings : international perspectives.* Springer Dordrecht ; London, 2008.

Voir page 16.

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software : Beyond Object-Oriented Programming.* ACM Press and Addison-Wesley, 2nd edition edition, 2002.

Voir page 26.

[Sif77] Joseph Sifakis. Use of petri nets for performance evaluation. In *Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating Computer Systems*, pages 75–93, Amsterdam, The Netherlands, The Netherlands, 1977. North-Holland Publishing Co.

Voir page 33.

[SLH+18] Haobin Shi, Zhiqiang Lin, Kao-Shing Hwang, Shike Yang, and Jialin Chen. An adaptive strategy selection method with reinforcement learning for robotic soccer games. *IEEE Access*, PP :1–1, 02 2018.

Voir page 97.

[SSF+18] Sören Schreiner, Razi Seyyedi, Maher Fakih, Kim Grüttner, and Wolfgang Nebel. Towards power management verification of time-triggered systems using virtual platforms. In *Proceedings of the 18th International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation*, SAMOS '18, pages 81–88, New York, NY, USA, 2018. ACM.

Voir page 83.

[Sut88] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1) :9–44, 1988.

Voir page 96.

[TŢM08] Adriana Tapus, Cristian Ţăpuş, and Maja J. Matarić. User-robot personality matching and assistive robot behavior adaptation for post-stroke rehabilitation therapy. *Intelligent Service Robotics*, 1(2) :169, Feb 2008.

Voir page 95.

[VDAVH04] Wil Van Der Aalst and Kees Van Hee. *Workflow Management : Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

Voir page 34.

[VN92] Nukala Viswanadham and Yadati Narahari. *Performance Modeling of Automated Manufacturing Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

Voir page 38.

[Wa17] Meldon Wolfgang and al. Gaining robotics advantage. `https://www.bcg.com/publications/2017/strategy-technology-digital-gaining-robotics-advantage.aspx`, 2017.

Voir page 13.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3) :279–292, Mai 1992.

Voir page 96.

[WGS+18] Rui Wang, Yong Guan, Houbing Song, Xinxin Li, Xiaojuan Li, Zhiping Shi, and Xiaoyu Song. A formal model-based design method for robotic systems. *IEEE Systems Journal*, PP :1–12, 09 2018.

Voir pages 5, 44, 45 et 48.

[WLG+14] Rui Wang, Ping Luo, Yong Guan, Hongxing Wei, Xiaojuan Li, Jie Zhang, and Xiaoyu Song. Timed automata based motion planning for a self-assembly robot system. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5624–5629, Mai 2014.

Voir pages 5, 44 et 48.

[WPK18] MAx Weissman, Daniel Ponsini, and Seung-yun Kim. Prioritized situation awareness for soccer robots using timed transition petri nets. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 134–138, Juillet 2018.

Voir page 39.

[WS06] Kazuyoshi Wada and Takanori Shibata. Robot therapy in a care house - its sociopsychological and physiological effects on the residents. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 3966–3971, Mai 2006.

Voir page 95.

Annexes

Exemples de graphes de parties

La figure 8.1 présente, à partir des logs obtenus, le graphe d'un joueur ayant terminé sa partie. Il a répondu aux trois questions de Nao et a terminé sa partie en 21 minutes.
La figure 8.2 présente, quand à elle, des erreurs de reconnaissance vocale. Dans la majorité des cas, Nao reconnaissait la bonne réponse mais le taux de confiance était trop faible. Nous pouvons expliquer cela par le bruit ambiant présent dans la pièce à ce moment là. Bien que le joueur ait rencontré ces difficultés, il a pu, néanmoins, terminer sa partie en 29 minutes.
La figure 8.3 présente le cas d'un joueur ayant abandonné sa partie. Il a répondu correctement à la première question mais n'est jamais revenu auprès du robot pour la terminer.

Légende

- Présentation du jeu && annonce de la première question
- Réponse à la première question && annonce de la deuxième question
- Réponse à la deuxième question && annonce de la troisième question
- Réponse à la troisième question && fin du jeu

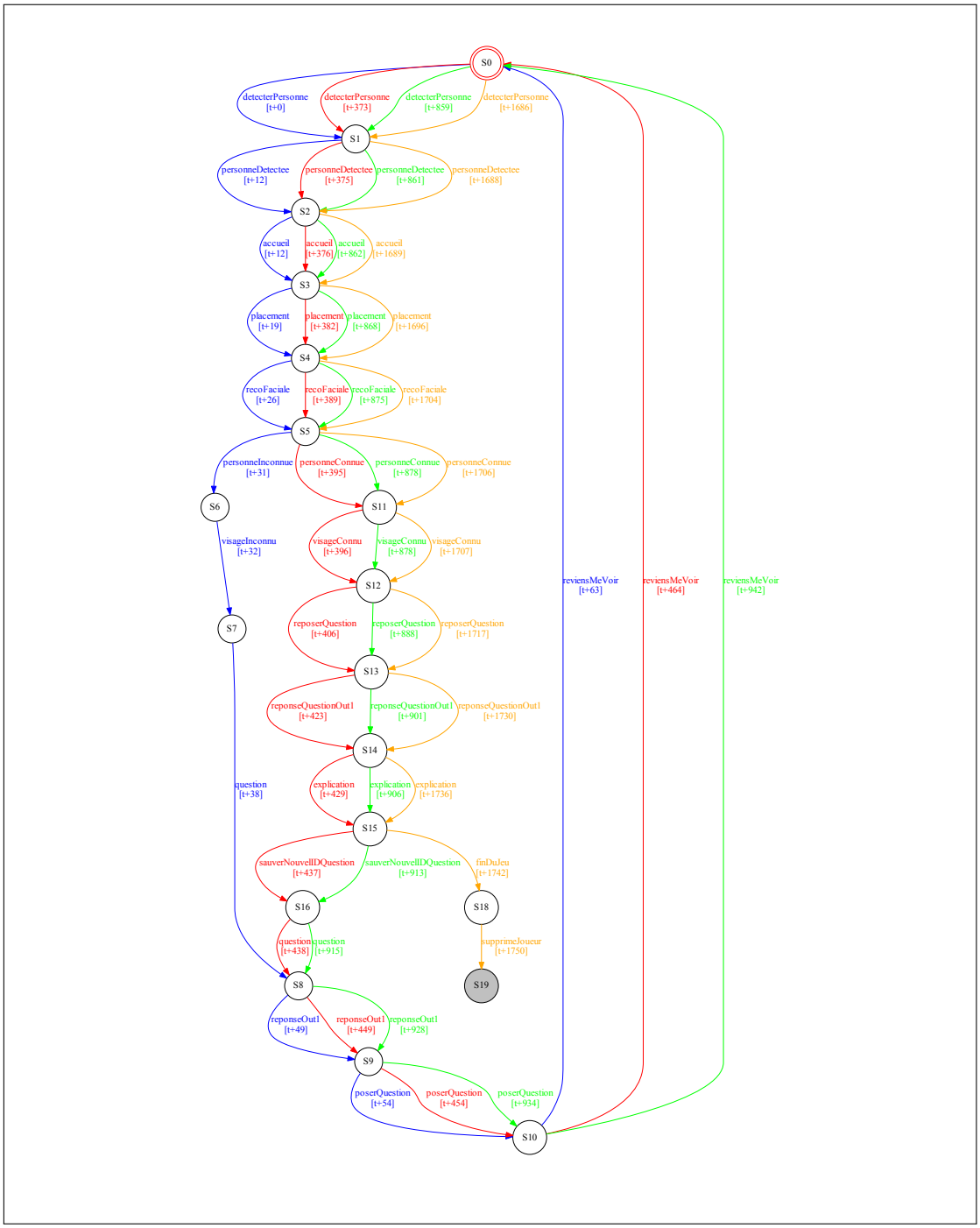


FIGURE 8.1 – Graphe d'un joueur ayant terminé sa partie sans erreur

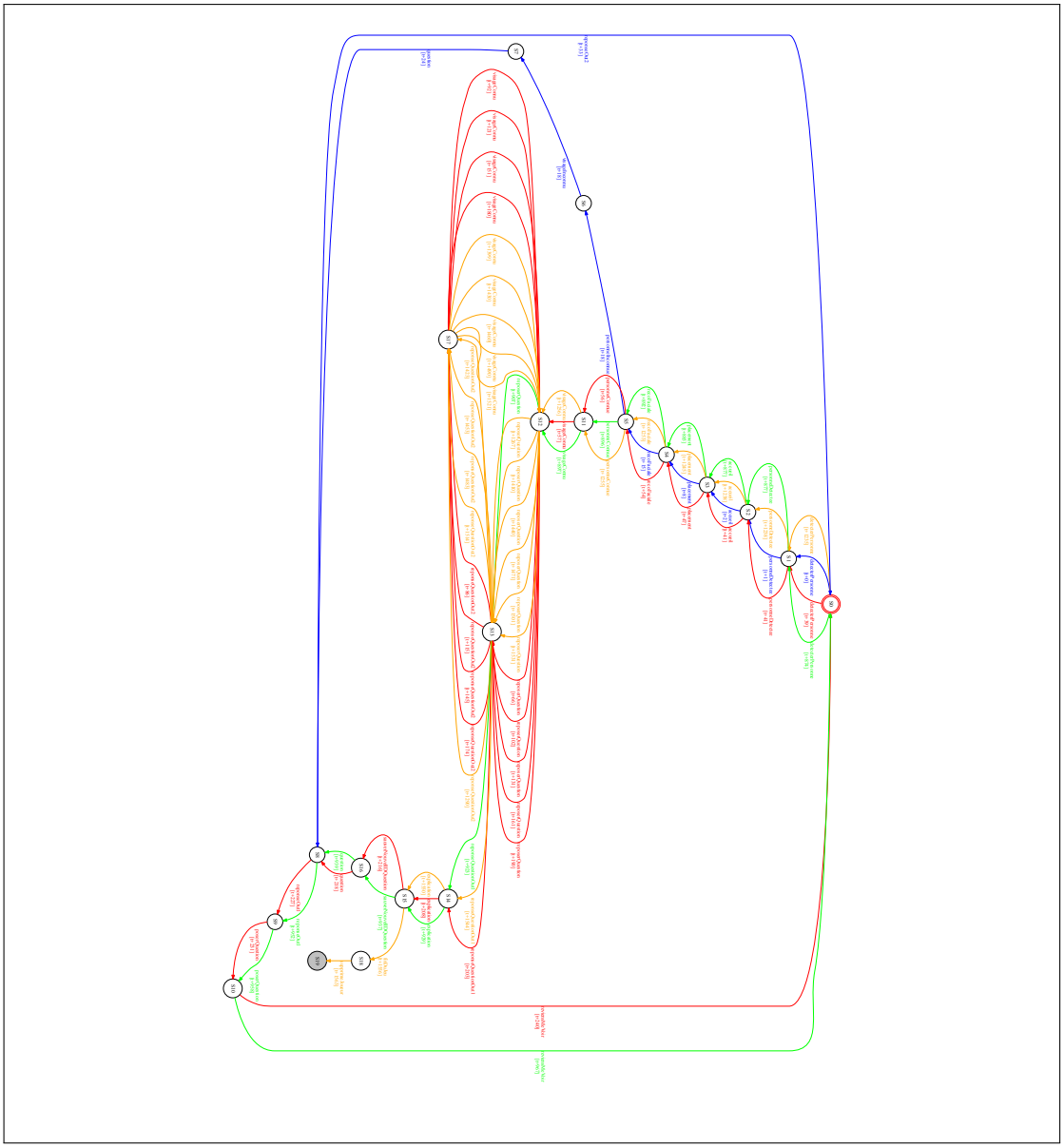


FIGURE 8.2 – Graphe d'un joueur ayant eu des problèmes de reconnaissances vocales avec Nao

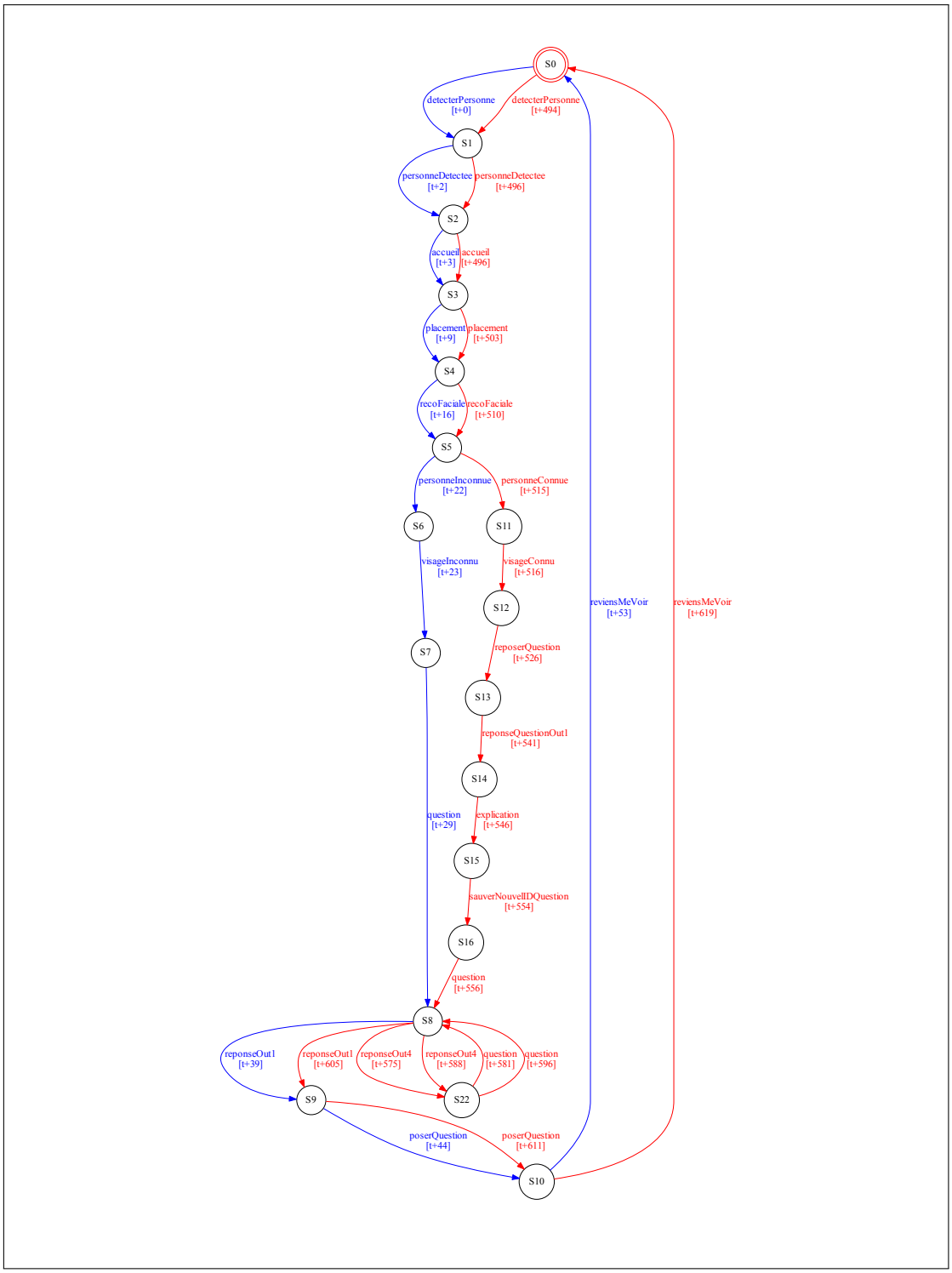


FIGURE 8.3 – Graphe d'un joueur ayant abandonné la partie

Gestion adaptative des contenus numériques : proposition d'un framework générique par apprentissage et re-scénarisation dynamique

Résumé : Cette thèse a pour objectif de proposer une architecture répondant aux problématiques de conception, de supervision, de pilotage et d'adaptation d'une expérience interactive. Nous proposons donc un framework complet destiné à faciliter la phase de modélisation d'un système interactif et garantissant une souplesse suffisante pour atteindre les objectifs de complexité, d'extensibilité, d'adaptabilité et d'amélioration par apprentissage automatique. Pour cela, le modèle formel, CIT, basé sur deux couches de description a été introduit. Le processus de supervision dynamique consiste à contrôler l'expérience interactive au regard du modèle formel, basé sur des réseaux d'automates temporisés à entrées/sorties. Deux plateformes logicielles, CELTIC (Common Editor for Location Time Interaction and Content) et EDAIN (Execution Driver based on Artificial INtelligence), implémentant respectivement le modèle CIT et le moteur de supervision de l'activité ont été développés au cours de cette thèse.

Mots clés : modélisation formelle, adaptation, interaction homme-robot

Adaptive content management : proposal of a generic framework through learning and dynamic adaptation

Abstract: This thesis aims to propose an architecture that addresses the design, supervision, management and adaptation of an interactive experience. We therefore propose a complete framework to facilitate the modeling phase of an interactive system and guarantee sufficient flexibility to achieve the objectives of complexity, scalability, adaptability and improvement through automatic learning. For this purpose, the formal model, CIT, based on two layers of description was introduced. The dynamic supervision process consists in controlling the interactive experience with regard to the formal model, based on networks of timed input/output automata. Two softwares, CELTIC (Common Editor for Location Time Interaction and Content) and EDAIN (Execution Driver based on Artificial INtelligence), implementing the CIT model and the activity supervision engine respectively, were developed during this thesis.

Keywords: formal model, adaptation, human-robot interaction

Laboratoire Informatique, Image, Interaction
Faculté des Sciences & Technologie
Avenue Michel Crépeau

17042 LA ROCHELLE CEDEX 1