Planification automatique pour personnes atteintes d'un déficit cognitif

par

Maxime La Placa

mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, Mai 2009

Your file Votre référence
ISBN: 978-0-494-53172-3
Our file Notre référence
ISBN: 978-0-494-53172-3

Le 30 juin 2009

le jury a accepté le mémoire de M. Maxime La Placa dans sa version finale.

Membres du jury

Mme Hélène Pigot
Directrice
Département d'informatique

M. Froduald Kabanza
Codirecteur
Département d'informatique

M. Shengrui Wang
Membre
Département d'informatique

M. Sylvain Giroux
Président-rapporteur
Département d'informatique

Sommaire

Il existe un vaste éventail de conditions pouvant induire une déficience cognitive chez un individu. On peut entre autres penser à la schizophrénie, la maladie d'Alzheimer, la déficience intellectuelle, les traumatismes crâniens ou les accidents cérébro-vasculaires. Chacune de ces conditions afflige l'individu d'une déficience cognitive présentant des caractéristiques différentes, mais qui provoque souvent une certaine perte d'autonomie découlant de difficultés de planification. Il est fort possible que l'individu éprouve des difficultés à planifier correctement des activités de la vie quotidienne telles que la préparation des repas, les activités d'hygiène ou même des activités de loisirs, qui peuvent être tout aussi importantes pour le processus de réadaptation. Un individu dans cet état requiert une forme d'assistance pour lui permettre de conserver une certaine autonomie.

Une solution souvent appliquée à cette problématique est d'assigner un clinicien professionnel à un individu atteint d'une déficience cognitive. Le clinicien assiste l'individu en planifiant les activités pour lesquelles ce dernier éprouve des difficultés, puis en lui fournissant un horaire qu'il peut alors suivre pour accomplir ses activités, sans avoir eu à les planifier lui-même. Cette solution permet à l'individu de rester relativement autonome. La tâche du clinicien est cependant difficile, car un clinicien est généralement affecté à un grand nombre d'individus, chacun souffrant de troubles cognitifs nécessitant des considérations particulières au niveau de la planification des activités.

Une orthèse cognitive nommée MOBUS a été développée au laboratoire DOMUS dans le cadre de projets connexes destinés à ce type de problématique. Cette orthèse fournit entre autres des services qui permettent d'assurer la communication entre le clinicien et son patient. Le clinicien dispose donc d'outils lui permettant de transmettre des horaires à son patient et de valider leur bonne exécution; cependant dans son état actuel, MOBUS n'offre aucun

service au clinicien lui permettant de faciliter le processus de planification en tant que tel. Le clinicien doit donc générer manuellement des horaires pour tous ses patients, une tâche ardue et répétitive nécessitant un temps considérable dont le clinicien pourrait disposer pour fournir d'autres types d'assistance à ses patients ou en prendre de nouveaux en charge.

Des recherches ont été effectuées dans le but de fournir des services au clinicien afin d'alléger sa tâche au niveau de la planification. Ce mémoire présente la démarche et les résultats de ces recherches, soit l'analyse de la problématique du clinicien, le développement d'un outil de planification automatique destiné à soutenir le clinicien, son intégration à l'orthèse cognitive MOBUS et finalement les résultats de certaines expérimentations de planification de cas cliniques typiques.

Veuillez prendre note que le masculin est utilisé au cours de ce mémoire uniquement dans le but d'alléger le texte.

Remerciements

J'aimerais remercier les personnes suivantes, sans lesquelles il m'aurait été impossible de compléter mes études.

Hélène Pigot, pour m'avoir accepté comme étudiant et pour m'avoir supporté tout au long de mes études.

Sylvain Giroux, pour avoir accepté de faire partie de mon jury et pour avoir facilité mon intégration au laboratoire DOMUS.

Shengrui Wang, pour avoir accepté de faire partie de mon jury et pour m'avoir assisté dans mes demandes de bourses.

Matthieu Castebrunet, Nicolas Marcotte, Céline Descheneaux, Pierre-Yves Groussard et Héloïse Moysan pour leur inépuisable soutien technique, académique, et surtout moral.

Francine Lapointe et Michel La Placa, pour absolument tout le reste.

Table des matières

Liste des abréviations

Abréviation	Signification
AVQ	*Activité de la Vie Quotidienne*
CSP	*Constraint Satisfaction Problem*
DOMUS	**DO**motique et informatique **M**obile à l'**U**niversité de **S**herbrooke
HTN	*Hierarchical Task Network*
IA	Intelligence Artificielle
MOBUS	**MOB**ilité à l'**U**niversité de **S**herbrooke
NASA	*National Aeronautics and Space Administration*
OO	Orienté Objet
PC	*Personal Computer*
PDA	*Personal Data Assistant*
TLP	*Temporal Logic Planning*
XML	*eXtensible Markup Language*

Liste des tableaux

Liste des figures

Introduction

Ce projet de recherche s'inscrit parmi les projets en développement du laboratoire DOMUS (DOmotique et informatique Mobile de l'Université de Sherbrooke). Les recherches effectuées au laboratoire DOMUS gravitent autour des habitats intelligents et de l'assistance aux personnes en perte d'autonomie due à une déficience cognitive. Le but de ces recherches est de permettre à ces personnes de conserver le plus d'autonomie possible, car il est généralement souhaitable qu'elles restent le plus indépendantes possible. On constate en effet que la dépendance d'une personne en perte d'autonomie envers un aidant (professionnel ou non) engendre non seulement des coûts matériels importants, mais peut également empirer la déficience cognitive dont elle souffre, ou accélérer sa progression. Le laboratoire DOMUS tente donc de mettre en place des outils qui permettent de réduire la dépendance d'une personne souffrant d'un déficit cognitif envers son ou ses aidant(s), ralentissant ainsi la progression de son déficit cognitif ou accélérant sa réadaptation. Plusieurs orthèses cognitives ont déjà été développées et déployées en milieu expérimental et clinique en lien avec cet objectif. Voyons maintenant dans quel contexte le présent projet de recherche s'insère.

Contexte

Plusieurs types de conditions peuvent engendrer une déficience cognitive chez un individu. On peut entre autres penser à la maladie mentale (schizophrénie et psychose par exemple), à la démence reliée au vieillissement (maladie d'Alzheimer), à des accidents provoquant des traumas cérébraux (traumatismes crâniens et accidents cérébro-vasculaires par exemple) ou à la déficience intellectuelle [16]. Un individu souffrant d'une telle condition et par conséquence d'un déficit cognitif est qualifié de *patient* dans la présente recherche. Le déficit cognitif du patient peut prendre diverses formes, soit des difficultés d'apprentissages, des troubles de comportement, des difficultés de résolution de problèmes, des troubles de

mémoire, et bien d'autres encore [36]. Cependant, la plupart des patients présentent des troubles d'organisation. Cette recherche s'intéresse à ce type de difficulté cognitive. Un patient présentant des troubles d'organisation ne peut plus planifier ses activités de la vie quotidienne (AVQ) correctement, et requiert de l'assistance de la part d'un aidant s'il doit conserver une certaine autonomie.

Lorsque le patient n'est pas assez autonome pour demeurer seul à domicile, il habitera dans des résidences alternatives, telles que foyer de groupe, famille d'accueil ou résidence supervisée. Dans ces hébergements, le logeur ou psychoéducateur sera responsable de s'assurer d'une vie sociale active pour le patient. Il planifiera des activités pour s'assurer que le patient réalise ses AVQ, ses activités de travail et ses activités de loisirs.

Une pratique courante dans le milieu clinique est d'assigner un aidant professionnel à un patient afin de l'aider à planifier ses activités. Un tel aidant est qualifié d'*intervenant* dans la présente recherche. Plusieurs patients sont généralement assignés à un intervenant simultanément. L'intervenant connaît les particularités des déficiences cognitives de ses patients et la nature de leurs activités quotidiennes. Son rôle est de planifier des horaires pour les activités qui ont été reconnues comme posant des problèmes au patient. Le patient peut ensuite exécuter ces horaires et accomplir ses activités sans avoir eu à les planifier lui-même.

Le patient et l'intervenant doivent disposer d'une méthode de communication qui permettra au patient d'accéder à l'assistance que son intervenant peut lui fournir. Le patient devrait idéalement pouvoir accéder à l'assistance de l'intervenant au sein de son environnement usuel ainsi que lors de ses déplacements. Un système permettant une telle communication est qualifié de *système mobile* dans la présente recherche. Une orthèse cognitive a été développée dans ce sens au Laboratoire DOMUS. MOBUS (MOBilité à l'Université de Sherbrooke) est un système qui permet à l'intervenant de spécifier des horaires à ses patients, et au patient de les consulter et d'en valider l'exécution sur une plate-forme mobile, qu'il peut facilement transporter dans ses déplacements. MOBUS est un outil dont l'utilité et la valeur ont été démontrées lors de plusieurs expérimentations en milieu clinique. MOBUS fournit d'excellentes fonctionnalités de communication entre le patient et l'intervenant [22].

Cependant, MOBUS n'offre aucun service qui supporte l'intervenant dans la planification des activités de ses patients. Un horaire doit être généré manuellement par l'intervenant pour chaque patient sur une base régulière (typiquement sur une base hebdomadaire). La tâche est compliquée par le grand nombre de patients assignés à un intervenant et par le fait que le déficit cognitif de chaque patient force l'intervenant à tenir compte de considérations particulières pour chaque horaire.

Dans ce contexte, l'intervenant doit allouer une grande partie de son temps à la gestion des horaires de ses patients sur une base cas par cas. Il s'agit là d'une tâche ardue et répétitive dont l'ampleur l'empêche de consacrer plus de temps à fournir d'autres types d'assistance à ses patients ou à prendre de nouveaux patients à sa charge.

Objectifs de recherche

Nous désirons donc faciliter la tâche de l'intervenant en lui fournissant un outil qui lui permettra d'automatiser la planification d'horaire en fonction de certaines caractéristiques associées à chaque patient. Cet outil devrait idéalement générer des horaires complets en fonction des caractéristiques des déficiences cognitives et des activités à accomplir de chaque patient, ou au moins fournir une base d'horaire que l'intervenant pourra compléter. Cet outil devrait également être en mesure de s'interfacer avec le système mobile, afin que les patients et intervenants déjà habitués à utiliser ce système puissent aisément continuer à le faire.

L'intervenant muni d'un tel outil pourra donc allouer moins de temps à la planification d'horaire, temps qu'il pourra consacrer à de nouveaux patients ou à fournir une meilleure assistance à ceux déjà à sa charge.

Méthodologie

Nous avons suivi la méthodologie suivante au cours de cette recherche et nous la présenterons en détail :

- Identification des besoins du patient et par conséquent de ceux de l'intervenant

- Inférence des caractéristiques que le planificateur doit présenter
- Sélection d'un type de planificateur approprié
- Implémentation d'un planificateur concret (modification et adaptation du type de planificateur)
- Tests sur scénarios typiques et interface avec le système mobile MOBUS

Résultats

Nous présenterons finalement les résultats de la recherche. Des scénarios typiques d'utilisation ont été recueillis lors de rencontres avec des intervenants ayant participé à des expérimentations du système mobile MOBUS en milieu clinique avec de véritables patients.

Des cas d'expérimentations ont été construits à partir de ces données et ont été soumis au planificateur implémenté. Nous discuterons des résultats obtenus et des futurs travaux dont pourront bénéficier le planificateur et le système mobile MOBUS.

Chapitre 1

Énoncé de la problématique

Notre objectif de recherche est de faciliter la tâche de l'intervenant en lui fournissant un outil qui lui permettra d'automatiser une partie de la planification d'horaire de ses patients. Pour accomplir cet objectif, nous désirons implémenter un planificateur capable de prendre automatiquement cette tâche en charge. Nous devons cependant considérer le contexte clinique dans lequel l'intervenant et le patient évoluent présentement, ainsi que leur actuelle utilisation du système mobile MOBUS. Lorsque nous aurons examiné le rôle de ces trois acteurs (intervenant, patient, MOBUS), nous pourrons alors examiner la problématique et définir les caractéristiques que le planificateur devra présenter ainsi que la nature de son intégration à MOBUS.

1.1 Le patient

Le patient est défini comme un individu atteint d'un déficit cognitif lui ayant causé une perte d'autonomie.

1.1.1 Description

Plusieurs types de conditions peuvent engendrer une déficience cognitive chez le patient. Dans le présent contexte, la clientèle généralement suivie par un intervenant peut souffrir d'une condition telle que [12]:

- Pathologies psychiatriques – schizophrénie, psychose.
- Démence reliée au vieillissement – maladie d'Alzheimer.

- Traumas – traumatismes crâniens, accidents cérébro-vasculaires, lésions cérébrales.
- Déficience intellectuelle – anomalies chromosomiques, malformations congénitales du système nerveux central, maladies génétiques, etc.

ou d'une combinaison de ces conditions. Toutes ces conditions engendrent un déficit cognitif, mais la nature de ce dernier peut s'avérer extrêmement variée. La plupart des cas présentent cependant au moins une forme de troubles d'organisation. Le patient éprouve des difficultés à planifier et donc exécuter correctement ses AVQ. Parmi les difficultés les plus régulièrement recensées, on retrouve [24]:

- Prise de médication : oubli, erreur de posologie, prise du mauvais médicament.
- Tâches ménagères : incapacité à réaliser des tâches d'entretien de la résidence correctement ou régulièrement.
- Rendez-vous en milieu clinique : oubli, difficulté à planifier ses déplacements.
- Hygiène : difficulté à accomplir des tâches d'hygiène personnelle régulièrement.
- Loisirs organisés : oubli, difficulté à planifier ses déplacements (les loisirs sont une partie importante du processus de réadaptation dans certains cas).
- Déplacements : difficulté à planifier correctement un trajet entre la résidence et le lieu d'un rendez-vous.

Les difficultés causées par un déficit cognitif varient de patient en patient, et ne sont pas nécessairement liées à la condition ayant provoqué ce déficit. Par exemple, pour deux patients atteints de schizophrénie, il est possible que l'un éprouve des difficultés à accomplir ses tâches ménagères, mais soit en mesure de planifier ses déplacements correctement, alors que l'autre n'éprouve aucun problème à entretenir sa résidence, mais soit incapable de planifier le déplacement nécessaire pour se rendre à un rendez-vous. Il n'existe pas de profil typique pour un déficit cognitif, chaque cas présente ses propres particularités [9]. Un patient

éprouvant ce type de difficulté a perdu une partie de son autonomie. Il est généralement souhaitable de permettre à un patient de garder le plus d'indépendance possible, en vertu du fait qu'extraire un patient de son environnement habituel pour l'intégrer dans un milieu clinique où il sera fortement encadré a généralement pour effet d'accentuer les déficits cognitifs ou d'en accentuer la dégénérescence [28]. Le patient a cependant besoin d'une forme d'assistance dans la planification de ses activités pour conserver son indépendance.

1.1.2 Rôle dans le présent contexte

Le patient en perte d'autonomie est assigné à un intervenant clinique qui lui fournira l'assistance nécessaire à la planification de ses AVQ pour lesquelles il éprouve des difficultés. Le patient pourra ainsi continuer à évoluer dans son environnement habituel.

Le patient n'est pas en contact direct ou continuel avec son intervenant. Il reçoit l'assistance de son intervenant via un système qui permet la communication bidirectionnelle entre les deux acteurs. Ce système est mobile, et peut l'accompagner dans ses déplacements, afin qu'il ait accès à l'assistance de son intervenant tant à l'intérieur qu'à l'extérieur de sa résidence. Le système mobile permet également au patient d'indiquer qu'il a complété les AVQ planifiées par son intervenant.

1.2 L'intervenant

L'intervenant est défini comme un aidant professionnel auquel des patients sont assignés.

1.2.1 Description

L'intervenant est un professionnel en milieu clinique qui détient une connaissance approfondie de la clientèle qu'il dessert, des conditions cliniques de celle-ci et des particularités des déficits cognitifs qui en résultent. Il a la responsabilité d'aider ses patients à planifier les AVQ pour lesquelles ils éprouvent des difficultés, mais ne doit pas fournir de l'assistance d'une manière qui rend ses patients dépendants de cette assistance. La raison d'être de l'intervenant est de permettre au patient de conserver le plus d'indépendance

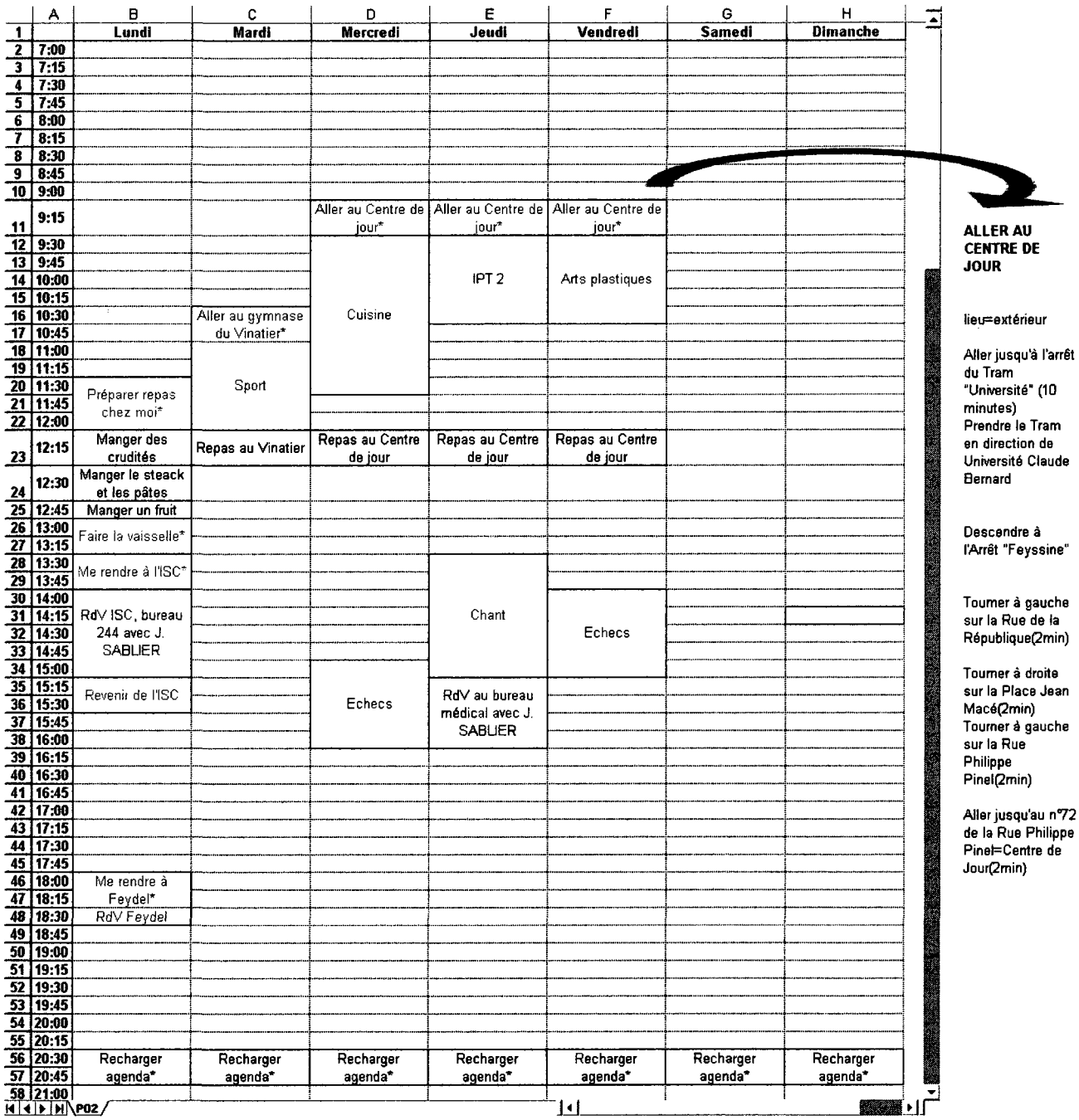
possible. L'intervenant doit donc fournir au patient l'assistance minimale qui lui permet d'accomplir les activités pour lesquelles il éprouve des difficultés.

L'intervenant a généralement la responsabilité de fournir de l'assistance à un grand nombre de patients, habituellement affligés par la même condition (schizophrénie, déficience intellectuelle, etc.).

1.2.2 Rôle dans le présent contexte

L'intervenant doit s'assurer que ses patients conservent le plus d'autonomie possible malgré leur déficit cognitif. Il doit donc identifier les AVQ pour lesquelles ses patients éprouvent des difficultés de planification et leur fournir une assistance en conséquence [19]. Cette assistance prend généralement la forme d'un horaire hebdomadaire explicitant les activités identifiées comme problématiques. Un exemple est donné à la Figure 1.

Une fois que l'intervenant a déterminé la nature des besoins d'assistance d'un patient, il doit produire un horaire pour ce dernier sur une base hebdomadaire. La nature de ces horaires varie dans le temps, et comme chaque patient assigné à l'intervenant présente des besoins typiquement différents, cette tâche est laborieuse. Dans certains cas, la réadaptation du patient peut aussi être un objectif du travail de l'intervenant. L'intervenant cherchera à planifier des horaires de moins en moins détaillés pour un patient, afin que celui-ci devienne graduellement moins dépendant de l'assistance fournie, jusqu'à ne plus en avoir besoin.



	A	B	C	D	E	F	G	H
1		Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
2	7:00							
3	7:15							
4	7:30							
5	7:45							
6	8:00							
7	8:15							
8	8:30							
9	8:45							
10	9:00							
11	9:15			Aller au Centre de jour*	Aller au Centre de jour*	Aller au Centre de jour*		
12	9:30			Cuisine	IPT 2	Arts plastiques		
13	9:45							
14	10:00							
15	10:15							
16	10:30		Aller au gymnase du Vinatier*					
17	10:45							
18	11:00		Sport					
19	11:15							
20	11:30	Préparer repas chez moi*						
21	11:45							
22	12:00							
23	12:15	Manger des crudités	Repas au Vinatier	Repas au Centre de jour	Repas au Centre de jour	Repas au Centre de jour		
24	12:30	Manger le steack et les pâtes						
25	12:45	Manger un fruit						
26	13:00	Faire la vaisselle*						
27	13:15							
28	13:30	Me rendre à l'ISC*			Chant			
29	13:45							
30	14:00					Echecs		
31	14:15	RdV ISC, bureau 244 avec J. SABLIER						
32	14:30							
33	14:45							
34	15:00			Echecs	RdV au bureau médical avec J. SABLIER			
35	15:15	Revenir de l'ISC						
36	15:30							
37	15:45							
38	16:00							
39	16:15							
40	16:30							
41	16:45							
42	17:00							
43	17:15							
44	17:30							
45	17:45							
46	18:00	Me rendre à Feydel*						
47	18:15							
48	18:30	RdV Feydel						
49	18:45							
50	19:00							
51	19:15							
52	19:30							
53	19:45							
54	20:00							
55	20:15							
56	20:30	Recharger agenda*	Recharger agenda*	Recharger agenda*	Recharger agenda*	Recharger agenda*	Recharger agenda*	Recharger agenda*
57	20:45							
58	21:00							

P02

ALLER AU CENTRE DE JOUR

lieu=extérieur

Aller jusqu'à l'arrêt du Tram "Université" (10 minutes)
Prendre le Tram en direction de Université Claude Bernard

Descendre à l'Arrêt "Feyssine"

Tourner à gauche sur la Rue de la République(2min)

Tourner à droite sur la Place Jean Macé(2min)
Tourner à gauche sur la Rue Philippe Pinel(2min)

Aller jusqu'au n°72 de la Rue Philippe Pinel=Centre de Jour(2min)

Figure 1 Un horaire typique planifié par l'intervenant

L'intervenant doit aussi effectuer un suivi de ses patients pour s'assurer que l'assistance fournie est adéquate, soit s'assurer que :

- l'horaire fourni à un patient cible bel et bien les activités avec lesquelles ce dernier éprouve des difficultés
- le patient accomplit les activités qui sont planifiées pour lui
- l'horaire fourni est assez détaillé pour permettre au patient d'être fonctionnel, sans lui causer une perte d'autonomie

L'intervenant utilise également le système mobile pour transmettre les horaires qu'il a planifiés pour ses patients, et pour faire le suivi des activités qui ont été complétées.

1.3 Le système mobile MOBUS

Le patient et l'intervenant utilisent un système mobile assurant la communication bidirectionnelle entre les deux parties. Le patient l'utilise pour recevoir l'assistance fournie par son intervenant, et l'intervenant l'utilise pour transmettre son assistance à ses patients et effectuer un suivi des activités complétées. Une implémentation d'un système capable d'assurer ces services a été développée au laboratoire DOMUS et déployée en milieu clinique. Il s'agit du système mobile MOBUS.

1.3.1 Description

MOBUS est une application client-serveur. Le serveur est conçu pour être déployé sur une plate-forme *hardware* PC de type *desktop*. Le client est aussi conçu pour être déployé sur *desktop*, mais peut également être déployé sur des appareils portables ayant une capacité d'accès à Internet, par exemple sur un *Personal Data Assistant* (PDA) ou un *Smartphone* [34,35]. Le client comporte deux versions, la *version intervenant* et la *version patient*. La Figure 2 illustre l'architecture MOBUS.

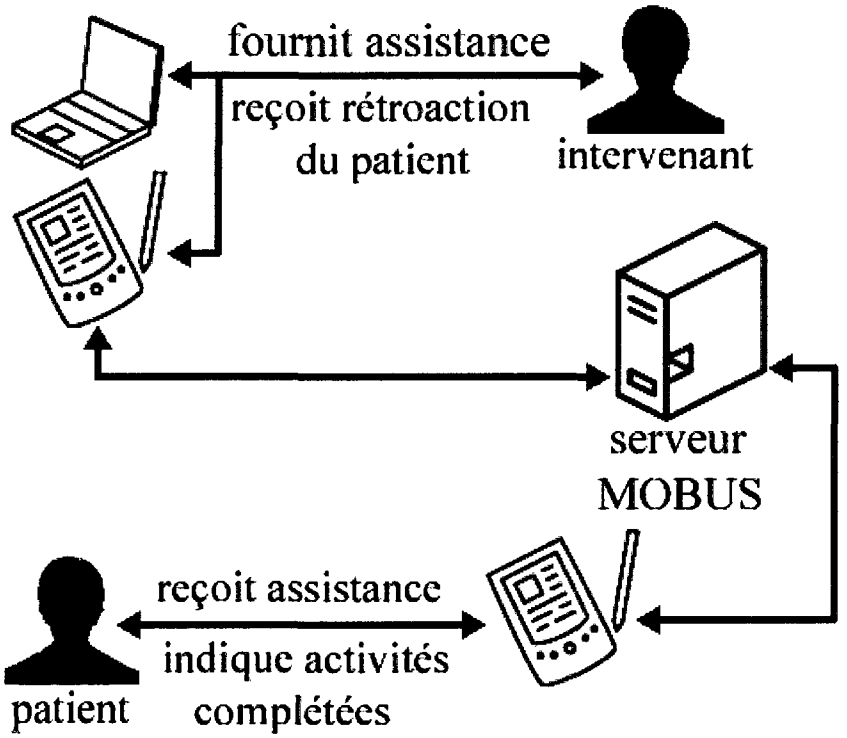


Figure 2 Architecture MOBUS

1.3.2 Rôle dans le présent contexte

MOBUS met en place les services nécessaires à la communication entre l'intervenant et ses patients.

La version intervenant permet à l'intervenant de se connecter au serveur MOBUS et d'y faire la gestion des ses patients, soit construire des horaires pour ses patients, les téléverser au serveur et plus tard obtenir de l'information concernant les activités complétées (ou négligées) par un patient en particulier. La *version client* permet au client de se connecter au serveur MOBUS et de télécharger l'horaire que son intervenant a planifié pour lui, puis d'y accéder dans un format rappelant un agenda. Le patient peut ensuite suivre son horaire et valider qu'il a accompli ses activités. Ces données de validation sont ensuite téléversées au serveur MOBUS qui les rend disponibles pour l'intervenant. La Figure 3 illustre l'affichage des activités à accomplir tel que présenté au patient.

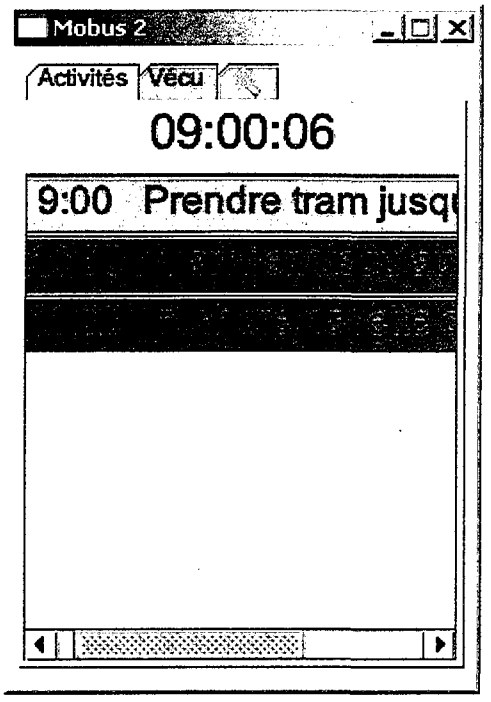


Figure 3 Interface du client patient

L'intervenant et le patient ont tous deux accès à un appareil portable, mais l'intervenant utilise généralement le client déployé sur une plate-forme *desktop*, en vertu du fait que cet environnement élimine plusieurs limitations au niveau de l'interface usager, et du fait qu'un *desktop* est très facilement accessible à partir de son environnement de travail. Les Figure 4 et Figure 5 illustrent les interfaces de cette version.

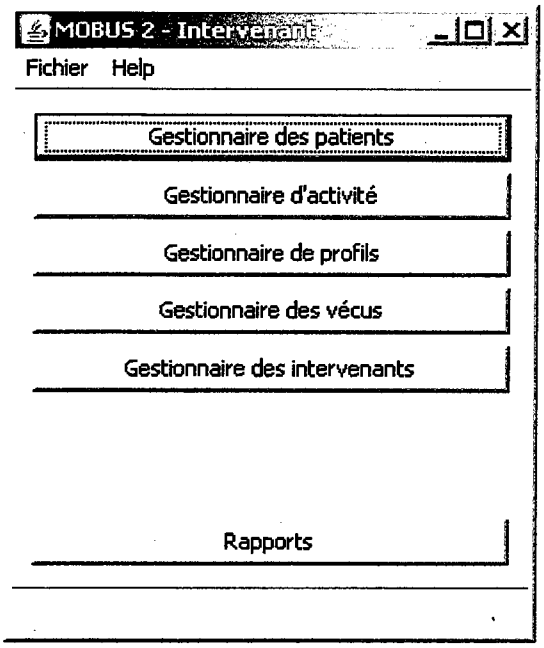


Figure 4 Interface du client intervenant (1)

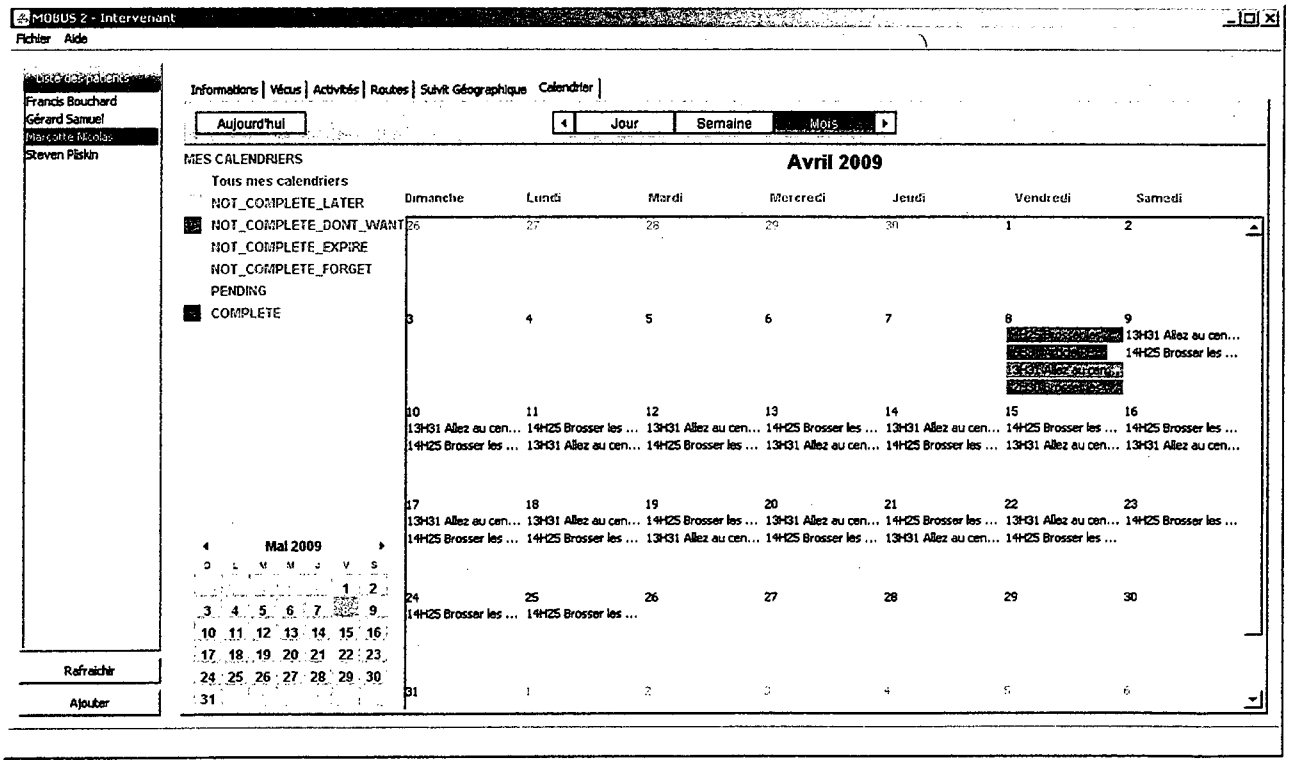


Figure 5 Interface du client intervenant (2)

Le patient utilise son client sur son appareil portable exclusivement, ce qui lui permet d'avoir accès à l'assistance de son intervenant en tout lieu.

MOBUS offre d'autres services, tels que la journalisation des symptômes du patient, la requête d'assistance en direct (bouton panique), l'aide contextuelle et la localisation GPS du patient. Une description plus complète est disponible en [23]. La présente recherche ne s'intéresse qu'à la partie agenda.

1.4 Problématique

L'intervenant dispose d'un temps limité pour s'occuper des patients qui lui sont assignés. Il doit planifier un grand nombre d'horaires pour des patients éprouvant des difficultés variées. De plus, plusieurs patients nécessiteront un horaire contenant des activités similaires (prises de médication, entretien ménager, loisirs organisés par exemple), mais comme les patients présentent des déficits différents et évoluent dans des environnements variés, l'intervenant ne peut généralement réutiliser le même horaire pour plusieurs patients, même si ces derniers doivent accomplir les mêmes activités. Par exemple, pour deux patients devant accomplir une tâche d'entretien ménager, il est possible que l'un éprouve des difficultés à l'accomplir l'après-midi, alors que l'autre éprouve des difficultés à l'accomplir une certaine journée de la semaine. L'intervenant doit faire la gestion de diverses *préférences* associées à chaque patient lors du processus de planification.

Les objectifs de réadaptation compliquent également la situation. En effet, même si un patient accompli hebdomadairement les mêmes activités, l'intervenant doit progressivement faire diminuer la quantité d'assistance qu'il lui fournit, et par conséquent ne peut pas réutiliser le même horaire semaine après semaine. L'intervenant doit faire la gestion de la *granularité* de l'assistance qu'il planifie pour ses patients.

La spécification des horaires au système mobile peut également s'avérer difficile, comme chaque activité planifiée doit être spécifiée manuellement, pour chaque horaire de chaque patient.

Dans son état actuel, MOBUS n'offre aucun service destiné à aider l'intervenant à produire des horaires pour ses patients. L'intervenant doit construire chaque horaire individuellement, puis le spécifier manuellement au système, tout en faisant la gestion de toutes les contraintes et particularités mentionnées jusqu'ici.

Ces facteurs rendent la tâche de l'intervenant laborieuse et répétitive, et le force à y consacrer une grande partie de son temps et de ses ressources. Si l'intervenant disposait d'un outil prenant en charge une partie du travail de planification, il disposerait alors de plus de temps et de ressources, qu'il pourrait utiliser pour fournir d'autres types d'assistance à ses patients ou à en prendre de nouveaux à sa charge.

1.5 Service d'aide à la planification

Afin d'alléger la tâche de l'intervenant, nous proposons le développement d'un service qui facilitera le processus de planification d'horaire. Ce service prendra la forme d'un planificateur automatique capable de générer des horaires valides pour les patients d'un intervenant, ou au moins générer des horaires pouvant servir de base à l'intervenant.

1.5.1 Rôle du planificateur

Dans le présent contexte, le planificateur aura la responsabilité de générer un horaire explicite pour un patient particulier à partir d'une spécification générale des activités qu'il doit accomplir, des préférences de ce dernier qu'il doit tenter de respecter et du niveau de granularité de l'assistance que l'intervenant désire fournir. L'horaire généré pour un patient est ensuite automatiquement exporté vers système mobile MOBUS, puis validé par l'intervenant. Une fois qu'un horaire a été exporté, il se comporte exactement comme un horaire ayant été entré manuellement par l'intervenant (i.e. le patient y a accès via le client MOBUS, l'intervenant peut consulter les informations sur les activités complétées, etc.). La Figure 6 illustre la manière dont le planificateur s'insère dans l'architecture MOBUS.

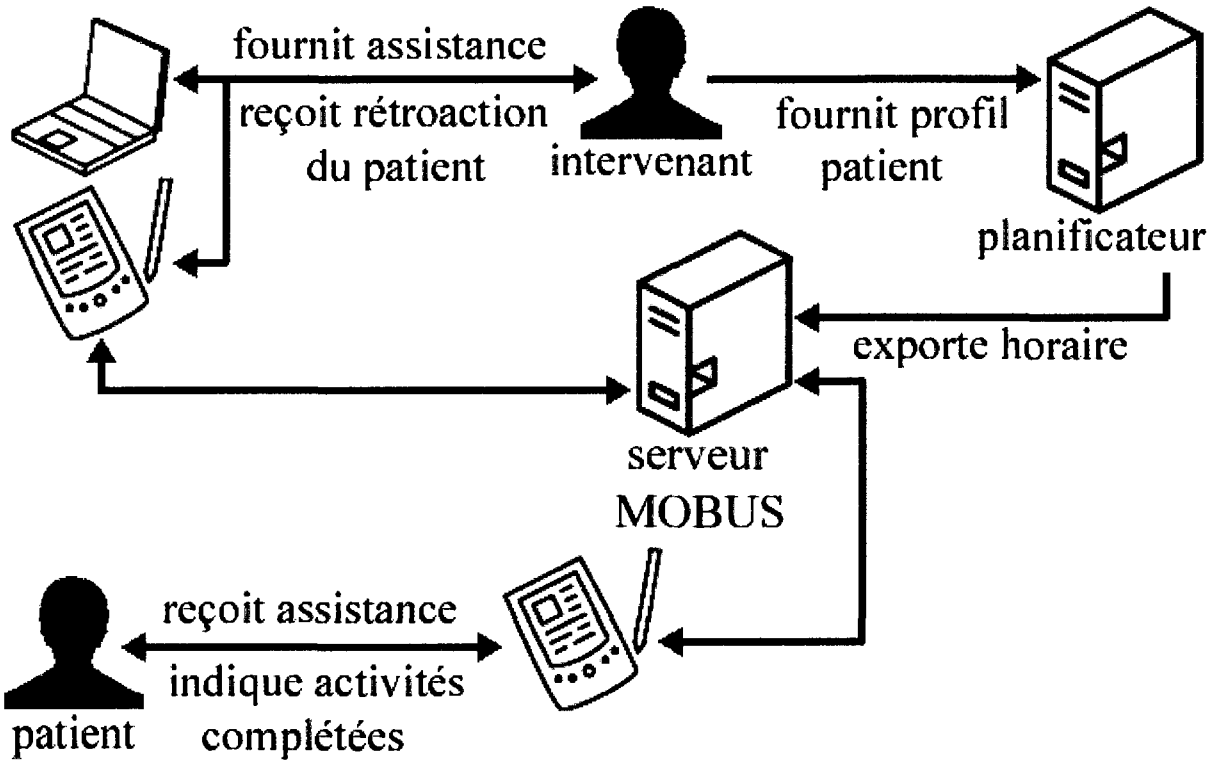


Figure 6 Le planificateur dans l'architecture MOBUS

1.5.2 Caractéristiques du planificateur

À la suite de l'analyse des rôles des acteurs du présent contexte, nous avons présenté le concept de planificateur automatique à des intervenants en milieu clinique utilisant déjà le système mobile MOBUS. Grâce à la coopération des intervenants, nous avons pu déterminer la nature des caractéristiques que le planificateur devrait comporter.

1.5.2.1 Planification orientées sur les tâches et notion de granularité

Le planificateur doit être orienté sur la notion de tâche décomposable en sous-tâches. L'intervenant exprime naturellement le problème de planification d'horaire d'une manière hiérarchique, du haut vers le bas. Lorsque nous avons demandé des exemples de planification pour un patient aux intervenants, nous avons obtenu des réponses telles que :

« Mon patient a un rendez-vous à la clinique. Il devra premièrement se préparer à sortir, puis se rendre à la clinique. Son rendez-vous est le matin, il devra donc s'habiller. Pour se rendre à la clinique, il devra prendre le bus. »

et ainsi de suite. On remarque qu'il est facile de déduire une hiérarchie de tâches à accomplir à partir de cet énoncé, tel qu'illustré par la Figure 7

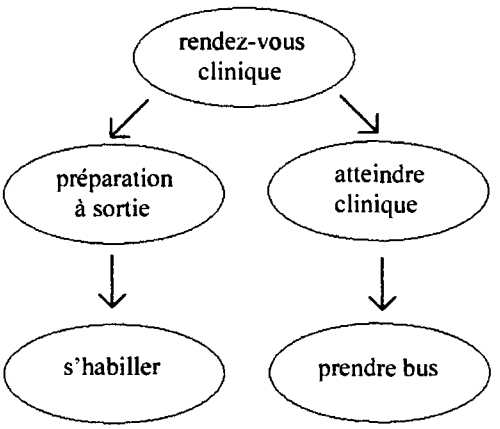


Figure 7 Une hiérarchie de tâches simple

Il n'est pas nécessaire que le planificateur soit orienté vers des buts à accomplir. L'intervenant planifie typiquement un horaire pour faire accomplir à son patient certaines tâches précises et connues (par exemple, prendre deux douches par semaine), et non pas pour lui faire atteindre un certain but (par exemple, être propre à la fin de la semaine). Les buts sont implicites et peu intéressants pour l'intervenant.

Les objectifs de réadaptation et les particularités de chaque patient rendent la notion de granularité très intéressante pour l'intervenant. Pour les raisons présentées dans la description des acteurs, l'intervenant doit souvent planifier des horaires plus ou moins détaillés accomplissant la même tâche pour un ou plusieurs patients. Il doit présentement générer des horaires différents pour chaque cas. Un planificateur permettant de présenter un horaire accomplissant les mêmes activités mais avec des niveaux de granularité différents permettrait à l'intervenant de réutiliser le même horaire plusieurs fois.

1.5.2.2 Contraintes temporelles

L'intervenant considère le temps comme une grille horaire représentant une semaine. La durée des blocs horaires peut varier, mais des blocs discrets de 15 minutes sont généralement utilisés. Une durée est associée à chaque activité planifiée. Certaines activités peuvent être considérées comme ayant une durée inférieure à un bloc horaire, et dans certains cas, il est possible de planifier plusieurs activités dans le même bloc horaire. Les contraintes temporelles suivantes doivent aussi pouvoir être exprimées :

- une tâche doit être planifiée à un moment précis dans la grille horaire
- une tâche doit être planifiée dans un intervalle particulier (jour de la semaine, soir, matin, etc.)
- une tâche doit être planifiée avant/après une autre tâche
- deux tâches doivent être exécutées le même jour de la semaine
- deux tâches doivent être planifiées l'une immédiatement après l'autre

Le planificateur peut donc considérer le temps comme discret et représentant une période hebdomadaire (l'intervenant ne planifie pas sur une base mensuelle, par exemple). Le planificateur doit pouvoir permettre l'expression des types de contraintes temporelles illustrées.

1.5.2.3 Préférences associées au patient

Tel que présenté dans la description des acteurs, l'intervenant doit tenir compte de certaines préférences associées au patient lors du processus de planification. Bien qu'il ne soit pas absolument nécessaire pour l'intervenant de respecter ces préférences lors de la planification d'un horaire pour un patient, il est souhaitable qu'il le fasse. Un horaire respectant les préférences associées à un patient a en effet beaucoup plus de chance d'être respecté par ce dernier. Le planificateur doit donc pouvoir exprimer cette notion de préférences, soit une contrainte qu'il doit tenter de respecter, mais qui peut être relaxée s'il est impossible de

générer un horaire la respectant. Les contraintes temporelles mentionnées plus haut doivent pouvoir être exprimées en tant que préférences. La présence d'une sous-tâche dans une hiérarchie de tâches doit aussi pouvoir être exprimée comme une préférence. Par exemple, l'intervenant sait que son patient éprouve des problèmes de concentration. Il va d'abord tenter d'inclure une période de répit entre deux activités considérées comme difficiles, afin de reposer son patient, s'il est possible de le faire. S'il est impossible d'inclure une période de répit, l'intervenant fournira à son patient un horaire contenant seulement les deux activités considérées comme difficiles. L'horaire accomplira tout de même les activités désirées, mais aura moins de chance d'être respecté par le patient.

Le planificateur doit se comporter de manière similaire, soit d'abord tenter de respecter toutes les préférences spécifiées, puis graduellement les lever si elles sont impossibles à respecter.

1.5.2.4 Autres considérations

Les performances quant au temps d'exécution du planificateur ne sont pas un facteur déterminant. L'intervenant planifie en effet les horaires de ses patients sur une base hebdomadaire, jamais en tant réel ou dans un contexte qui nécessite un temps de réponse extrêmement court. Les performances doivent néanmoins rester raisonnables.

Les horaires générés par le planificateur doivent pouvoir être automatiquement exportés vers le système mobile MOBUS, sans devoir être modifiés manuellement. De plus, l'insertion du planificateur dans l'architecture MOBUS ne doit avoir aucune influence sur les services déjà présents, tant du coté de l'intervenant que de celui du patient. MOBUS est déjà déployé en milieu clinique, et l'insertion d'un nouveau service doit être complètement transparente pour le patient. L'intervenant doit pouvoir continuer à utiliser les services existants de la même manière.

Chapitre 2

État de l'art

Tel qu'énoncé dans la définition de la problématique, nous désirons ajouter à l'orthèse cognitive MOBUS un service de planification automatique. Il existe cependant d'autres orthèses implémentant des services similaires ou connexes fondés sur une relation de type intervenant-patient. Examinons d'abord ces orthèses afin de déterminer si l'utilisation de MOBUS est toujours justifiable dans le présent contexte.

2.1 Orthèses basées sur la planification

2.1.1 MEMOS

MEMOS est une application fondée sur le principe client-serveur. Le client est déployé sur un téléphone cellulaire assigné au patient. L'intervenant définit des activités pour son patient en utilisant une définition de langage *Extensible Markup Language (XML)* [17]. L'intervenant peut également définir une rétroaction au patient lors de la complétion d'une activité, ainsi qu'une série de rappels qui doivent être transmis au patient par rapport à une activité. Le patient confirme la complétion de ses activités via son client, et peut reporter une activité à une date ultérieure, s'il le désire. Le processus de planification survient lorsqu'une activité est négligée; MEMOS tente alors de repositionner l'activité en fonction de sa nature, sa durée et de l'horaire du patient. MEMOS utilise des techniques de planification spécifiquement conçues pour cette application. Une description complète est disponible en [32]. Les notions de granularité d'horaires ou de préférences associées à un patient ne sont pas prises en considération. MEMOS permet cependant une gestion très efficace des activités négligées par le patient.

2.1.2 PEAT – Planning and Execution Assistant Trainer

PEAT est une application de gestion d'activités destinée à être déployée sur une plate-forme *Pocket PC* qui met en place des services de planification automatique. Une liste d'activités devant être planifiées est spécifiée sous la forme de *scripts*, auxquels sont associées des contraintes de durée et de positionnement de ces activités (exemple: telle activité dure 45 minutes et doit être effectuée entre 8h00 et 9h00). Les scripts peuvent être spécifiés tant par l'intervenant que par le patient. Lorsque les scripts sont correctement définis, le système génère ensuite un horaire quotidien pour le patient. L'horaire respecte les contraintes spécifiées, si possible. PEAT offre également des services d'assistance visuelle et auditive au patient et permet le monitoring de la complétion d'activités. Des services de re-planification automatique sont aussi fournis; si certaines activités planifiées à un horaire ne sont pas complétées, PEAT tente de re-planifier un horaire quotidien les incluant. Une description complète est disponible en [14]. La planification est basée sur un système d'IA développé à la NASA et nommé *PROgram Planning and Execution Language* [13]. Ce système de planification est basé sur les buts à atteindre et a été conçu pour minimiser l'intervention humaine. Les services de re-planification de PEAT sont particulièrement intéressants, notamment le fait que la re-planification prend place sur la plate-forme mobile assignée au patient, ce qui lui fournit une autonomie supplémentaire. Certains facteurs doivent cependant être pris en considération dans un contexte où l'on minimise l'interaction humaine. Ces considérations font l'objet d'une discussion au chapitre 5. Les notions de granularité d'horaires ou de préférences associées à un patient ne sont pas prises en considération.

2.1.3 Autominder

Autominder est une orthèse cognitive basée sur une philosophie différente de celles mentionnées jusqu'ici. Autominder ne cherche pas à fournir un horaire de type agenda au patient. Le système vise plutôt à analyser l'exécution d'un plan par le patient en temps réel, puis à lui fournir des rappels et à le motiver à accomplir les activités pour lesquelles Autominder a détecté la possibilité de non complétion, ou une autre forme de situation problématique prenant en considération la définition d'horaire du patient et la nature de son

déficit cognitif. Autominder a d'abord été conçu pour être déployé sur un robot mobile, dans le cadre du projet *Nursebot* [26], mais le système pourrait clairement être adapté à d'autres plates-formes, telles que des appareils portables [25]. Les mécanismes de description des troubles cognitifs d'Autominder peuvent s'avérer intéressants, en particulier au niveau de la spécification de préférences associées au patient. Aucune notion de granularité au niveau de l'horaire n'est prise en considération.

2.1.4 Conclusion

Toutes les orthèses cognitives mentionnées ci-haut mettent l'accent sur la planification d'un horaire pour le patient, mais n'accordent que peu d'attention à soutenir l'intervenant dans un contexte d'élaboration de plan en initiative mixte. Bien que la gestion de préférences soit supportée dans certaines d'entre elles, il reste qu'aucun mécanisme de profilage de patients n'est mis en place, et que les préférences ne peuvent être gérées sur une base patient par patient. L'orthèse cognitive MOBUS a quant à elle été fondée sur l'hypothèse que l'intervenant devra fournir de l'assistance à plusieurs patients ayant des profils différents. Les infrastructures requises pour en faire la gestion sont déjà en place. De plus, MOBUS est déjà déployé dans la population cible de notre problématique, et cette dernière est déjà à l'aise avec son utilisation. Il est donc logique de considérer MOBUS comme orthèse cognitive dans le présent contexte.

2.2 Planificateurs

Plusieurs types d'algorithmes peuvent être utilisés pour effectuer la planification d'un horaire de type agenda. Examinons les solutions classiques, ainsi que leur forces et faiblesses par rapport à la problématique énoncée.

2.2.1 Exploration de l'espace d'états

Les algorithmes d'exploration de l'espace d'états tel que *A-star* [29] et *Forward/Backward Search* [31] peuvent être utilisés pour effectuer une planification. Il suffit de définir une série d'opérateurs représentant des actions, un état initial et un état but à atteindre, puis de lancer le

processus d'exploration. Des mécanismes d'heuristique peuvent accélérer l'exploration. Des planificateurs ont déjà été construits sur ces techniques, par exemple *STRIPS* [5].

Ces techniques ont l'avantage d'être extrêmement simples à implémenter. Cependant, elles sont généralement peu performantes par rapport aux autres techniques présentées. De plus, il peut être difficile d'exprimer les problèmes du présent contexte en une série d'opérateurs accomplissant un but, puisque les buts sont implicites et peu intéressants pour l'intervenant. Le concept de préférence y est également difficile à exprimer.

2.2.2 Problèmes à satisfaction de contraintes (*constraint satisfaction problems*)

On peut définir un problème à satisfaction de contraintes (CSP) comme une série de variables libres auxquelles un certain nombre de valeurs peuvent être affectées. Des contraintes peuvent ensuite être définies pour chaque variable [33]. Résoudre le problème revient à trouver une assignation de valeurs qui respecte toutes les contraintes spécifiées. Des techniques d'exploration très performantes existent pour résoudre ce type de problèmes, par exemple la propagation de contraintes *Forward Checking*, la sélection *Minimum Remaining Values* et la sélection *Least Constraining Value* [30].

La génération d'une grille horaire peut aisément se réduire à un CSP, il suffit de définir une série de variables représentant les blocs de la grille horaire, et des activités pouvant les occuper. On spécifie ensuite les contraintes pour chaque activité.

Des systèmes de gestion d'agenda ont déjà été construits à partir de ces techniques, telles que *SelfPlanner* [27].

Les techniques de résolution de CSP sont très bien adaptées à la recherche d'une solution respectant des contraintes et elles sont relativement simples à implémenter. Dans le présent contexte, elles peuvent être adaptées pour admettre des contraintes optionnelles (préférences).

Cependant, les techniques de résolution de CSP n'effectuent pas de planification en tant que telle, elles ne font qu'assigner des valeurs à des variables en cherchant à respecter les

contraintes spécifiées. Il peut être difficile d'exprimer certaines situations du présent contexte.

2.2.3 Planification avec logique temporelle (*temporal logic planning*)

La planification avec logique temporelle (TLP) est une technique de planification basée sur la technique d'exploration de l'espace d'états *Forward Search* [1]. TLP met en place un mécanisme permettant d'orienter la recherche. Des stratégies de recherche sont définies au moyen d'expressions de type *Linear Temporal Logic*. Lors de la recherche, ces expressions sont évaluées afin de déterminer si l'exploration peut continuer à progresser à partir d'un état, i.e. s'il est possible de respecter les stratégies spécifiées. L'exploration des sous-espaces d'états respectant les stratégies spécifiées est favorisée, ce qui mène à la découverte d'un plan plus rapidement qu'une exploration non orientée.

TLPlan [2] est un planificateur très performant qui a été construit à partir de ces techniques.

De par le fait qu'elles reposent sur la logique temporelle, les techniques de planification TLP peuvent naturellement être étendues pour faire la gestion de contraintes temporelles. Une solution a également été proposée pour faire la gestion de préférences [3]. Cette solution a l'intéressante particularité de trouver un plan respectant une quantité optimale de préférences spécifiées.

Cependant, les performances remarquables des techniques de planification TLP reposent sur une bonne définition de stratégie de recherche. En l'absence d'une bonne stratégie, les performances sont réduites à celles des techniques classiques d'exploration de l'espace d'état. De plus, les techniques de planification TLP restent orientées sur la recherche d'un état but; dans le présent contexte la planification orientée sur les buts est peu intéressante car elle va à contresens du raisonnement naturel de l'intervenant.

2.2.4 Planification avec réseaux de tâches hiérarchisées (*hierarchal task networks*)

La planification avec réseaux de tâches hiérarchisées (HTN) est une technique de planification qui présente la particularité intéressante d'exprimer ses opérateurs de planification comme une série de méthodes qui peuvent être appliquées à des tâches que l'on désire planifier [8]. Le processus de planification applique une méthode à une tâche, ce qui la décompose en sous-tâches qui doivent elles-mêmes être planifiées, puis applique d'autres méthodes à ces sous-tâches et ainsi de suite jusqu'à ce qu'il ne reste aucune tâche à décomposer, et qu'un plan soit par conséquent trouvé. Des planificateurs complets basés sur la planification HTN ont déjà été construits, notamment pour considérer des problèmes génériques tel que *SHOP2* [20], ou pour considérer des problèmes précis incluant des contraintes temporelles complexes tel que *ConfPlan* [4].

Le processus de décomposition d'une tâche de haut niveau en sous-tâches qui doivent être à leur tour décomposées est remarquablement proche du processus de planification manuel de l'intervenant tel que présenté en 1.5.2.1.

HTN offre également un mécanisme naturel pour générer des plans avec divers niveaux de granularité. Lors du processus de planification, lorsqu'une méthode est appliquée à une tâche, les sous-tâches en résultant peuvent être marquées de la tâche ayant provoqué leur création. Ceci est répété au fur et à mesure du processus, de sorte que le plan final contiendra des opérateurs qui peuvent être associés aux tâches de haut niveau ayant mené à leur insertion dans le plan. Avec cette information en main, il est désormais possible de présenter le plan avec un niveau configurable de granularité, en regroupant les opérateurs selon une ou plusieurs tâches de haut niveau. Ceci permet de réutiliser le même plan dans divers scénarios.

De plus, l'algorithme HTN peut être facilement étendu pour faire la gestion de préférences en considérant ces dernières comme des priorités dans la décomposition de tâche. À la définition d'une méthode, on peut marquer une sous-tâche de sorte que le planificateur tentera seulement de la décomposer s'il est possible de le faire. Lors du processus de planification,

une tâche ainsi marquée sera d'abord incluse dans la décomposition, mais si cette décomposition mène éventuellement à un échec de la planification, un retour en arrière sera alors effectué et une nouvelle décomposition n'incluant pas la tâche marquée sera générée. Le même raisonnement peut être appliqué à la spécification de contraintes.

Il faut cependant noter que la complexité de l'algorithme HTN peut rendre l'implémentation de ces techniques de planification plus lourde que les autres approches mentionnées. La spécification des hiérarchies de tâches définissant bien un problème de planification peut également s'avérer lourde.

2.2.5 Conclusion

En considérant la présente problématique, le choix d'une technique de planification HTN comme base du service de planification devant être intégré à MOBUS apparaît comme une bonne décision. Il y a une forte corrélation entre le raisonnement de l'intervenant et le processus de planification HTN, ce qui devrait faciliter son utilisation. Le processus peut également être étendu pour accommoder les caractéristiques requises énoncées dans la problématique. Nous adoptons donc HTN comme base de recherche.

Chapitre 3

Planificateur

Nous avons présenté les techniques de planification HTN et les raisons pour lesquelles nous avons retenu ce type de solution au Chapitre 2. Nous présentons maintenant la réalisation du planificateur concret basé sur ce choix, soit les concepts de bases sur lesquels il est fondé, l'algorithme de planification qui en résulte, et finalement l'intégration du planificateur au système mobile MOBUS. La description de l'environnement de développement ainsi que le code du planificateur tel qu'implémenté sont fournis à l'annexe A

3.1 Concepts de base

Décrivons d'abord les concepts de base sur lesquels notre algorithme de planification est fondé. Nous développons le planificateur dans un environnement orienté objet (OO) [6,18], et nous supposons que le lecteur est familier avec ce type d'environnement.

3.1.1 Variable

Une *variable* est un objet pouvant prendre une valeur quelconque. Une variable est dite *instanciée* si une valeur lui est assignée. Une variable est dite *non instanciée* si aucune valeur ne lui est assignée.

3.1.2 État

Un *état* est un objet comprenant une description d'état et un certain nombre de paramètres (des variables instanciées ou non), décrivant une situation quelconque.

3.1.3 Tâche et primitive

Une *tâche* ou *primitive* est un objet comprenant un nom, un certain nombre de paramètres (des variables instanciées ou non), décrivant une action à accomplir. Une tâche est une action qui doit être décomposée en sous-tâches et/ou sous-primitives, alors qu'une primitive est une action qui ne requiert aucune décomposition pour être réalisée. Une primitive comprend également une liste de pré-conditions (une liste d'états devant être respectés pour pouvoir réaliser l'action), effets (une liste d'états qui seront générés par la réalisation de l'action), ainsi qu'une durée temporelle en nombre de blocs horaire (voir 3.1.6). La durée peut être considérée comme nulle (instantanée) ou infinie. Une primitive est dite complètement instanciée si toutes ses variables sont instanciées.

3.1.4 Relation

Une *relation* est un objet comprenant une description et un certain nombres de paramètres (des tâches ou primitives), décrivant une relation quelconque entre plusieurs tâches et primitives. Par exemple, une tâche A avant une primitive B, une tâche A le même jour qu'une primitive B, etc. Les relations implémentées dans la version actuelle du planificateur sont données à l'annexe E.

3.1.5 Méthode

Une *méthode* est un objet applicable à une tâche comprenant un nom, un paramètre (une tâche à décomposer), une liste de pré-conditions (une liste d'états devant être respectés pour pouvoir appliquer la méthode à la tâche en paramètre), une liste de sous-tâches et/ou sous-primitives décomposant la tâche en paramètre, ainsi qu'une liste de relations impliquant les sous-tâches et/ou sous-primitives. Lorsqu'une méthode est appliquée à une tâche, cette dernière est décomposée en sous-tâches et/ou sous-primitives spécifiées par la méthode, et les relations spécifiées par la méthode sont générées.

3.1.6 Grille horaire

Une *grille horaire* est un objet représentant la tranche de temps considérée par le planificateur. La grille horaire constitue une discrétisation du temps, soit un nombre de blocs horaire consécutifs représentant une certaine période. Chaque bloc horaire est considéré comme ayant la même durée que tous les autres. Les blocs d'une grille horaire peuvent contenir une ou plusieurs références à une primitive, indiquant la position de cette primitive dans la période de temps discrète. Un bloc horaire peut contenir plusieurs références à des primitives si ces dernières sont de durée nulle ou infinie. Un bloc horaire est considéré comme occupé s'il contient une référence à une primitive ayant une durée autre que nulle ou infinie.

3.1.7 Monde

Un *monde* est un objet comprenant un état (l'état courant du monde), une liste de tâches et de primitives, une grille horaire (indiquant la position des primitives du monde), ainsi qu'une liste de relations (les relations entre les tâches et primitives du monde). Une primitive peut être appliquée à un monde, ce qui modifie l'état courant du monde en fonction de la définition de la primitive appliquée. Une décomposition de tâche (le résultat d'une application d'une méthode sur une tâche) peut également être appliquée à un monde, ce qui ajoute les sous-tâches/sous-primitives de la décomposition ainsi que leurs relations au monde, et retire la tâche décomposée.

3.1.8 Plan

Un *plan* est un objet comprenant une liste de primitives et une grille horaire indiquant leur position dans le temps. Un plan peut être vide, i.e. ne contenir aucune primitive. Un plan peut être marqué comme étant un échec, auquel cas son contenu n'a aucune signification. Une primitive peut être ajoutée à un plan, générant ainsi un nouveau plan. L'ajout d'une primitive à un plan marqué comme un échec génère un autre plan marqué comme un échec.

3.1.9 Préférence

Une *préférence* est un statut qui peut être appliqué à une relation, une sous-tâche ou une primitive lors de la spécification d'une méthode. Un objet marqué du statut de préférence est considéré comme pouvant être ignoré si nécessaire lors du processus de décomposition de tâche. Si l'application d'une méthode à une tâche mène à un échec, le planificateur peut revenir en arrière et tenter d'appliquer la méthode de nouveau en ignorant un ou plusieurs objet(s) marqué(s) comme préférence. Si une préférence sur une relation est relaxée, celle-ci ne sera pas engendrée par la décomposition de la tâche. Si une préférence sur une sous-tâche ou sous-primitive est relaxée, celle-ci ainsi que toutes les relations lui faisant référence ne seront pas engendrées par la décomposition de la tâche.

Une priorité est associée à chaque préférence. Le planificateur relaxera les préférences d'une méthode dans l'ordre croissant, de la moins prioritaire à la plus prioritaire. Si plusieurs préférences ont le même niveau de priorité, elles sont toutes relaxées en même temps. Il est important de noter que le planificateur applique d'abord une méthode en incluant toutes les préférences, et tente uniquement de les relaxer si cela aboutit à un échec.

Considérons un exemple afin d'illustrer le comportement du planificateur par rapport aux préférences. Soit la définition de méthode suivante, qui décompose une tâche `(cleaning-apartment (patient))` réalisant le ménage de l'appartement d'un patient :

```
(clean-apartment
        decomposes
                (cleaning-apartment (patient))
        into
                (cleaning-living-room (patient))
                (cleaning-kitchen (patient))
                (cleaning-bedroom (patient))
                (pref take-rest (patient))
        relations
                (precedence cleaning-kitchen cleaning-living-room)
                (pref last take-rest))
```

La sous-tâche `(take-rest (patient))` est marquée comme une préférence, signifiant qu'elle peut être omise s'il est impossible de trouver un plan l'incluant. La relation `(last`

take-rest)) indiquant que la sous-tâche (take-rest (patient)) doit être effectuée après toutes les autres sous-tâches est également marquée comme une préférence, signifiant qu'elle peut être relaxée s'il est impossible de trouver un plan la respectant.

Supposons qu'au cours du processus de planification, le planificateur doive décomposer une tâche (cleaning-apartment (patient)). Il lui applique la méthode (clean-apartment) en respectant toutes les préférences. Le processus de planification se poursuit ensuite. Supposons maintenant que le processus échoue: il est incapable de trouver un plan à partir de cette décomposition, tel qu'illustré à la Figure 8.

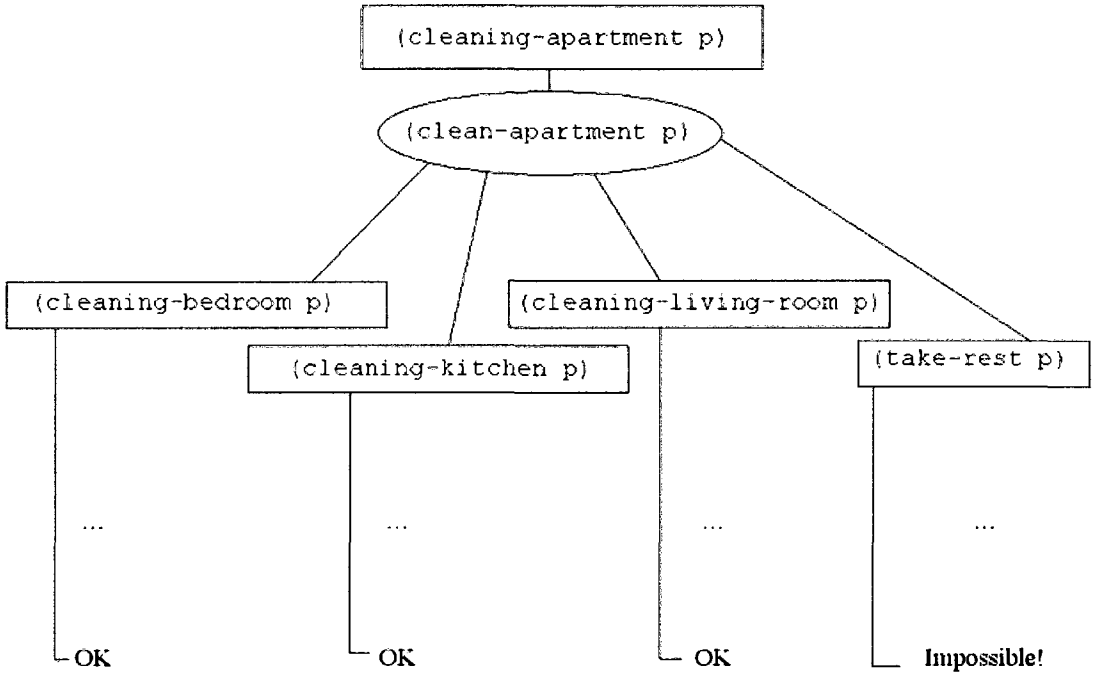


Figure 8 Une décomposition en sous-tâches menant à un échec

Le processus n'a pas été capable de trouver une décomposition pour la sous-tâche (take-rest (patient)). Avant de conclure que l'application de la méthode (clean-apartment) à la tâche (take-rest (patient)) mène à un échec, le planificateur examinera d'abord la méthode (clean-apartment) afin de déterminer s'il est possible de relaxer une préférence. Dans le présent cas, la méthode considérée contient effectivement deux préférences pouvant

être relaxées. Le planificateur choisi de relaxer la relation `(last take-rest))`, et relance le processus de planification, menant à la décomposition illustrée à la Figure 9.

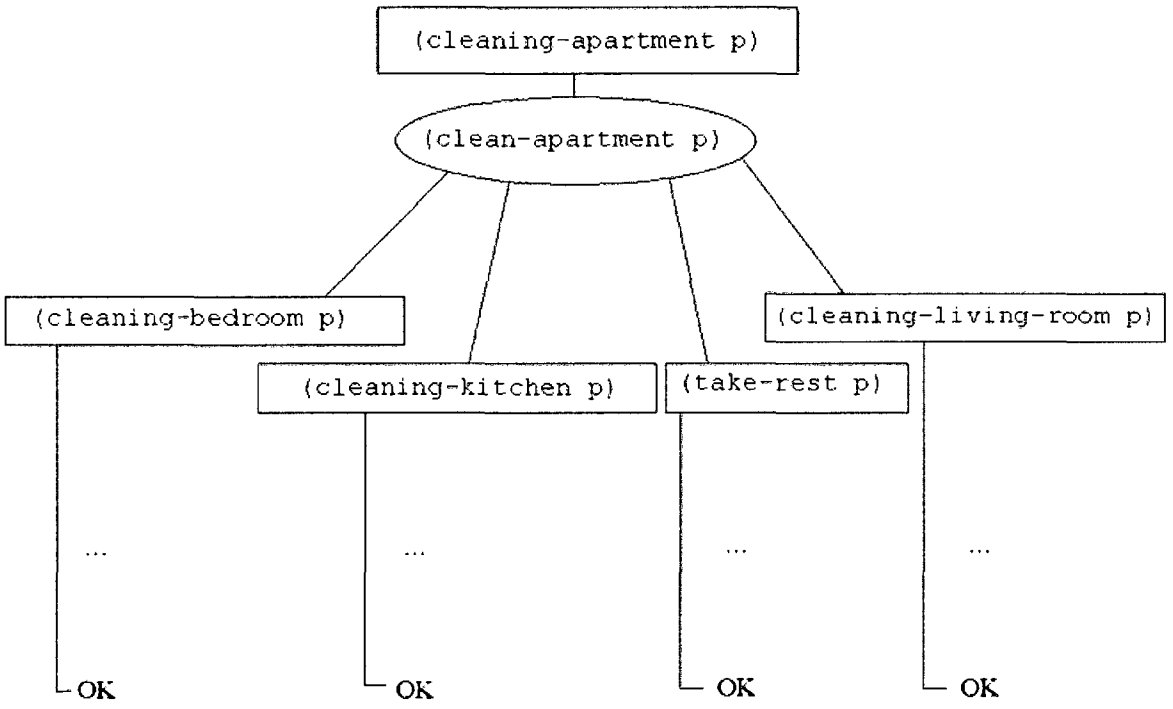


Figure 9 Une décomposition en sous-tâches menant à succès

Cette fois-ci, le processus de planification se poursuit et mène à un succès. Si le processus avait mené à un échec, la préférence sur la sous-tâche `(take-rest (patient))` aurait à son tour été relaxée, ce qui aurait mené à un plan dans lequel aucun repos n'est prévu lorsqu'on réalise la tâche `(cleaning-apartment (patient))`. Le planificateur choisit les préférences à relaxer en fonction d'un ordre de priorité spécifié à la définition du problème.

3.1.10 Granularité

Afin de pouvoir présenter un plan avec divers niveaux de *granularité*, le planificateur effectue un marquage des sous-tâches et sous-primitives engendrées par l'application d'une méthode à une tâche. Une sous-tâche ou sous-primitive est marquée comme provenant de la tâche l'ayant engendrée. Toutes les marques sont conservées et propagées vers le bas, de

sorte que pour toute primitive il est possible de déterminer la hiérarchie de tâche l'ayant engendrée.

3.2 Le planificateur

Les concepts de base ayant été définis, nous pouvons maintenant considérer le problème de planification et l'algorithme à lui appliquer.

3.2.1 Problème de planification

La définition d'un problème de planification, ou *domaine*, correspond à la spécification des entités suivantes :

- une configuration de grille horaire, soit la taille de la grille horaire et la tranche de temps qu'elle discrétise
- un bassin de méthodes disponibles, soit la spécification de toutes les méthodes pouvant être utilisées pour décomposer des tâches dans le processus de planification
- un monde initial, soit un monde représentant l'état initial et contenant la ou les tâches initiales à réaliser

3.2.2 Algorithme

Soit l'algorithme suivant qui retourne un plan et est appliqué au monde initial. Le bassin de méthodes disponibles est considéré comme global, constant et accessible à tout point de l'algorithme. La Figure 10 illustre l'algorithme.

```
PLANIFICATEUR(monde)
1       si monde ne contient aucune tâche
1.1             retourne plan vide

2       choisir U une tâche ou primitive de monde [*]

3       si U est une primitive
3.1             si U est complètement instanciée
3.1.1                   choisir S une position de la grille de monde [*]
3.1.2                   mondeSuivant ← monde avec U appliquée et
                                       positionée en S
3.1.3                   plan ← PLANIFICATEUR(mondeSuivant)
3.1.4                   retourne U union plan
3.2             sinon
3.2.1                   permute ← toutes les valeurs possibles pour
                                  les variables non instanciées de U
3.2.2                   choisir P un élement de permute [*]
3.2.3                   U' ← P appliqué à U
3.2.4                   choisir S une position de la grille de monde [*]
3.2.5                   mondeSuivant ← monde avec U' appliquée et
                                       positionée en S
3.2.6                   plan ← PLANIFICATEUR(mondeSuivant)
3.2.7                   retourne U union plan

4       si U est une tâche
4.1             applicable ← toutes les méthodes applicables à U
4.2             si applicable est vide
4.2.1                   retourne échec

4.3             choisir M une méthode de applicable [*]
4.4             mondeSuivant ← monde avec M appliqué à U
4.5             plan ← PLANIFICATEUR(mondeSuivant)

4.6             si plan est un échec
4.6.1                   choisir P une préférence de M [*]
4.6.2                   MRelaxée ← M avec la préférence P relaxée
4.6.3                   mondeSuivant ← monde avec MRelaxée appliquée à U
4.6.4                   retourne PLANIFICATEUR(mondeSuivant)
4.7             sinon
4.7.1                   retourne plan
```

Figure 10 Algorithme de planification

Les lignes marquées d'un [*] constituent des points de retour en arrière (*backtracking*), i.e. si le prochain plan généré par une sélection s'avère être un échec, une autre sélection sera effectuée et le processus sera relancé à partir de ce point. Le planificateur effectue les sélections dans un ordre non-déterministe par défaut, mais peut être configuré pour les faire de manière déterministe. Ce principe s'applique à toutes les sélections sauf à la sélection de

préférence en 4.6.1. Les préférences sont toujours relaxées dans leur ordre croissant de priorité, de la moins prioritaire à la plus prioritaire.

L'algorithme instancie les primitives non complètement instanciées en 3.2. Les valeurs possibles que peuvent prendre les variables d'une primitive sont inférées à partir du monde courant, et toutes les permutations possibles de ces valeurs sont ensuite calculées. L'instanciation ne se fait qu'au niveau des primitives, les tâches ayant des variables non instanciées sont décomposées sans être instanciées.

Le marquage des sous-tâches et sous-primitives est effectué aux lignes 4.4 et 4.6.3. Une sous-tâche ou primitive est marquée de la tâche qui l'a engendrée, et hérite également de toute les marques de cette dernière. Une primitive est donc marquée de toutes les tâches l'ayant éventuellement engendrée. Il est donc possible d'analyser un plan obtenu afin de présenter au patient un horaire regroupant des primitives provenant de la décomposition d'une même tâche. Notons que cette analyse est un processus effectué en dehors de l'algorithme.

3.3 Intégration au système mobile MOBUS

Le planificateur dispose des services requis pour exporter un plan au système mobile MOBUS automatiquement.

Un identifiant de patient ainsi qu'une date représentant le début de la période de planification doivent être fournis au planificateur. Le planificateur convertit chaque primitive d'un plan en une activité dont le format est reconnu par MOBUS, s'interface ensuite directement avec la base de données utilisée par MOBUS et y exporte les activités créées. Du point de vue du système mobile, il n'y a aucune différence entre les activités créées par le planificateur et celles entrées manuellement par l'intervenant. Les dates de début et de fin d'activité sont automatiquement inférées à partir de la date fournie, des positions dans la grille horaire et des durées des primitives du plan.

Le planificateur peut être configuré pour présenter un plan selon une granularité correspondant à une certaine tâche. Les primitives générées par cette tâche sont alors remplacées par une nouvelle primitive correspondant à la tâche, dont la position et durée sont inférées à partir des primitives originales.

Un plan exporté vers MOBUS de la sorte peut être consulté par l'intervenant et le patient, avec les présents clients MOBUS respectifs, sans aucune modification. Les activités apparaissent de la même manière, qu'elles aient été entrées manuellement par l'intervenant ou générées par le planificateur.

Des exemples détaillés sont présentés au Chapitre 4.

Chapitre 4

Résultats

Le planificateur implémenté a été soumis à des expérimentations destinées à évaluer la qualité des caractéristiques qu'il devait présenter ainsi que des besoins auxquels il devait répondre tels que présentés dans les chapitres précédents. Les domaines de planification ont d'abord été construits à partir d'un problème jouet, soit un cas théorique très simpliste destiné à vérifier le bon fonctionnement des fonctionnalités de base du planificateur (i.e. s'il existe un plan pour un domaine, peut-il le trouver, la gestion des préférences, de la granularité et de la planification horaire est-elle correctement effectuée, etc.). D'autres domaines ont ensuite été construits à partir de scénarios d'utilisation typiques récoltés auprès d'intervenants ayant participé à des expérimentations antérieures du système mobile MOBUS en milieu clinique.

4.1 Cas théorique

Nous avons débuté par soumettre le planificateur à un domaine de planification théorique classique : la construction d'une pile de blocs. Plusieurs algorithmes de planification font leurs premières preuves sur ce type de problème, qui peut facilement être adapté pour illustrer plusieurs types de contraintes de planification [11].

4.1.1 Domaine théorique

Nous considérons le domaine suivant qui permet de représenter des blocs de diverses couleurs et de construire une pile de trois blocs, selon certaines contraintes :

```
Primitives:
stackFromTable
        paramètres : A B
        // prend le bloc A sur la table et le dépose sur le bloc B

unstackFromBlock
        paramètres : A B
        // prend le bloc A sur le bloc B et le dépose sur la table

Méthodes:
StackTwoBlocks
        Décompose : (stackTwoBlocks A B)
        Paramètres : A B
        Sous-tâches : (stackFromTable A B)
        Relation : nil
        // empile le bloc A sur le bloc B

ClearBlockA
        Décompose : (clearBlock A)
        Paramètres : A
        Sous-tâches : nil
        Relation : nil
        // ne fait rien, le bloc A est déjà libre

ClearBlockB
        Décompose : (clearBlock A)
        Paramètres : A B
        Sous-tâches : (unstackFromBlock B A)
        Relation : nil
        // dépile le bloc B du bloc A, libérant ainsi le bloc A

Make3Pile
        Décompose : (make3Pile A B C)
        Paramètres : A B C
        Sous-tâches : (clearBlock A) (clearBlock B) (clearBlock C)
                      (stackTwoBlocks B C) (stackTwoBlocks A B)
        Relation : (before (stackTwoBlocks B C) (stackTwoBlocks A B))
                      (before (clearBlock A) (stackTwoBlocks B C))
                      (before (clearBlock B) (stackTwoBlocks B C))
                      (before (clearBlock C) (stackTwoBlocks B C))
        // construit une pile de trois blocs, A au dessus, B au milieu
        // et C à la base
```

Les pré-conditions et effets sont conséquents aux opérations effectuées et sont omis pour simplifier l'exemple. Le domaine complet tel qu'implémenté est fourni à l'annexe C.

Ce domaine extrêmement simple permet de créer une pile de trois blocs en s'assurant d'abord que les trois blocs sont libres, puis en empilant le deuxième sur le troisième et finalement le premier sur le deuxième. Notons que de par cette spécification, pour empiler deux blocs, le

bloc à déplacer doit se trouver sur la table. Rappelons qu'il ne s'agit pas d'un domaine complet ou permettant de trouver un plan optimal, simplement d'un outil qui permet de valider le bon fonctionnement du planificateur.

Nous construisons ensuite sept scénarios basés sur ce domaine. Pour chaque scénario, la configuration initiale du monde est telle qu'illustrée par la Figure 11.

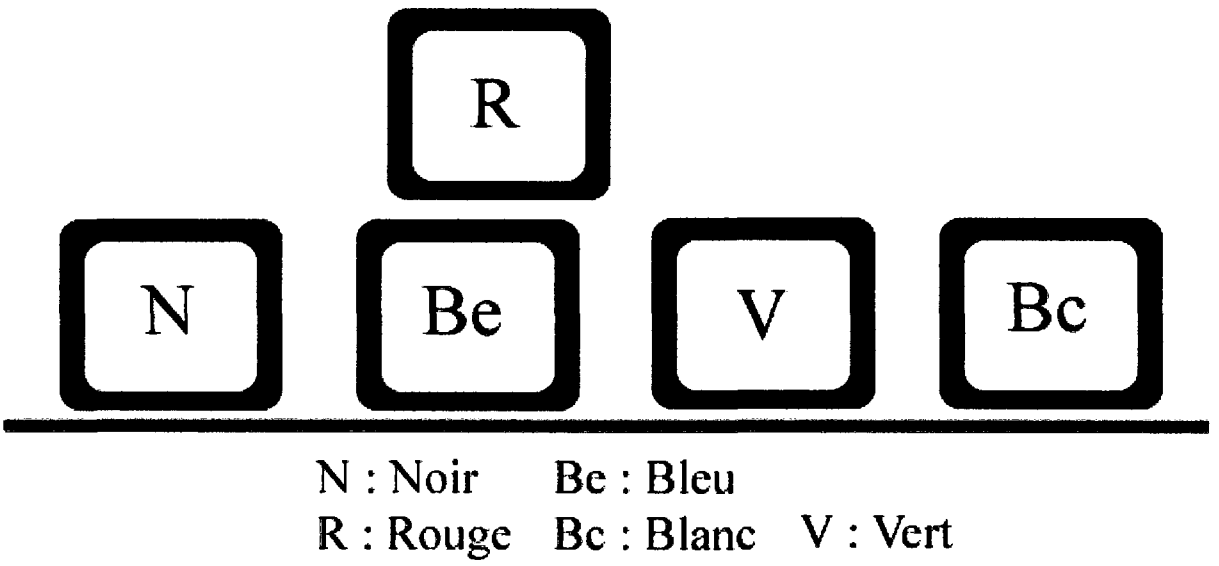


Figure 11 Configuration initiale du cas théorique

soit quatre blocs (blanc, bleu, noir et vert) reposant sur la table et un bloc rouge reposant sur le bloc bleu. Chaque scénario utilise une grille horaire constituée de 8 blocs. Cette grille horaire ne représente aucune tranche de temps discrétisée (i.e. une journée, une heure, etc.) mais simplement 8 blocs de temps discret que les primitives peuvent occuper. Ce domaine simpliste doublé de cette configuration initiale limite les plans possibles pour une tâche initiale, ce qui permet d'isoler facilement les caractéristiques du planificateur à valider.

Le temps d'exécution moyen pour 100 exécutions est fourni pour chaque scénario. L'algorithme est utilisé de manière non déterministe. L'environnement de test est décrit à

l'annexe B. Le surligneur dans les résultats indique qu'un bloc horaire est occupé par une primitive.

4.1.1.1 Scénario théorique 1

Nous vérifions d'abord que le planificateur trouve un plan valide s'il en existe un. La durée des primitives est fixée à un bloc horaire. La tâche initiale est fixée à `(make3Pile RED BLUE GREEN)`, soit faire une pile telle que le bloc vert est à la base, le bloc bleu repose sur le bloc vert et le bloc rouge repose au sommet de la pile. Nous obtenons le résultat typique :

```
Time Grid Contents:
0:
1:
2: (PRIMITIVE unstackFromBlock (&37 RED) (&33 BLUE))
3:
4: (PRIMITIVE stackFromTable (&112 BLUE) (&113 GREEN))
5:
6:
7: (PRIMITIVE stackFromTable (&132 RED) (&133 BLUE))
Found schedule in 0:00:00:032 (32 ms)
```

qui constitue en effet le seul plan valide pour réaliser la pile désirée à partir de la configuration initiale.

Temps d'exécution moyen: 32 ms. Sans aucune contrainte et avec des blocs spécifiés, le planificateur trouve rapidement un horaire valide.

4.1.1.2 Scénario théorique 2

Nous vérifions ensuite la capacité du planificateur à gérer les variables non instanciées. La durée des primitives est fixée à un bloc horaire. La tâche initiale est fixée à `(make3Pile RED ? ?)` soit faire une pile de trois blocs dont le bloc du dessus est rouge Nous obtenons le résultat typique :

```
Time Grid Contents:
0:
1:
2:
3:
4: (PRIMITIVE unstackFromBlock (&5B RED) (&58 BLUE))
5: (PRIMITIVE stackFromTable (&80 WHITE) (&81 BLUE))
```

```
6: (PRIMITIVE stackFromTable (&C3 RED) (&C4 WHITE))
7:
Found schedule in 0:00:00:140 (140 ms)
```

Les piles construites présentent toujours le bloc rouge au dessus et par conséquent la première primitive de chaque plan généré est toujours `(unstackFromBlock RED BLUE)`, ce qui est cohérent avec le domaine, pour lequel il est en effet impossible d'empiler un bloc ne reposant pas sur la table sur un autre bloc. La couleur des deux autres blocs est aléatoire tel qu'attendu de la part d'un algorithme non déterministe.

Temps d'exécution moyen: 114 ms. Une augmentation rationnelle étant donné que le planificateur doit désormais examiner le monde et en déduire des permutations d'instances pour les variables non instanciées.

4.1.1.3 Scénario théorique 3

Nous vérifions maintenant la capacité du planificateur à effectuer l'exploration complète de l'espace de décomposition et à réaliser qu'aucun plan valide n'existe. La durée des primitives est fixée à un bloc horaire. La tâche initiale est fixée à `(make3Pile RED BLUE RED)` soit faire une pile de trois blocs avec 2 blocs rouges. Nous obtenons le même résultat à chaque exécution :

```
PLAN: FAILED
Found schedule in 0:00:00:359 (359 ms)
```

La tâche initiale est en effet impossible à réaliser, le monde initial ne contenant qu'un seul bloc rouge. Il n'existe aucun plan valide, et le planificateur le réalise.

Temps d'exécution moyen: 357 ms. Le planificateur doit faire l'exploration complète de l'espace de décomposition pour réaliser qu'aucun plan valide n'existe. On remarque que ce temps est considérable pour un espace si restreint. Une discussion sur les performances suivra au Chapitre 5.

4.1.1.4 Scénario théorique 4

Nous vérifions maintenant la capacité du planificateur à faire la gestion correcte de la grille horaire. Nous modifions le domaine de sorte que les seuls plans valides réalisant la tâche initiale comportent des primitives dont la durée cumulative excèdent la taille de la grille horaire. La durée des primitives est fixée à trois blocs horaires. La tâche initiale est fixée à `(make3Pile RED ? ?)` soit faire une pile de trois blocs dont le bloc du dessus est rouge. Nous obtenons le même résultat à chaque exécution :

```
PLAN: FAILED
Found schedule in 0:00:00:922 (922 ms)
```

Bien qu'il existe un plan valide pour réaliser la tâche initiale, ce dernier doit contenir au minimum trois primitives en fonction de la configuration initiale du monde. Pour former une pile présentant le bloc rouge sur le dessus, il faudra obligatoirement dépiler le bloc rouge du bloc bleu (car on ne peut déplacer un bloc pour l'empiler sur un autre que s'il repose sur la table), et par conséquent, le plus court plan valide contient trois primitives, soit :

```
(unstackFromBlock RED BLUE)
(stackFromTable A B)
(stackFromTable RED A)
```

avec `A` et `B` deux blocs de couleurs différentes autre que `RED`. Trois primitives d'une durée de trois blocs horaire ne peuvent être simultanément placées dans une grille ne contenant que huit blocs horaire. Il n'existe donc pas de plan qui puisse réaliser la tâche initiale et le planificateur le réalise.

Temps d'exécution moyen: 920 ms. Le planificateur doit faire l'exploration complète de l'espace de décomposition ainsi que des permutations d'instances pour les variables non instanciées afin de réaliser qu'aucun plan valide n'existe, augmentant le temps d'exécution par rapport au scénario 3.

Remarquons que si ce scénario est modifié pour utiliser une grille horaire de 9 blocs (ou plus), le planificateur trouve un plan valide, par exemple :

```
Time Grid Contents:
0: (PRIMITIVE unstackFromBlock (&3 RED) (&56 BLUE))
1:
2:
3: (PRIMITIVE stackFromTable (&24C BLUE) (&24D GREEN))
4:
5:
6: (PRIMITIVE stackFromTable (&28F RED) (&290 BLUE))
7:
8:
Found schedule in 0:00:00:078 (78 ms)
```

4.1.1.5 Scénario théorique 5

Nous vérifions une fois de plus la capacité du planificateur à faire la gestion correcte de la grille horaire. Le domaine reste le même qu'au scénario 4.1.1.4, mais cette fois-ci il existe des plans valides réalisant la tâche initiale dont la durée cumulative des primitives est inférieure à la taille de la grille horaire. La durée des primitives est fixée à trois blocs horaires. La tâche initiale est fixée à `(make3Pile ? ? ?)` soit faire une pile de trois blocs quelconques. Nous obtenons le résultat typique :

```
Time Grid Contents:
0:
1: (PRIMITIVE stackFromTable (&132C BLACK) (&132D WHITE))
2:
3:
4: (PRIMITIVE stackFromTable (&1374 GREEN) (&1375 BLACK))
5:
6:
7:
Found schedule in 0:00:00:219 (219 ms)
```

Tel que vu au scénario 4, il est impossible de générer un plan contenant trois primitives avec cette configuration, et par conséquent aucun plan généré ne place le bloc rouge au-dessus de la pile. Le planificateur trouve cependant un plan valide pour réaliser une pile de trois blocs qu'il peut insérer dans la grille horaire. Les blocs sélectionnés sont aléatoires tel qu'attendu.

Temps d'exécution moyen: 197 ms. Le planificateur risque d'explorer un grand nombre de décompositions qui mèneront vers des plans où il faudra empiler le bloc rouge, et qui contiendront donc trois primitives. Ces explorations échoueront toutes, ce qui explique l'augmentation du temps d'exécution par rapport au scénario 2, très similaire.

4.1.1.6 Scénario théorique 6

Nous vérifions maintenant la capacité du planificateur à gérer la notion de préférence. La durée des primitives est fixée à un bloc horaire. Nous spécifions également une préférence temporelle de haute priorité (mais pouvant être relaxée) au niveau de la méthode `Make3Pile`, modifiant les relations de ses sous-tâches comme suit :

```
Relation : (before (stackTwoBlocks B C) (stackTwoBlocks A B))
               (before (clearBlock A) (stackTwoBlocks B C))
               (before (clearBlock B) (stackTwoBlocks B C))
               (before (clearBlock C) (stackTwoBlocks B C))
               (BeginAt (stackTwoBlocks A B) 2 highPriority)
```

Cette nouvelle relation indique que les primitives engendrées par la décomposition de `(stackTwoBlocks A B)` devraient commencer à la position du bloc horaire 2, si possible. La tâche initiale est fixée à `(make3Pile RED BLUE GREEN)`. Nous obtenons le même résultat à chaque exécution :

```
Time Grid Contents:
0: (PRIMITIVE unstackFromBlock (&48 RED) (&44 BLUE))
1: (PRIMITIVE stackFromTable (&37F BLUE) (&380 GREEN))
2: (PRIMITIVE stackFromTable (&3DB RED) (&3DC BLUE))
3:
4:
5:
6:
7:
Found schedule in 0:00:00:094 (94 ms)
```

Il s'agit en effet du seul horaire possible respectant cette préférence. Comme au scénario 1, le seul plan possible pour réaliser la tâche initiale est :

```
(unstackFromBlock RED BLUE)
(stackFromTable BLUE GREEN)
(stackFromTable RED BLUE)
```

dans cet ordre. En considérant que chaque primitive a une durée d'un bloc horaire, si on désire placer `(stackFromTable RED BLUE)` au bloc horaire 2, il s'en suit que `(stackFromTable BLUE GREEN)` doit être placé au bloc horaire 1 et `(unstackFromBlock RED BLUE)` doit être placé au bloc horaire 0. Le planificateur réalise qu'il existe un horaire

respectant l'ordonnancement du plan et la préférence temporelle spécifiée, et est en mesure de le trouver.

Temps d'exécution moyen: 70 ms. Le planificateur requiert plus de temps qu'au scénario 1 pour générer un plan similaire. Ceci provient du fait qu'il doive potentiellement considérer plusieurs positions horaires pour les primitives qui échoueront avant de trouver la configuration respectant la préférence.

4.1.1.7 Scénario théorique 7

Nous vérifions finalement la capacité du planificateur à relaxer une préférence, si nécessaire. La durée des primitives est fixée à un bloc horaire. Nous spécifions également une préférence temporelle de haute priorité (mais pouvant être relaxée) au niveau de la méthode `Make3Pile`, modifiant les relations de ses sous-tâches comme suit :

```
Relation : (before (stackTwoBlocks B C) (stackTwoBlocks A B))
               (before (clearBlock A) (stackTwoBlocks B C))
               (before (clearBlock B) (stackTwoBlocks B C))
               (before (clearBlock C) (stackTwoBlocks B C))
               (BeginAt (stackTwoBlocks A B) 1 highPriority)
```

Cette nouvelle relation indique que les primitives engendrées par la décomposition de `(stackTwoBlocks A B)` devraient commencer à la position du bloc horaire 1, si possible. La tâche initiale est fixée à `(make3Pile RED BLUE GREEN)`. Nous obtenons le résultat typique :

```
Time Grid Contents:
0: (PRIMITIVE unstackFromBlock (&3 RED) (&400C BLUE))
1:
2:
3: (PRIMITIVE stackFromTable (&4064 BLUE) (&4065 GREEN))
4:
5:
6: (PRIMITIVE stackFromTable (&40A2 RED) (&40A3 BLUE))
7:
Found schedule in 0:00:00:391 (391 ms)
```

Tel que vu au scénario 6, il est en effet impossible de positionner `(stackFromTable RED BLUE)` au bloc horaire 1, car les deux autres primitives doivent être positionnées antérieurement. Le planificateur réalise qu'il ne peut respecter la préférence spécifiée. Par

conséquent, il la relaxe pour trouver un plan valide réalisant la tâche initiale, mais ne respectant pas la préférence.

Temps d'exécution moyen: 331 ms. Le planificateur considère d'abord la préférence comme une contrainte obligatoire, et doit donc faire l'exploration complète du sous-espace de décomposition qui la respecte avant de la relaxer, d'où l'augmentation du temps d'exécution par rapport au scénario 6.

4.1.2 Résumé des résultats : cas théorique

Le Tableau 1 résume les résultats obtenus pour le cas théorique.

Scénario	Caractéristique validée	Résultat	Comportement adéquat ?	Performance moyenne
1	Découverte d'un plan valide variables complètement instanciées	Trouve le seul plan valide possible, génère un horaire correct	oui	32 ms
2	Découverte d'un plan valide variables non instanciées	Trouve un plan valide, variables correctement instanciées, génère un horaire correct	oui	114 ms
3	Détection de l'absence de plan valide tâche initiale irréalisable	Ne trouve aucun plan valide (échec de planification)	oui	357 ms
4	Détection de l'absence de plan valide tâche initiale réalisable, mais horaire impossible	Ne trouve aucun plan valide (échec de planification)	oui	920 ms
5	Gestion horaire tâche initiale réalisable, contraintes horaires	Trouve un plan valide, génère un horaire correct	oui	197 ms
6	Gestion de préférences préférence accommodable	Trouve un plan valide, génère un horaire respectant la préférence	oui	94 ms
7	Gestion de préférences préférence non accommodable	Trouve un plan valide, génère un horaire ne respectant pas la préférence	oui	331 ms

Tableau 1 Résumé des résultats - cas théorique

Notons que la gestion de la granularité sera validée lors d'une expérimentation avec le premier scénario clinique décrit plus bas, qui s'y prête parfaitement.

4.2 Cas clinique

Ayant validé le comportement de base du planificateur grâce aux tests effectués avec le cas théorique, nous avons ensuite construit des domaines correspondant à certains scénarios d'utilisation typique tels que recueillis auprès d'intervenants au cours de rencontres dans la phase d'identification des besoins du patient et de l'intervenant (présenté au Chapitre 1).

Il est important de considérer la représentation discrète du temps lors de la configuration du planificateur. Comme l'espace d'états au cours du processus de planification croît (entre autres) en fonction de la taille de la grille horaire, il est important de choisir la taille de grille minimale qui représente une discrétisation du temps suffisamment précise pour répondre aux besoins de l'intervenant. Si la taille de la grille est trop petite, l'intervenant perd la capacité de spécifier des horaires précis, et si la taille est inutilement grande, les performances du planificateur ne sont plus acceptables. Les scénarios recueillis indiquent que l'intervenant planifie les horaires de ses patients pour une période couvrant une semaine. Les activités sont habituellement découpées en blocs de 15 minutes, ce qui semble fournir une précision suffisante pour la plupart des cas. Nous utilisons donc une grille horaire représentant une semaine, soit 4 blocs par heure, 24 heures par jour, 7 jours par semaine pour un total de 672 blocs. La semaine débute au bloc 0, soit le dimanche 0h00 et se termine au bloc 671, soit le samedi 23h45.

Trois domaines représentant un problème spécifique ont ensuite été construits. D'abord, la planification d'un déplacement entre le lieu de résidence et un centre de jour. Ensuite, la planification d'une journée tenant compte de la prise de médicaments, des repas, d'une activité de loisirs et d'un rendez-vous à une clinique. Finalement, la planification d'une semaine complète typique tenant compte de plusieurs types d'activités. Des scénarios ont ensuite été construits pour chaque domaine afin de valider certains comportements du planificateur. Comme pour le cas théorique, le temps d'exécution moyen pour 100 exécutions est fourni pour chaque scénario. L'algorithme est utilisé de manière non déterministe. L'environnement de test est décrit à l'annexe B.

4.2.1 Domaine 1 : déplacement résidence – centre de jour

Ce domaine permet la spécification d'un déplacement de la résidence du patient jusqu'au centre de jour où il est attendu. Le domaine spécifie une hiérarchie de méthodes détaillée pour chaque étape du déplacement (prendre le tramway à partir de la résidence, marcher du tramway à l'édifice hébergeant le centre de jour, prendre la navette de retour jusqu'à la résidence). Cette activité est planifiée pour le dimanche et doit débuter à 9h00. Le domaine complet tel qu'implémenté est fourni à l'annexe C. Ce type d'activité correspond au cas typique de réadaptation, pour lequel on désire progressivement diminuer la quantité d'assistance fournie au patient, et permet donc de valider la capacité du planificateur à gérer la granularité. Notons que seule la partie de la grille horaire meublée par le plan est fournie pour des fins de clarté. Le surligneur indique qu'un bloc est occupé par une primitive/tâche. Contrairement au cas théorique, on associe une valeur temporelle à chaque bloc horaire (une heure de la journée).

4.2.1.1 Scénario 1 : détail complet

Nous configurons d'abord le planificateur pour générer un horaire avec le plus haut niveau de granularité possible, soit le détail complet de l'activité. Nous obtenons le résultat suivant :

```
NOT COLLAPSING ANYTHING -- FULL PLAN
WEEK SCHEDULE:

SUNDAY
  ...
  08:45:
  09:00: (PRIMITIVE Aller à l'arrêt Tram Université (&A PATIENT))
  09:15: (PRIMITIVE Prendre Tram direction de Université Claude Bernard
          (&F PATIENT))
  09:30:
  09:45: (PRIMITIVE Descendre à l'Arrêt Feyssine (&7 PATIENT))
  10:00: (PRIMITIVE Tourner à gauche sur Rue de la république
           (&28 PATIENT))
         (PRIMITIVE Tourner à droite sur la Place Jean Macé (&24 PATIENT))
         (PRIMITIVE Tourner à gauche sur la Rue Philippe Pinel
           (&1F PATIENT))
         (PRIMITIVE Aller jusqu'au n°72 de la Rue Philippe Pinel
           (&19 PATIENT))
         (PRIMITIVE Aller au local 505, 5eme étage (&17 PATIENT))
  10:15:
  10:30:
```

```
  10:45:
  11:00: (PRIMITIVE Aller à l'arrêt de la navette (&2F PATIENT))
         (PRIMITIVE Prendre navette (&2E PATIENT))
  11:15:
  ...
Found schedule in 0:00:00:614 (614 ms)
```

Les primitives complètes pour effectuer un déplacement de la résidence jusqu'au centre de jour sont incluses dans l'horaire. Remarquons également que cet horaire illustre la capacité du planificateur à faire la gestion de plusieurs primitives positionnées au même bloc horaire (10h00 et 11h00 dans le présent cas) et considérées comme ayant une durée nulle (inférieure à un bloc horaire). Dans l'état actuel du planificateur, deux types de primitives reçoivent une considération particulière par rapport au positionnement dans la grille horaire.

D'abord, un bloc horaire peut contenir une quantité illimitée de primitives ayant une durée nulle. Lors du processus de planification, on considère qu'un bloc horaire contenant une telle primitive peut en contenir d'autres du même type, mais aucun autre type de primitives (par exemple, une primitive ayant une durée non nulle). En d'autres termes, un bloc horaire contenant une primitive de durée nulle est considéré comme occupé, mais d'autres primitives de durée nulle peuvent y être positionnées. Cette considération est requise pour permettre la planification de tâches ayant une durée réelle inférieure à la durée d'un bloc horaire (généralement 15 minutes dans les utilisations cliniques observées). Sans ce traitement particulier, le planificateur considérerait simplement un bloc horaire contenant une telle primitive comme occupé, et ne pourrait y positionner d'autres primitives. Par exemple, dans le résultat précédent, les cinq primitives positionnées au bloc horaire de 10h00 devraient être positionnées dans des blocs horaires différents.

Ensuite, un bloc horaire contenant une primitive ayant une durée infinie (soit une durée particulière représentant le fait que la durée réelle de la primitive est inconnue ou sans importance) n'est pas considéré comme étant occupé. Un tel bloc peut contenir une autre primitive de durée non nulle ou une série de primitives de durée nulle (tel que présenté ci-haut) avant d'être considéré comme occupé. Un bloc horaire peut contenir plusieurs

primitives de durée infinie. Un tel bloc est toujours considéré comme libre. Cette considération permet la planification de tâches intemporelles ou dont la durée n'a aucune importance. Elles sont typiquement utilisées pour représenter le début d'une tâche dont la durée dépassera l'étendue de la grille horaire et qui peut être interrompue en tout temps, pour indiquer au patient de débuter la lecture d'un livre par exemple.

Temps d'exécution moyen: 625 ms.

4.2.1.2 Scénario 2 : détails de prise de tram omis

Nous configurons ensuite le planificateur pour générer un horaire omettant les détails nécessaires pour prendre le tram. Nous obtenons le résultat suivant :

```
COLLAPSING (Prendre tram jusqu'au centre)
WEEK SCHEDULE:

SUNDAY
  ...
  08:45:
  09:00: (PRIMITIVE Prendre tram jusqu'au centre (&32 PATIENT))
  09:15:
  09:30:
  09:45:
  10:00: (PRIMITIVE Tourner à gauche sur Rue de la république
           (&28 PATIENT))
         (PRIMITIVE Tourner à droite sur la Place Jean Macé (&24 PATIENT))
         (PRIMITIVE Tourner à gauche sur la Rue Philippe Pinel
           (&1F PATIENT))
         (PRIMITIVE Aller jusqu'au n°72 de la Rue Philippe Pinel
           (&19 PATIENT))
         (PRIMITIVE Aller au local 505, 5eme étage (&17 PATIENT))
  10:15:
  10:30:
  10:45:
  11:00: (PRIMITIVE Aller à l'arrêt de la navette (&2F PATIENT))
         (PRIMITIVE Prendre navette (&2E PATIENT))
  11:15:
  ...
Found schedule in 0:00:00:625 (625 ms)
```

Les primitives accomplissant la prise du tram ne sont plus incluses dans l'horaire. Ces primitives originellement engendrées par la décomposition de cette tâche sont remplacées par

celle-ci. La durée et position de la nouvelle tâche sont inférées à partir des durées et positions de ses primitives.

Temps d'exécution moyen: 625 ms.

4.2.1.3 Scénario 3 : détails de prise de tram, de marche et de prise de navette omis

Nous configurons ensuite le planificateur pour générer un horaire omettant les détails nécessaires pour prendre le tram, marcher jusqu'au centre de jour et prendre la navette de retour. Nous obtenons le résultat suivant :

```
COLLAPSING (Prendre tram jusqu'au centre), (Marcher jusqu'au centre) AND
(Prendre navette de retour)
WEEK SCHEDULE:

SUNDAY
  ...
  08:45:
  09:00: (PRIMITIVE Prendre tram jusqu'au centre (&32 PATIENT))
  09:15:
  09:30:
  09:45:
  10:00: (PRIMITIVE Marcher jusqu'au centre (&33 PATIENT))
  10:15:
  10:30:
  10:45:
  11:00: (PRIMITIVE Prendre navette de retour (&34 PATIENT))
  11:15:
  ...
Found schedule in 0:00:00:641 (641 ms)
```

Les primitives originellement engendrées par la décomposition des tâches de plus haut niveau sont remplacées par celles-ci.

Temps d'exécution moyen: 625 ms.

4.2.1.4 Scénario 4 : aucun détail

Nous configurons finalement le planificateur pour générer un horaire omettant les détails nécessaires pour réaliser la tâche globale de se rendre au centre de jour . Nous obtenons le résultat suivant :

```
COLLAPSING (Aller au centre de jour)
WEEK SCHEDULE:

SUNDAY
  ...
  08:45:
  09:00: (PRIMITIVE Aller au centre de jour (&32 PATIENT))
  09:15:
  09:30:
  09:45:
  10:00:
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  ...
Found schedule in 0:00:00:615 (615 ms)
```

Les tâches de haut niveau originellement engendrées par la décomposition de la tâche globale sont remplacées par celle-ci.

Temps d'exécution moyen: 625 ms.

Le planificateur est en mesure de gérer adéquatement la notion de granularité. Remarquons que le temps moyen d'exécution n'est pas affecté par le niveau de granularité. Le planificateur doit en effet considérer la plus haute granularité possible afin d'examiner les primitives d'un tel horaire pour ensuite pouvoir inférer les durées et effets de tâches de haut niveau à substituer. Peu importe la granularité spécifiée, un horaire complet est toujours généré, et il est donc normal que le temps d'exécution ne soit pas affecté par la granularité désirée.

4.2.2 Domaine 2 : journée – médication, prise de repas, autres activités

Ce domaine permet la spécification de la prise de médication et la prise de repas du patient pour une journée particulière (un samedi). Le domaine spécifie une hiérarchie de méthodes pour la prise de médication et des repas. D'autres hiérarchies sont spécifiées pour des activités complémentaires comme des activités de loisirs ou un rendez-vous à la clinique. Des préférences pour chaque type d'activités sont ensuite spécifiées. Le domaine complet tel qu'implémenté est fourni à l'annexe C. Ce type de domaine correspond à la planification

typique de la journée d'un patient par l'intervenant. Les hiérarchies d'activités et les préférences qui y sont associées sont indépendantes les unes des autres en fonction du type d'activités (médication, repas, loisirs, clinique, etc.). On demande au planificateur de construire un horaire qui les accomplira tout en respectant, si possible, les préférences spécifiées. Ce domaine permet de valider la capacité du planificateur à faire une gestion correcte des préférences pour des activités n'ayant à priori aucune hiérarchie commune. Les contraintes temporelles matin et soir sont définies telles que :

- matin : la tâche doit être positionnée dans le deuxième quart de la représentation discrète de la journée, soit entre 6h00 et 11h45 inclusivement

- soir : la tâche doit être positionnée dans le quatrième quart de la représentation discrète de la journée, soit entre 18h00 et 23h45 inclusivement

Ces définitions de matin et soir pourraient être modifiées en changeant la relation `Timezone` implémentée dans le planificateur actuel (voir annexe E). Notons que seule la partie de la grille horaire meublée par le plan est fournie pour des fins de clarté.

4.2.2.1 Scénario 1 : médication, repas, loisirs, sans conflit

Nous spécifions d'abord les activités comme suit :

Médication
Leponex 2 fois par jour
Tardiferon 1 fois par jour
Contrainte temporelle obligatoire sur Leponex: matin et soir
Contrainte temporelle obligatoire sur Tardiferon: matin
Préférence : prise de medication à 8h00 et 19h00
durée de 1 bloc - 15 minutes

Déjeuner
Contrainte temporelle obligatoire: 8h30
durée de 2 blocs - 30 minutes

Diner
Contrainte temporelle obligatoire: 12h15
durée de 3 blocs - 45 minutes

Souper
Contrainte temporelle obligatoire: 18h00
durée de 4 blocs - 1 heure

Jeu d'échecs
Contrainte temporelle obligatoire: matin
durée de 4 blocs - 1 heure

Le planificateur est ensuite lancé. Nous obtenons le résultat typique suivant :

```
WEEK SCHEDULE:

SATURDAY
  ...
  07:45:
  08:00: (PRIMITIVE takeMedication (&27 PATIENT) (&28 Leponex)
         (&29 Tardiferon))
  08:15:
  08:30: (PRIMITIVE breakfast (&2D PATIENT))
  08:45:
  09:00:
  ...
  10:30:
  10:45: (PRIMITIVE chess (&32 PATIENT))
  11:00:
  11:15:
  11:30:
  11:45:
  12:00:
  12:15: (PRIMITIVE lunch (&19 PATIENT))
  12:30:
  12:45:
  13:00:
  ...
  17:45:
  18:00: (PRIMITIVE dinner (&20 PATIENT))
  18:15:
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&8 PATIENT) (&3 Leponex))
  19:15:
```

```
  ...
Found schedule in 0:00:00:188 (188 ms)
```

Pour chaque résultat, la médication et les repas sont toujours correctement positionnés aux mêmes moments, en fonction des contraintes temporelles spécifiées. Le jeu d'échecs est toujours positionné le matin, suivant la définition de la tranche de temps matinale (second quart de la journée). Cette spécification ne génère aucun conflit facilement soluble par le déplacement de l'activité jeu d'échecs. Un horaire valide respectant toutes les préférences spécifiées existe et le planificateur le trouve.

Temps d'exécution moyen: 172 ms.

4.2.2.2 Scénario 2 : médication, repas, avec conflit

Nous spécifions ensuite les activités comme suit :

```
Médication
        Leponex 2 fois par jour
        Tardiferon 1 fois par jour
        Contrainte temporelle obligatoire sur Leponex: matin et soir
        Contrainte temporelle obligatoire sur Tardiferon: matin
        Préférence : prise de medication à 8h00 et 19h00
        durée de 1 bloc - 15 minutes
Déjeuner
        Contrainte temporelle obligatoire: 8h30
        durée de 2 blocs - 30 minutes
Diner
        Contrainte temporelle obligatoire: 12h15
        durée de 3 blocs - 45 minutes
Souper
        Contrainte temporelle obligatoire: 18h15
        durée  de 4 blocs - 1 heure
```

Le planificateur est ensuite lancé. Nous obtenons le résultat typique suivant :

```
WEEK SCHEDULE:

SATURDAY
  ...
  07:45:
  08:00: (PRIMITIVE takeMedication (&1B PATIENT) (&1C Leponex)
         (&1D Tardiferon))
  08:15:
```

```
  08:30: (PRIMITIVE breakfast (&E PATIENT))
  08:45:
  09:00:
  ...
  12:00:
  12:15: (PRIMITIVE lunch (&8 PATIENT))
  12:30:
  12:45:
  13:00:
  ...
  18:00:
  18:15: (PRIMITIVE dinner (&A PATIENT))
  18:30:
  18:45:
  19:00:
  19:15:
  19:30: (PRIMITIVE takeMedication (&1A PATIENT) (&3 Leponex))
  19:45:
  ...
Found schedule in 0:00:00:234 (234 ms)
```

Pour chaque résultat, la médication du matin et les repas sont toujours correctement positionnés aux mêmes moments, en fonction des contraintes temporelles obligatoires spécifiées. Il est cependant impossible de respecter la préférence de prise de médication à 19h00 puisqu'elle entre en conflit avec la contrainte temporelle obligatoire spécifiée pour le souper à 18h15. Le souper a effectivement une durée de 4 blocs horaires et débute à 18h15, ce qui implique que cette primitive couvre les blocs horaires de 18h15, 18h30, 18h45 et 19h00. Le bloc horaire de 19h00 étant donc occupé, il est impossible d'y positionner la primitive `(takeMedication PATIENT Leponex)`, tel que la préférence définie sur la prise de médication l'indique.

Il n'existe pas d'horaire valide respectant toutes les préférences spécifiées. Le planificateur le réalise et relaxe la préférence sur la prise de médication du soir, en la positionnant ailleurs. Remarquons que pour chaque résultat, toutes les contraintes temporelles obligatoires sont toujours respectées (en particulier, la deuxième prise de médication est toujours positionnée le soir, mais jamais à 19h00). Un horaire valide respectant toutes les contraintes temporelles obligatoires (ainsi que certaines préférences) existe et le planificateur le trouve.

Temps d'exécution moyen: 207 ms. Le planificateur réalise relativement rapidement qu'il doit relaxer une préférence par rapport au scénario 1.

4.2.2.3 Scénario 3 : médication, repas, loisirs, clinique, sans conflit

Nous spécifions ensuite les activités comme suit :

```
Médication
        Leponex 1 fois par jour
        Tardiferon 1 fois par jour
        Contrainte temporelle obligatoire sur Leponex: matin
        Contrainte temporelle obligatoire sur Tardiferon: matin
        Préférence : prise de medication à 7h30
        durée de 1 bloc - 15 minutes
Déjeuner
        Contrainte temporelle obligatoire: 8h45
        durée de 2 blocs - 30 minutes
Clinique
        Contrainte temporelle obligatoire: 10h00
        durée de 10 blocs - 2 heures 30 minutes
Jeu d'échecs
        Contrainte temporelle obligatoire: matin
        durée de 4 blocs - 1 heure
```

Le planificateur est ensuite lancé. Nous obtenons le résultat typique suivant :

```
WEEK SCHEDULE:

SATURDAY
  ...
  07:15:
  07:30: (PRIMITIVE takeMedication (&3D PATIENT) (&3F Leponex)
         (&40 Tardiferon))
  07:45: (PRIMITIVE chess (&11 PATIENT))
  08:00:
  08:15:
  08:30:
  08:45: (PRIMITIVE breakfast (&D PATIENT))
  09:00:
  09:15:
  09:30:
  09:45:
  10:00: (PRIMITIVE Clinic appointment with Dr. M (&3B PATIENT))
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  11:30:
  11:45:
```

```
  12:00:
  12:15:
  12:30:
  ...
Found schedule in 0:00:00:125 (125 ms)
```

Pour chaque résultat, la médication du matin, le déjeuner et le rendez-vous à la clinique sont toujours correctement positionnés aux mêmes moments, en fonction des contraintes temporelles spécifiées. Le planificateur positionne le jeu d'échecs d'une manière qui respecte toutes les préférences et contraintes obligatoires, mais ce faisant il est possible qu'il choisisse un emplacement peu pratique (par exemple, 6h00). Ce scénario illustre la nécessité d'une définition précise des préférences pour chaque activité. Nous pourrions par exemple définir une préférence additionnelle indiquant que le jeu d'échecs doive être positionné après la prise de médicament du matin, ce qui garantirait un horaire plus réaliste pour le patient (bien que par rapport au domaine fourni au planificateur, l'horaire généré soit parfaitement valide).

Dans l'état actuel du planificateur, les relations permettant de spécifier diverses contraintes ont été définies d'un point de vue d'informaticien. Avant d'utiliser le planificateur dans un milieu clinique, il sera important de redéfinir les relations d'une manière correspondant aux besoins explicites de l'intervenant. Par exemple, il serait probablement à propos de redéfinir la contrainte temporelle matin comme spécifiant que la tâche doit être positionnée entre **8h00** et 11h45 inclusivement.

Temps d'exécution moyen: 120 ms.

4.2.2.4 Scénario 4 : médication, repas, loisirs, clinique, avec conflit

Nous spécifions finalement les activités comme suit :

Médication
Leponex 1 fois par jour
Tardiferon 1 fois par jour
Contrainte temporelle obligatoire sur Leponex: matin
Contrainte temporelle obligatoire sur Tardiferon: matin
Préférence : prise de medication à 7h30
durée de 1 bloc - 15 minutes

Déjeuner
Contrainte temporelle obligatoire: 8h45
durée de 2 blocs - 30 minutes

Clinique
Contrainte temporelle obligatoire: 10h00
durée de 10 blocs - 2 heures 30 minutes

Jeu d'échecs
Contrainte temporelle obligatoire: matin
durée de 8 blocs - 2 heures
Préférence : les échecs peuvent être omis

Le planificateur est ensuite lancé. Nous obtenons le résultat typique suivant :

```
WEEK SCHEDULE:

SATURDAY
  ...
  07:15:
  07:30: (PRIMITIVE takeMedication (&4995 PATIENT) (&4998 Leponex)
         (&4999 Tardiferon))
  07:45:
  08:00:
  08:15:
  08:30:
  08:45: (PRIMITIVE breakfast (&49A2 PATIENT))
  09:00:
  09:15:
  09:30:
  09:45:
  10:00: (PRIMITIVE Clinic appointment with Dr. M (&499A PATIENT))
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  11:30:
  11:45:
  12:00:
  12:15:
  12:30:
  ...
Found schedule in 0:00:29:204 (29204 ms)
```

Pour chaque résultat, la médication du matin, le déjeuner et le rendez-vous à la clinique sont toujours correctement positionnés aux mêmes moments, en fonction des contraintes temporelles spécifiées. La nouvelle durée du jeu d'échecs (2 heures) le rend cependant impossible à positionner dans la période matinale tel que le spécifie une contrainte temporelle obligatoire. En effet, la présence des primitives `(breakfast PATIENT)` et `(Clinic appointment with Dr. M PATIENT)` (dont le positionnement est dicté par des contraintes temporelles obligatoires) ne laisse aucune séquence suffisamment grande de blocs horaires libres pour positionner la primitive `(chess PATIENT)` le matin (une contrainte temporelle dure indique que le jeu d'échecs doit être obligatoirement positionné le matin). Dans ce domaine-ci, le jeu d'échecs constitue cependant une activité qui peut être omise. Le planificateur le réalise et génère donc un horaire n'incluant pas le jeu d'échecs.

Temps d'exécution moyen: 28050 ms. Le planificateur considère l'entièreté de la grille horaire lors du processus de planification, même si des préférences ou contraintes obligatoires spécifient qu'une primitive doit occuper une position particulière. Le planificateur doit faire l'exploration complète des positions où il est possible de placer le jeu d'échecs avant de réaliser qu'il est impossible de générer un horaire l'incluant, et donc de pouvoir relaxer la préférence spécifiant la présence du jeu d'échecs dans l'horaire. Ce scénario illustre une faiblesse de l'algorithme et ouvre une discussion sur les améliorations possibles à lui apporter, qui font l'objet du Chapitre 5.

4.2.3 Domaine 3 : semaine complète

Ce domaine permet la spécification de diverses activités pour la planification d'une semaine complète. Le domaine spécifie des hiérarchies de méthodes pour des activités de loisirs (dessin et sport), des activités d'entretien (ménage de la résidence), des activités d'hygiène (douches), des activités sociales (rencontres planifiées) ainsi que des activités quotidiennes de routine (médication, recharge d'agenda électronique). Le domaine complet tel qu'implémenté est fourni à l'annexe C. Ce type de domaine correspond à la planification typique d'une semaine d'un patient par l'intervenant. Ce domaine de synthèse permet de valider la capacité

du planificateur à générer un horaire correspondant à des scénarios d'utilisation typique tels que recueillis auprès d'intervenants en milieu clinique.

4.2.3.1 Scénario 1 : semaine complète

Nous spécifions les hiérarchies de méthodes nécessaires à la représentation des activités suivantes :

- prise de médication : Leponex, Tardiferon et cachet pour le ventre quotidiennement
- loisirs : dessin de paysage, dessin de portrait, sport libre, piscine, badminton
- hygiène : deux douches dans la semaine
- entretien : ménage de la résidence et lessive
- social : instance unique de rencontre à la résidence, repas thérapeutique et centre social
- routine : tri du linge sale et recharge d'agenda quotidiennement

Certaines préférences et contraintes obligatoires sont aussi spécifiées. Certaines activités doivent être accomplies à un moment particulier (piscine et badminton par exemple), d'autres doivent présenter un ordonnancement particulier, certaines peuvent être déplacées (les douches par exemples), etc. Le planificateur est ensuite lancé. Nous obtenons le résultat illustré à la Figure 12, présenté sous forme graphique. Une grille horaire typique donnée en sortie par le planificateur est fournie à l'annexe D.

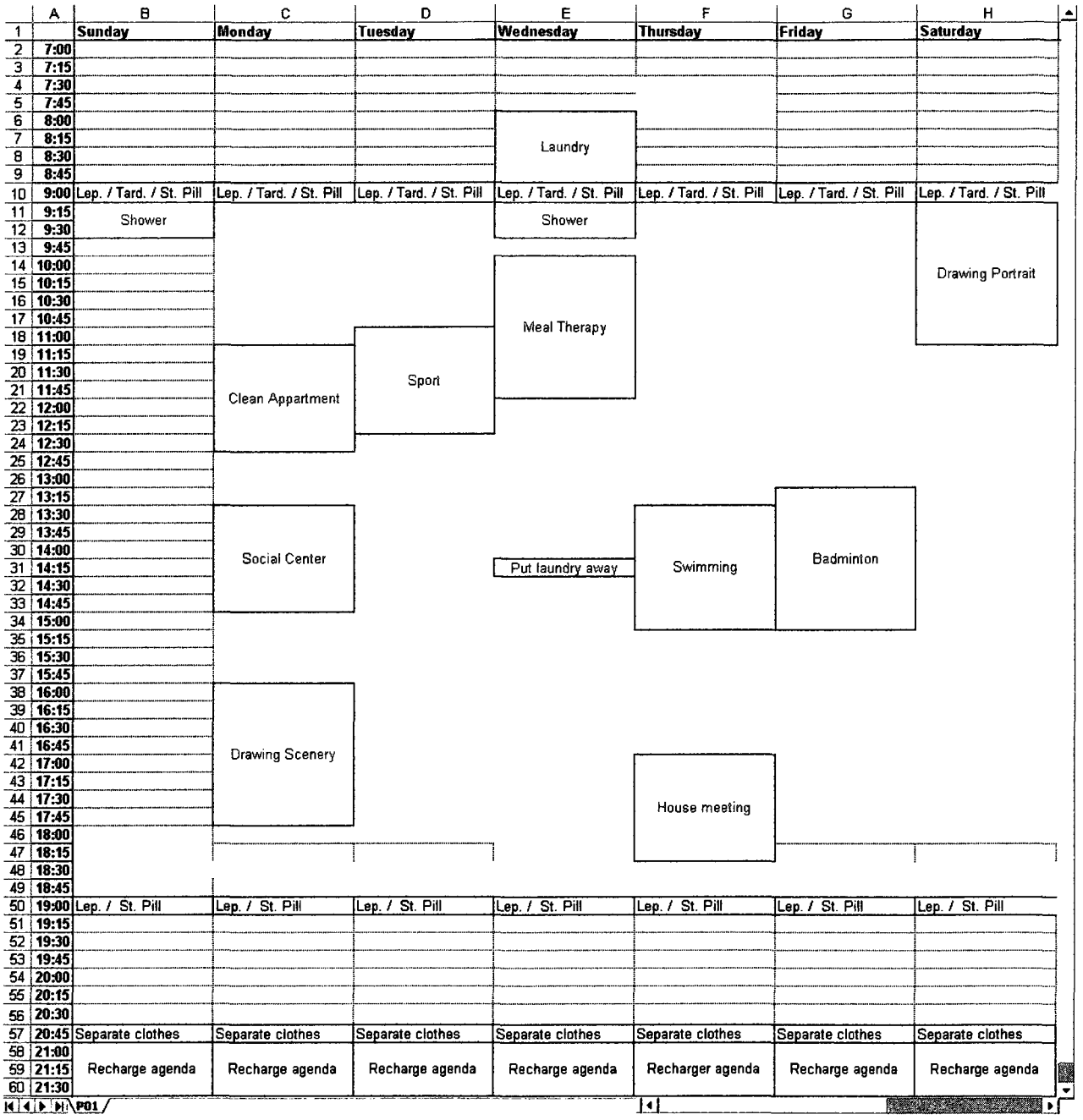


Figure 12 Un résultat typique de la planification d'une semaine complète

Temps d'exécution moyen: 847 ms. Le planificateur dispose de la grille horaire complète pour positionner ses primitives et trouve un horaire valide dans un délai acceptable.

Les horaires générés par le planificateur pour ce domaine présentent de grandes similitudes avec les scénarios d'utilisation typique recueillis. Ils pourraient servir de base à un intervenant, ou même être directement utilisés pour certains patients. Remarquons que ce domaine peut être étendu pour inclure des hiérarchies de méthodes décrivant d'autres activités, ou modifié afin que les hiérarchies incluses soient plus ou moins détaillées. Des notions de granularité pourraient aussi être utilisées.

4.2.4 Résumé des résultats : cas clinique

Les Tableau 2, Tableau 3 et Tableau 4 résument les résultats obtenus pour le cas clinique.

Domaine 1 : déplacement résidence – centre de jour

Scénario	Caractéristique validée	Résultat	Performance moyenne	Notes
1	Horaire accomplissant un déplacement détail complet	Génère un horaire valide, présente le détail complet	625 ms	Gestion de primitives multiples à la même position
2	Omission des détails d'une sous-tâche particulière	Omet les primitives de la sous-tâche, présente les autres détails	625 ms	Inférence de durée et position de la sous-tâche à partir de ses primitives
3	Omission des détails de toutes les sous-tâches	Remplace toutes les primitives par leur tâche de plus haut niveau	625 ms	
4	Aucun détail, activité seulement	Remplace toutes les primitives par l'activité elle-même	625 ms	Inférence de durée et position de l'activité à partir des primitives de ses sous-tâches

Tableau 2 Résumé des résultats - cas clinique, domaine 1

Domaine 2 : journée – médication, prise de repas, autres activités

Scénario	Caractéristique validée	Résultat	Performance moyenne	Notes
1	Horaire d'une journée accomplissant médication, repas, loisir sans conflit	Génère un horaire valide, respecte les contraintes obligatoires et préférences	172 ms	
2	Horaire d'une journée accomplissant médication, repas avec conflit	Génère un horaire valide, respecte les contraintes obligatoires mais relaxe une préférence	207 ms	La préférence relaxée est impossible à respecter
3	Horaire d'avant-midi accomplissant médication, repas, loisirs, clinique sans conflit	Génère un horaire valide, respecte les contraintes obligatoires et préférences	120 ms	La position des loisirs est correcte mais peut s'avérer peu pratique
4	Horaire d'avant-midi accomplissant médication, repas, loisirs (optionnel), clinique avec conflit	Génère un horaire valide, respecte les contraintes obligatoires mais élimine les loisirs	28050 ms	Les loisirs sont impossibles à placer, requiert l'exploration de l'espace horaire complet

Tableau 3 Résumé des résultats - cas clinique, domaine 2

Domaine 3 : Semaine complète

Scénario	Caractéristique validée	Résultat	Performance moyenne	Notes
1	Horaire d'une semaine typique, préférences et contraintes diverses	Génère un horaire valide, respecte les contraintes obligatoires et préférences si possible	847 ms	Cas d'utilisation typique, peut être augmenté

Tableau 4 Résumé des résultats - cas clinique, domaine 3

4.3 Exportation vers le système mobile MOBUS

Nous avons également valider l'implémentation de l'interface entre le planificateur et le système mobile MOBUS, afin de vérifier si les horaires générés par le planificateur peuvent y être correctement exportés de manière automatique.

Le planificateur met en place les services nécessaires pour exporter les horaires générés pour les scénarios précédents vers le système mobile MOBUS. Lorsqu'un horaire hebdomadaire est généré, il peut ensuite être exporté vers MOBUS pour un patient et une semaine en particulier. La conversion de la sortie du planificateur en un format reconnu par MOBUS ainsi que l'interface avec le serveur MOBUS sont automatiquement prises en charge par le planificateur. Une date représentant le début de la semaine désirée ainsi qu'un identifiant de patient doivent être fournis au planificateur, qui se charge ensuite de créer les activités MOBUS à partir de l'horaire généré. L'horaire généré est considéré comme représentant une semaine complète (peu importe la taille de la grille horaire lui étant associée); les dates de début et les durées des activités MOBUS sont inférées à partir de la grille horaire, de la date de début de semaine et de l'identifiant de patient fournis. Lors de la définition de domaine, il est possible d'attacher des informations supplémentaires aux primitives et aux tâches qui permettent de définir le titre de l'activité MOBUS qui sera finalement présenté au patient via le client MOBUS. Par défaut, le nom de la primitive ou de la tâche est utilisé, mais on peut spécifier tout autre titre voulu, ou construire des titres à partir des variables d'une primitive.

On pourrait par exemple configurer une primitive `(PRIMITIVE takeMedication (&23D PATIENT) (&253 Leponex) (&254 Tardiferon))` pour afficher le titre d'activité MOBUS *Prendre Leponex et Tardiferon* plutôt que d'afficher *takeMedication.*

A titre d'exemple, nous présentons les informations fournies au patient par le client MOBUS suite à l'exportation du domaine 1 – scénario 3 , soit l'horaire suivant :

```
SUNDAY
  ...
  08:45:
  09:00: (PRIMITIVE Prendre tram jusqu'au centre (&32 PATIENT))
  09:15:
  09:30:
  09:45:
  10:00: (PRIMITIVE Marcher jusqu'au centre (&33 PATIENT))
  10:15:
  10:30:
  10:45:
  11:00: (PRIMITIVE Prendre navette de retour (&34 PATIENT))
  11:15:
  ...
```

Le patient accède à son horaire exactement comme si chaque activité avait été entrée manuellement par un intervenant. Il peut les visualiser, confirmer, etc, au même titre que toute autre activité MOBUS. La Figure 13 illustre le cycle de vie typique d'un horaire pour le patient y accédant via son client MOBUS.

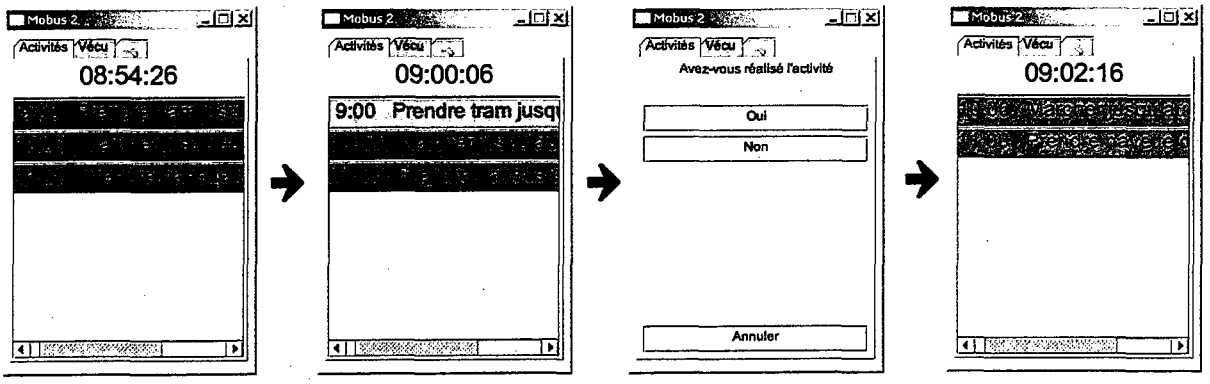


Figure 13 Cycle de vie d'un horaire pour le patient

Notons que tout horaire généré par le planificateur peut être exporté vers MOBUS de la même manière, incluant les plans du cas théorique, bien que cela n'ait aucun sens.

Chapitre 5

Discussion et travaux futurs

Pour résoudre la problématique énoncée, nous avons élaboré un planificateur basé sur les algorithmes de décomposition de hiérarchies de tâches. À la suite des résultats obtenus, nous discutons maintenant de la pertinence de la recherche effectuée et en faisons une critique. Nous présentons également les travaux futurs qui peuvent être envisagés.

5.1 Contributions

Au cours de cette recherche, nous avons étendu un algorithme de planification établi avec des notions de planification dans le temps, gestion de préférences et gestion de granularité au niveau des tâches, afin qu'il puisse être adapté au problème particulier de la planification d'horaire pour personnes atteintes de déficit cognitif. Le planificateur construit à partir de cet algorithme étendu a cependant été implémenté de manière à être découplé des domaines de planification qu'il considère. Il pourrait aisément être utilisé pour résoudre d'autres types de problèmes. Le cas théorique présenté au chapitre 4 l'illustre bien. Il y a en effet peu à faire pour passer de ce domaine à un domaine similaire aux problèmes de *Dock-Worker Robots* [7].

La nature OO de l'environnement de développement du planificateur le rend également facile à modifier afin d'exprimer de nouveaux types de relations entre les tâches et primitives d'un domaine de planification. La nature et la quantité des relations que le planificateur peut considérer ne sont limitées que par les ressources matérielles disponibles et les besoins du programmeur les implémentant. Une relation définie et utilisée lors de la résolution d'un domaine devient disponible pour les futurs problèmes de planification.

Par conséquent, le planificateur implémenté fait preuve d'une grande flexibilité par rapport aux types de problèmes qu'il peut exprimer. La notion de granularité est cependant plus attachée à la problématique d'une population atteinte de déficits cognitifs, mais elle peut être complètement ignorée si elle n'est pas requise.

Dans le présent contexte d'application, le planificateur construit a été intégré à l'orthèse cognitive MOBUS dont l'utilité a été démontrée, et qui est présentement utilisée en milieu clinique. MOBUS a été construit sur le principe qu'un intervenant fournit de l'assistance à un grand nombre de patients et les infrastructures nécessaires pour en faire la gestion sont déjà en place. Les résultats initiaux indiquent que l'ajout de ce nouveau service d'aide à la planification allégera effectivement la tâche de l'intervenant, lui permettant de consacrer plus de temps à fournir une meilleure assistance à ses patients.

5.2 Critique

Le choix d'un algorithme de type HTN comme prémisse de recherche a permis la construction d'un planificateur très flexible, tel qu'illustré par les domaines très variés qu'il peut prendre en considération, mais cette flexibilité a un prix.

D'abord, la spécification de domaine de planification, bien que permettant d'exprimer des problèmes complexes, peut s'avérer très lourde. Dans l'état actuel de la recherche, les hiérarchies de tâches pour chaque domaine doivent être spécifiées en code JAVA respectant les standards et définitions d'objets (tâches, primitives, méthodes, relations, préférences, etc.) mis en place par le planificateur. Comme le reste du projet, le code requis pour définir les hiérarchies de tâches respecte les principes OO et devrait pouvoir être manipulé relativement aisément par un individu ayant des connaissances de programmation OO. Il est cependant irréaliste de supposer qu'un intervenant sera en mesure de le faire, ce qui impose la présence d'un informaticien à ses côtés lors des expérimentations futures. Ceci pose un problème, car dans le présent contexte d'utilisation, l'intervenant devrait pouvoir interagir avec l'orthèse cognitive MOBUS lui-même, sans aucune assistance externe.

Ensuite, les performances en temps d'exécution du planificateur pourront éventuellement s'avérer problématiques. Les performances observées lors des tests sur les scénarios typiques restent acceptables, mais de par la nature d'un algorithme de type HTN, à mesure que les domaines deviendront complexes et/ou contiendront de plus en plus de préférences générant des conflits, le temps requis par processus de planification convergera vers une valeur inacceptable. À titre d'exemple, lors du développement, des scénarios de tests au pire cas ont été construits afin d'assurer la validité du planificateur.

Certaines définitions modestes de domaines incluant une quarantaine de tâches à planifier pouvaient générer des temps d'exécution atteignant plusieurs heures, sur les mêmes environnements de test que ceux présentés au chapitre 4, ce qui constitue un délai inacceptable. Il est cependant important de remarquer que ces domaines ont été construits de manière à générer des conflits au niveau des préférences de manières à ce que le planificateur ne puisse trouver une solution qu'après avoir effectué l'exploration complète de l'espace de décompositions et relaxé toutes les préférences possibles. En d'autres termes, la toute dernière décomposition explorée était celle qui menait à un plan. Il est possible que le planificateur trouve un plan valide pour un domaine incluant beaucoup plus de tâches à décomposer en un temps en dessous de quelques secondes si aucun conflit ne survient ou si les conflits potentiels sont facilement solubles (par exemple, s'il suffit de repositionner une primitive à un autre bloc horaire). Comme pour toute planification HTN, une bonne définition de domaine est requise pour assurer un temps de planification minimal [21]. Cependant, dans le présent contexte, l'intervenant ne devrait pas avoir à se soucier de contraintes d'optimisation du domaine.

Des solutions potentielles à ces problèmes sont présentées dans la prochaine section.

Nous remarquons finalement que la plupart des scénarios typiques recueillis auprès des intervenants constituent des problèmes qui peuvent généralement être exprimés comme une grille horaire discrète que l'on tente de remplir d'activités auxquelles sont associées certaines contraintes. Certains algorithmes de type CSP peuvent résoudre ces problèmes d'une manière extrêmement performante, et ont l'avantage de présenter une spécification de problème

généralement plus simple que les algorithmes de planification. Il aurait pu être intéressant de considérer une solution de type CSP, au prix de la perte de l'expressivité rendue possible par le planificateur construit.

5.3 Travaux futurs

Plusieurs projets peuvent être envisagés à la suite de la présente recherche.

5.3.1 Expérimentation en milieu clinique

La capacité du planificateur à générer des horaires correspondant aux besoins de l'intervenant a été validée par des tests effectués sur des scénarios d'utilisation typiques recueillis à partir d'expérimentations passées en milieu clinique [23]. La prochaine étape rationnelle consiste à mettre en place une série d'expérimentations dans ce même milieu, avec l'orthèse cognitive MOBUS étendue des services de planification. De telles expérimentations permettront de valider le comportement et l'utilité du planificateur dans un véritable contexte d'utilisation ainsi que de cibler de nouveaux besoins de l'intervenant et d'adapter le planificateur en conséquence.

Avant de lancer un tel projet d'expérimentation, nous devons cependant considérer la complexité de la spécification de domaines de planification évoquée à la section précédente. Au cours d'une expérimentation initiale, il serait acceptable d'assigner un informaticien à l'intervenant pour l'aider à définir les domaines de planification de ses patients, mais il serait plus intéressant de profiter de cette occasion pour développer des outils permettant une spécification plus facile.

Une hiérarchie de tâches déjà définie peut être réutilisée dans n'importe quel domaine. Nous pourrions d'abord envisager un outil permettant la gestion de « librairies » de hiérarchies de tâches prédéfinies (hiérarchie pour la médication, hiérarchie pour l'hygiène, hiérarchie pour l'entretien, etc.). L'intervenant utiliserait cet outil pour spécifier quelle(s) hiérarchie(s) de tâches doivent être planifiées pour son patient. L'outil générerait ensuite le code nécessaire pour la spécification de domaine à partir des hiérarchies prédéfinies.

Cette solution serait acceptable pour des expérimentations initiales. Elle est cependant limitée par la quantité de hiérarchies prédéfinies. Si l'intervenant désire spécifier de nouvelles hiérarchies de tâches, le problème de complexité apparaît à nouveau.

Idéalement, le développement d'un outil capable de prendre une spécification informelle et/ou en langage naturel de la part de l'intervenant et de la transformer en une spécification formelle (telle qu'utilisée par le planificateur) éliminerait complètement le problème de complexité. Le passage du langage naturel à une spécification formelle est cependant un problème extrêmement complexe en lui-même [15].

5.3.2 Amélioration des performances

Bien que les performances en temps d'exécution du planificateur paraissent acceptables pour la plupart des cas d'utilisation typiques, il serait intéressant d'effectuer un travail d'optimisation, si ce n'est que pour améliorer le temps de réponse afin de faciliter la tâche l'intervenant. Nous pouvons remarquer deux points en particulier qui pourraient mener à des améliorations de performance considérables

5.3.2.1 Exploration parallèle

L'algorithme présenté au chapitre 3 constitue une exploration de l'espace de décomposition des hiérarchies de tâches. L'exploration d'espace se prête bien au parallélisme.

Lors de la sélection d'un élément (tâche, primitive, position horaire, valeur de variable, méthode ou préférence à relaxer), il serait possible de modifier l'algorithme afin que plusieurs éléments soient sélectionnés (2, 4, 8 etc.). Un même nombre de *mondeSuivant* pourrait ensuite être générés, et les appels récursifs à **PLANIFICATEUR** en découlant pourraient être lancés dans des processus concurrents. Sur une plate-forme supportant des processus concurrents, l'exploration complète sera accélérée. On pourrait limiter les explorations concurrentielles à un certain type d'élément (les tâches et primitives par exemple), afin de garder le nombre de processus concurrents plus bas. À titre d'exemple, la Figure 14 illustre la décomposition concurrente de deux tâches.

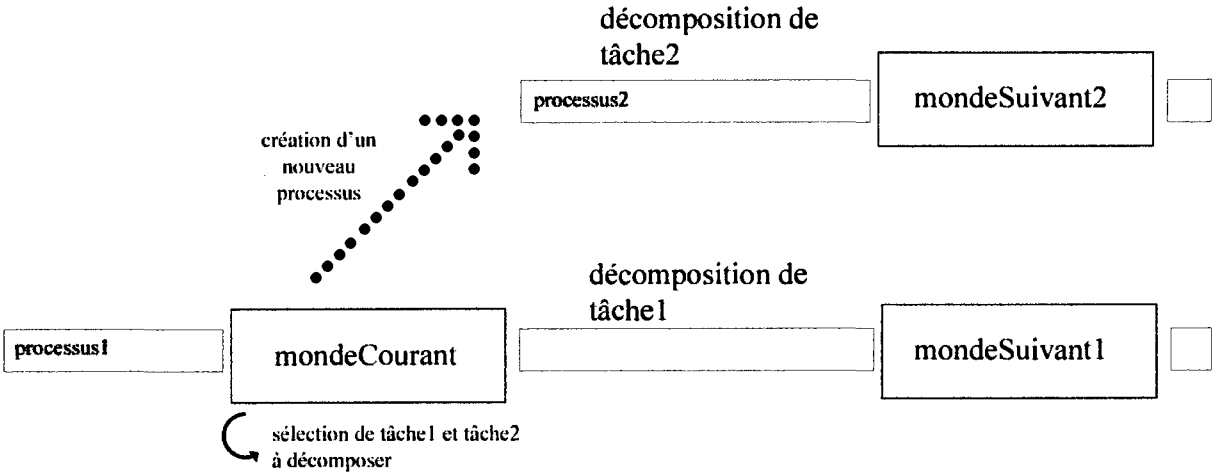


Figure 14 Exploration concurrente de deux décompositions de tâche

Le planificateur a été implémenté en JAVA 5.0. L'environnement JAVA 6.0 met en place un un puissant *framework* implémentant tous les services requis au développement d'applications désirant utiliser des processus d'exécution concurrents (création/destruction de processus, communication inter-processus, découplage de la plate-forme matérielle, etc.).Ce *framework* a également été intégré à l'environnement JAVA 5.0 [10]. Il serait intéressant de l'utiliser pour paralléliser le planificateur.

5.3.2.2 Exploration intelligente

Dans l'implémentation actuelle du planificateur, l'exploration de l'espace de décomposition est effectuée de manière non-déterministe. Les sélections d'éléments à considérer sont basés sur un ordre pseudo-aléatoire. Bien que fonctionnelle, cette exploration aveugle peut causer des problèmes de performance, en particulier lorsque des préférences et/ou des contraintes obligatoires génèrent des conflits, comme l'illustre le temps moyen d'exécution du scénario présenté en 4.2.2.4. L'algorithme doit effectuer l'exploration complète du sous-espace courant avant de pouvoir déterminer qu'il doive lever une préférence, ou réaliser que ce sous-espace ne mène à aucun plan valide. Il n'y a aucun mécanisme orientant l'exploration.

Il serait possible d'étendre l'algorithme de planification pour orienter l'exploration en fonction des relations spécifiées pour les tâches et primitives. Nous pouvons envisager deux principaux mécanismes d'orientation.

Il serait d'abord intéressant d'implémenter un mécanisme d'ordonnancement au niveau de la sélection de blocs horaire. Lorsque le planificateur considère une primitive à positionner dans la grille horaire, tous les blocs horaires libres accommodant la durée de la primitive sont à priori considérés comme un emplacement potentiel. La primitive est ensuite positionnée à un tel emplacement, puis une vérification est faite afin de déterminer si les contraintes spécifiées pour cette primitive sont respectées. Comme pour toutes les autres sélections, les blocs horaires sont considérés dans un ordre pseudo-aléatoire. Cependant, nous avons constaté que des contraintes et/ou préférences temporelles sont habituellement associées aux primitives dans la plupart des domaines d'utilisation typiques. Un mécanisme qui forcerait le planificateur à considérer d'abord les blocs horaires respectant les contraintes et/ou préférences temporelles associées à une primitive aurait pour effet de forcer le planificateur à considérer en premier lieu les sous-espaces d'exploration ayant des chances de mener à un plan respectant ces contraintes et/ou préférence. La découverte d'un plan valide (s'il existe) serait donc accélérée.

Il serait également intéressant d'implémenter un mécanisme analysant les causes ayant mené à l'échec d'une exploration d'un sous-espace, afin d'orienter l'exploration à la suite de cet échec, ou même l'arrêter. Reprenons le scénario présenté en 4.2.2.4. Tel que présenté, la lenteur du temps d'exécution provient du fait que le planificateur considère toutes les décompositions de tâches possibles incluant la primitive du jeu d'échecs, et pour chaque décomposition, considère toutes les positions possibles de chaque primitive avant de pouvoir réaliser qu'il est impossible de respecter la préférence spécifiant l'inclusion du jeu d'échecs. La décision de lever la préférence est prise seulement après avoir exploré toutes les décompositions menant à un échec. S'il était possible de déterminer la cause d'un échec dès que celui-ci survient, il serait possible d'éliminer les explorations qui provoqueront un échec pour la même cause, réduisant ainsi la taille de l'espace à explorer.

Cependant, la nature complètement libre de l'implémentation des relations mise en place par le planificateur rend ce type d'analyse difficile. Une relation peut prendre toute forme imaginée par le programmeur l'ayant implémentée et l'échec d'une exploration peut être engendré par une multitude de combinaisons de facteurs (qui peuvent évoluer dans la durée de vie du planificateur). Il est donc laborieux d'élaborer une mécanique formelle permettant l'analyse d'échec.

5.3.3 Re-planification automatique

L'intégration d'un service de planification automatique au système mobile MOBUS mène naturellement à la considération d'un processus de re-planification automatique. Comme le système mobile MOBUS conserve des informations sur les activités ayant été complétées (ou négligées) par le patient, il serait possible de comparer ces informations à l'horaire original du patient. La re-planification automatique d'un nouvel horaire accomplissant les activités négligées pourrait alors être effectuée, possiblement sans l'intervention de l'intervenant. Cependant, certains points doivent être considérés dans le présent contexte.

Premièrement, une certaine prudence doit être exercée si l'on désire fournir un horaire au patient sans l'intervention directe de l'intervenant. Comme nous l'avons vu, le déficit cognitif du patient peut engendrer des contraintes particulières lors de la construction d'un horaire; l'horaire re-planifié pourrait ne plus respecter ces contraintes.

Deuxièmement, une quelconque forme de détection de faux négatifs doit être implémentée au niveau de la complétion des activités. Un patient utilisant MOBUS est atteint d'un déficit cognitif, et, comme l'information sur les activités qu'il a complétées est essentiellement obtenue à partir d'une confirmation du patient, il est raisonnable de supposer que certaines activités seront incorrectement marquées comme complétées et vice versa. Un processus de re-planification qui ne détecterait pas au moins un minimum de faux négatifs fournirait au patient des horaires trompeurs ne correspondant pas à la situation réelle du patient, ce qui ne ferait qu'accentuer ses troubles cognitifs plutôt que de les soulager. Le planificateur et le

système MOBUS l'utilisant doivent être intégrés dans un environnement capable d'assurer la supervision du patient nécessaire à cette détection de faux négatifs.

Finalement, nous devons considérer la plate-forme sur laquelle le processus de re-planification prendra place. Il serait intéressant de re-planifier à même l'appareil portable assigné au patient. Ceci pourrait lui assurer une certaine autonomie s'il advenait que le lien au serveur MOBUS ne soit plus disponible pour une raison quelconque. Malheureusement, dans l'état actuel de l'implémentation du planificateur, il n'est pas envisageable de le déployer sur une plate-forme portable en vertu de ses exigences en mémoire et en puissance de calcul. Un travail d'optimisation du planificateur par rapport à la plate-forme portable envisagée devrait être effectué.

Conclusion

Dans ce mémoire, nous avons présenté la relation entre un patient et son intervenant, leur utilisation du système mobile MOBUS et comment l'introduction d'un planificateur automatique dans ce contexte peut contribuer à réduire la charge de travail de l'intervenant et par conséquent améliorer la qualité de l'assistance que le patient reçoit.

Les résultats initiaux obtenus à partir de scénarios typiques provenant d'expérimentations antérieures indiquent que le planificateur implémenté est en mesure de répondre aux besoins de l'intervenant par rapport à la gestion de la granularité, des préférences associées à un patient et de la planification dans le temps. Les scénarios recueillis proviennent principalement de populations schizophrènes, mais nous estimons que les techniques de planification utilisées peuvent aussi s'avérer utiles pour d'autres populations.

Il faut cependant remarquer que dans son état actuel, la spécification d'un problème de planification pour un patient s'avère difficile pour un usager n'ayant pas de connaissances informatiques, ce qui est le cas de la plupart des intervenants. Il sera nécessaire d'élaborer des outils permettant une interface facile entre l'intervenant et le planificateur.

L'intégration d'un planificateur automatique au système mobile MOBUS est néanmoins un pas dans la bonne direction, et une série d'expérimentations en milieu clinique permettra de raffiner ses caractéristiques et de s'assurer qu'il soit bien adapté aux besoins présents et futurs de l'intervenant, lui permettant ainsi de fournir une meilleure assistance à ses patients.

Annexe A

Implémentation du planificateur

A.1 Environnement de développement

- PC sous Windows XP SP 3
- Projet OO
- JAVA 5.0 (jre 1.5.0_15)
- SDK Eclipse 3.2.2
- JDBC 3.0 (interface à MOBUS)
- Intégré aux infrastructures de développement du laboratoire DOMUS : projet `mobus_planner`

A.2 Implémentation de l'algorithme de planification

Ce code correspond à l'implémentation de l'algorithme présenté en 3.2.2. Le lecteur est invité à consulter le projet complet disponible par les infrastructures du laboratoire DOMUS.

```
public Plan recursivePlan(World currentWorld, boolean determinist) {

    Plan partialPlan = new Plan(false, currentWorld.getScheduleSize());

    if (currentWorld.getActionPool().getAllAction().isEmpty()) { // nothing to plan, return
successful but empty plan
        return partialPlan;
    }

    ActiveSet<Action> eligibleAction = currentWorld.getActionWithoutPrecedence();
    Iterator<Action> eligibleActionIterator;
    if (determinist) {
        eligibleActionIterator = eligibleAction.iterator();
    } else {
        eligibleActionIterator = eligibleAction.nditerator();
    }

    while (eligibleActionIterator.hasNext()) {
        Action thisAction = eligibleActionIterator.next();

        if (thisAction.isPrimitive()) {
            if (((Primitive) thisAction).nbFreeVariable().compareTo(0) == 0) {
               // this primitive is fully instanciated
                ActiveSet<Integer>freeScheduleSpot=
                currentWorld.getFreeSpotForPrimitive((Primitive) thisAction);
                Iterator<Integer> freeScheduleSpotIterator;
                if (determinist) {
                    freeScheduleSpotIterator = freeScheduleSpot.iterator();
                } else {
                    freeScheduleSpotIterator = freeScheduleSpot.nditerator();
                }

                boolean precondBroken = false;
                while (freeScheduleSpotIterator.hasNext()) {
                    currentWorld.removePrimitiveFromSchedule((Primitive) thisAction);
                    if ( precondBroken ) { // this primitive breaks preconds, it's useless to
                                           try it in other spots
                        break;
                    }

                    Integer thisScheduleSpot = freeScheduleSpotIterator.next();
                    currentWorld.setPrimitiveInSchedule((Primitive)thisAction,
                                                        thisScheduleSpot);

                    World nextWorld = currentWorld.dumbClone(thisAction);
                    try {
                        nextWorld.ApplyPrimitive(
                                    (Primitive)currentWorld.getNextWorldAction());
                    } catch (PrimitiveNotApplicableToWorld e) { // preconds not respected
                        precondBroken = true;
                        continue;
                    } catch (PrimitiveNotInActionPool e) {
                        e.printStackTrace();
                        System.exit(0);
                    }
```

```
            partialPlan = recursivePlan(nextWorld, determinist);

            if (! partialPlan.isFailed()) {
                partialPlan.addActionAt((Primitive) thisAction, 0, thisScheduleSpot);
                return partialPlan;
            }
        }

    } else { // this primitive has free variables
        Permutator permutator = new Permutator();
        try {
            permutator.buildPermutationSet(((Primitive) thisAction).nbFreeVariable(),
             currentWorld.getWorldState().getAllPossibleVariable());
        } catch (NotEnoughElements e) { // can't fill the free variables, the plan
                                        fails
            partialPlan.setFailed(true);
            return partialPlan;
        }

        Iterator<Permutation> permutationIterator;
        if (determinist) {
            permutationIterator = permutator.iterator();
        } else {
            permutationIterator = permutator.nditerator();
        }

        while(permutationIterator.hasNext()) {
            World nextWorld = null;

            try {
                ((Primitive) thisAction).clearFreeVariable();
                ((Primitive) thisAction).applyPermutation(permutationIterator.next());
                currentWorld.performActionIntegrityCheck(false);
            } catch (PermutationNotApplicable e) {
                e.printStackTrace();
                System.exit(0);
            } catch (FailedDuplicateCheck e) { // permutation induced an invalid
                                               world, move on
                ((Primitive) thisAction).clearFreeVariable();
                continue;
            }

            ActiveSet<Integer>freeScheduleSpot=
                       currentWorld.getFreeSpotForPrimitive((Primitive) thisAction);
            Iterator<Integer> freeScheduleSpotIterator;
            if (determinist) {
                freeScheduleSpotIterator = freeScheduleSpot.iterator();
            } else {
                freeScheduleSpotIterator = freeScheduleSpot.nditerator();
            }

            boolean precondBroken = false;
            while (freeScheduleSpotIterator.hasNext()) {
                currentWorld.removePrimitiveFromSchedule((Primitive) thisAction);
                if ( precondBroken ) { // this primitive breaks preconds, it's useless
                                       to try it in other spots
                    break;
                }

                Integer thisScheduleSpot = freeScheduleSpotIterator.next();
                currentWorld.setPrimitiveInSchedule((Primitive) thisAction,
                                                    thisScheduleSpot);
                try {
                    nextWorld = currentWorld.dumbClone(thisAction);
                    nextWorld.ApplyPrimitive(
                                    (Primitive) currentWorld.getNextWorldAction());
                } catch (PrimitiveNotApplicableToWorld e) { // preconds not respected
```

```
                        ((Primitive) thisAction).clearFreeVariable();
                        precondBroken = true;
                        continue;
                    } catch (PrimitiveNotInActionPool e) {
                        e.printStackTrace();
                        System.exit(0);
                    }

                    partialPlan = recursivePlan(nextWorld, determinist);

                    if (! partialPlan.isFailed()) {
                        partialPlan.addActionAt((Primitive) thisAction, 0,
                                                thisScheduleSpot);
                        return partialPlan;
                    }
                }

                ((Primitive) thisAction).clearFreeVariable();
            }
        }

    } else {
        ActiveSet<Method> applicableMethod =
                        methodPool.getAllMethodsDecomposingTask((Task) thisAction);
        MethodPreferenceLifterIterator applicableMethodIterator =
        new MethodPreferenceLifterIterator(applicableMethod.getActiveSet(),determinist);

        while (applicableMethodIterator.hasNext()) {
            Method thisApplicableMethod = applicableMethodIterator.next();
            World nextWorld = currentWorld.dumbClone(thisAction);
            try {
                // apply nextWorld task to method which will decompose it
                thisApplicableMethod.applyTaskVar(
                            (Task) currentWorld.getNextWorldAction(), true);

                // decompose task, this will check if preconds are respected
                nextWorld.ApplyMethodToTask(thisApplicableMethod,(
                                        Task) currentWorld.getNextWorldAction(),
                                        englobingGranularity, desiredGranularity);
            } catch (MethodNotApplicableToWorld e) { // method can't be applied to
                                         nextWorld due to preconds, dead end, move on
                continue;
            } catch (TaskNotInActionPool e) { // can't remove task we decomposed,
                                                fatal error
                e.printStackTrace();
                System.exit(0);
            } catch (TaskNotApplicableToMethod e) { // caught a bad task, move on
                continue;
            }
            partialPlan = recursivePlan(nextWorld, determinist);

            if (! partialPlan.isFailed()) {
                return partialPlan;
            }
        }
    }
}

// all actions/methods/schedule spots tried at this point,
   and all preferences have been lifted,
// no plan found, it's a dead end
partialPlan.setFailed(true);
return partialPlan;
}
```

Annexe B

Environnement de test

- Pentium 4 single core CPU 3.0 Ghz (hyper-threading activé)
- 2 GB RAM
- machine virtuelle JAVA 1.5.0_15
- tests exécutés via l'environnement Eclipse 3.2.2

Annexe C

Domaines

Suivent les domaines de planification des cas théoriques et cliniques tels qu'implémenté lors des tests. Le lecteur est invité à consulter la définition complète des tests, disponible par les infrastructures du laboratoire DOMUS.

C.1 Cas théorique

```
//      methods to stack blocks
//      one for stacking from the table, one from stacking from another block
 Task t_stack2Blocks = new Task("stack2Blocks");
 Variable v_blockA = new Variable();
 Variable v_blockB = new Variable();
 t_stack2Blocks.setParam(v_blockA, v_blockB);

 Method m_Stack2Blocks = new Method("Stack2Blocks", t_stack2Blocks);
 m_Stack2Blocks.setParam(v_blockA, v_blockB);
 m_Stack2Blocks.setPrecond(new State("clear", v_blockA),
                           new State("clear", v_blockB),
                           new State("onTable", v_blockA));
     Primitive p_stackFromTable = new Primitive("stackFromTable");
     p_stackFromTable.setParam(v_blockA, v_blockB);
     p_stackFromTable.setPrecond(new State("clear", v_blockA),
                                 new State("clear", v_blockB),
                                 new State("onTable", v_blockA));
     p_stackFromTable.setPositiveEffect(new State("on", v_blockA, v_blockB));
     p_stackFromTable.setNegativeEffect(new State("onTable", v_blockA),
                                        new State("clear", v_blockB));
     p_stackFromTable.setTimeCellLength(1);
     if (testCase == 3 || testCase == 4) {
         p_stackFromTable.setTimeCellLength(3);
     }
 m_Stack2Blocks.setSubtask(p_stackFromTable);
 m_Stack2Blocks.setSubtaskRelation();

//      methods to clear a block
//      one for a block that's already clear, one for a block under another block
Task t_clearBlock = new Task("clearBlock");
Variable v_blockToClear = new Variable();
t_clearBlock.setParam(v_blockToClear);

Method m_clearBlockA = new Method("ClearBlockA", t_clearBlock);
m_clearBlockA.setParam(v_blockToClear);
m_clearBlockA.setPrecond(new State("clear", v_blockToClear));
m_clearBlockA.setSubtask(); // already clear, do nothing
m_clearBlockA.setSubtaskRelation();

Method m_clearBlockB = new Method("ClearBlockB", t_clearBlock);
```

```
m_clearBlockB.setParam(v_blockToClear);
    Variable v_blockBlocking = new Variable();
m_clearBlockB.setPrecond(new State("on", v_blockBlocking, v_blockToClear),
                         new State("clear", v_blockBlocking));
    Primitive p_unstackFromBlock = new Primitive("unstackFromBlock");
    p_unstackFromBlock.setParam(v_blockBlocking, v_blockToClear);
    p_unstackFromBlock.setPrecond(new State("on", v_blockBlocking, v_blockToClear),
                                  new State("clear", v_blockBlocking));
    p_unstackFromBlock.setPositiveEffect(new State("onTable", v_blockBlocking),
                                         new State("clear", v_blockToClear));
    p_unstackFromBlock.setNegativeEffect(new State("on", v_blockBlocking, v_blockToClear));
    p_unstackFromBlock.setTimeCellLength(1);
    if (testCase == 3 || testCase == 4) {
        p_unstackFromBlock.setTimeCellLength(3);
    }
m_clearBlockB.setSubtask(p_unstackFromBlock);
m_clearBlockB.setSubtaskRelation();

//      methods to make a 3 pile
Task t_make3Pile = new Task("make3Pile");
Variable v_blockOne = new Variable();
Variable v_blockTwo = new Variable();
Variable v_blockThree = new Variable();
t_make3Pile.setParam(v_blockOne, v_blockTwo, v_blockThree);

Method m_make3Pile = new Method("Make3Pile", t_make3Pile);
m_make3Pile.setParam(v_blockOne, v_blockTwo, v_blockThree);
    // to make a 3 pile we need to
    Task t_clearBlockOne = new Task("clearBlock");  // clear the first block
    t_clearBlockOne.setParam(v_blockOne);

    Task t_clearBlockTwo = new Task("clearBlock"); ;  // clear the second block
    t_clearBlockTwo.setParam(v_blockTwo);

    Task t_clearBlockThree = new Task("clearBlock"); // clear the third block
    t_clearBlockThree.setParam(v_blockThree);

    Task t_stackTwoOnThree = new Task("stack2Blocks"); // stack block one on block two
    t_stackTwoOnThree.setParam(v_blockTwo, v_blockThree);

    Task t_stackOneOnTwo = new Task("stack2Blocks"); // stack block three on block two
    t_stackOneOnTwo.setParam(v_blockOne,v_blockTwo);

m_make3Pile.setSubtask(t_clearBlockOne,t_clearBlockTwo,  t_clearBlockThree,  t_stackTwoOnThree,
t_stackOneOnTwo);
m_make3Pile.setSubtaskRelation(new Precedence(t_stackTwoOnThree, t_stackOneOnTwo),
                               new Precedence(t_clearBlockOne, t_stackTwoOnThree),
                               new Precedence(t_clearBlockTwo, t_stackTwoOnThree),
                               new Precedence(t_clearBlockThree, t_stackTwoOnThree));
if ( testCase == 5 ) {
    m_make3Pile.setSubtaskRelation(new Precedence(t_stackTwoOnThree, t_stackOneOnTwo),
            new Precedence(t_clearBlockOne, t_stackTwoOnThree),
            new Precedence(t_clearBlockTwo, t_stackTwoOnThree),
            new Precedence(t_clearBlockThree, t_stackTwoOnThree),
            new BeginAt(t_stackOneOnTwo,2,10)
    );
}

if ( testCase == 6 ) {
    m_make3Pile.setSubtaskRelation(new Precedence(t_stackTwoOnThree, t_stackOneOnTwo),
            new Precedence(t_clearBlockOne, t_stackTwoOnThree),
            new Precedence(t_clearBlockTwo, t_stackTwoOnThree),
            new Precedence(t_clearBlockThree, t_stackTwoOnThree),
            new BeginAt(t_stackOneOnTwo,1,10)
    );
}
```

C.2 Cas clinique

C.2.1 Domaine 1 : déplacement résidence – centre de jour

```
Variable v_patient = new Variable();

//      method for saturday activities
Task t_satAct = new Task("saturdayActivities");
t_satAct.setParam(v_patient);
Method m_satAct = new Method("DoSaturdayActivities", t_satAct);
m_satAct.setParam(v_patient);
m_satAct.setPrecond();
    Task t_sat = new Task("Aller au centre de jour");
    t_sat.setParam(v_patient);
m_satAct.setSubtask(t_sat);
m_satAct.setSubtaskRelation(new Timezone(t_sat, Timezone.SATURDAY));

//      method for reaching day center
Task t_dayCenter = new Task("Aller au centre de jour");
t_dayCenter.setParam(v_patient);
Method m_dayCenter = new Method("DoGoDayCenter", t_dayCenter);
m_dayCenter.setParam(v_patient);
m_dayCenter.setPrecond();
    Task t_takeBus = new Task("Prendre bus jusqu'au centre");
    t_takeBus.setParam(v_patient);
    Task t_walk = new Task("Marcher jusqu'au centre");
    t_walk.setParam(v_patient);
    Task t_getBack = new Task("Prendre navette de retour");
    t_getBack.setParam(v_patient);
m_dayCenter.setSubtask(t_takeBus, t_walk, t_getBack);
m_dayCenter.setSubtaskRelation(new Precedence(t_takeBus, t_walk),
                               new Precedence(t_walk, t_getBack));
m_dayCenter.setExplicitOn(new GranularityType("explicitDayCenter"));

//            method for taking the bus
Task t_busToCenter = new Task("Prendre bus jusqu'au centre");
t_busToCenter.setParam(v_patient);
Method m_busToCenter = new Method("DoBusToDayCenter", t_busToCenter);
m_busToCenter.setParam(v_patient);
m_busToCenter.setPrecond();
    Primitive p_bus01 = new Primitive("Aller à l'arrêt Tram Université");
    p_bus01.setParam(v_patient);
    p_bus01.setPrecond();
    p_bus01.setTimeCellLength(1);
    Primitive p_bus02 = new Primitive("Prendre Tram direction de Université Claude Bernard");
    p_bus02.setParam(v_patient);
    p_bus02.setPrecond();
    p_bus02.setTimeCellLength(1);
    Primitive p_bus03 = new Primitive("Descendre à l'Arrêt Feyssine");
    p_bus03.setParam(v_patient);
    p_bus03.setPrecond();
    p_bus03.setTimeCellLength(1);
m_busToCenter.setSubtask(p_bus01, p_bus02, p_bus03);
    Vector<Action> agg = new Vector<Action>();
    agg.add(p_bus01);
    agg.add(p_bus02);
m_busToCenter.setSubtaskRelation(new BeginAtDaySpot(p_bus01, 36),
                                 new Precedence(p_bus01, p_bus02),
                                 new Agglomerate(agg),
                                 new BeginAtDaySpot(p_bus03, 39) );
m_busToCenter.setExplicitOn(new GranularityType("explicitTakeBus"));

//            method for walking to center
```

```
Task t_walkToCenter = new Task("Marcher jusqu'au centre");
t_walkToCenter.setParam(v_patient);
Method m_walkToCenter = new Method("DoWalkToCenter", t_walkToCenter);
m_walkToCenter.setParam(v_patient);
m_walkToCenter.setPrecond();
    Primitive p_walk01 = new Primitive("Tourner à gauche sur Rue de la république");
    p_walk01.setParam(v_patient);
    p_walk01.setPrecond();
    p_walk01.setTimeCellLength(0);
    Primitive p_walk02 = new Primitive("Tourner à droite sur la Place Jean Macé");
    p_walk02.setParam(v_patient);
    p_walk02.setPrecond();
    p_walk02.setTimeCellLength(0);
    Primitive p_walk03 = new Primitive("Tourner à gauche sur la Rue Philippe Pinel");
    p_walk03.setParam(v_patient);
    p_walk03.setPrecond();
    p_walk03.setTimeCellLength(0);
    Primitive p_walk04 = new Primitive("Aller jusqu'au n°72 de la Rue Philippe Pinel");
    p_walk04.setParam(v_patient);
    p_walk04.setPrecond();
    p_walk04.setTimeCellLength(0);
    Primitive p_walk05 = new Primitive("Aller au local 505, 5eme étage");
    p_walk05.setParam(v_patient);
    p_walk05.setPrecond();
    p_walk05.setTimeCellLength(0);
m_walkToCenter.setSubtask(p_walk01, p_walk02, p_walk03, p_walk04, p_walk05);
m_walkToCenter.setSubtaskRelation(new Precedence(p_walk05, p_walk04),
                                  new Precedence(p_walk04, p_walk03),
                                  new Precedence(p_walk03, p_walk02),
                                  new Precedence(p_walk02, p_walk01),
                                  new BeginAtDaySpot(p_walk01, 40),
                                  new BeginAtDaySpot(p_walk02, 40),
                                  new BeginAtDaySpot(p_walk03, 40),
                                  new BeginAtDaySpot(p_walk04, 40),
                                  new BeginAtDaySpot(p_walk05, 40));
m_walkToCenter.setExplicitOn(new GranularityType("explicitWalk"));

//            method for walking to center
Task t_takeShuttleBack = new Task("Prendre navette de retour");
t_takeShuttleBack.setParam(v_patient);
Method m_takeShuttleBack = new Method("DoTakeShuttleBack", t_takeShuttleBack);
m_takeShuttleBack.setParam(v_patient);
m_takeShuttleBack.setPrecond();
    Primitive p_shuttle01 = new Primitive("Aller à l'arrêt de la navette");
    p_shuttle01.setParam(v_patient);
    p_shuttle01.setPrecond();
    p_shuttle01.setTimeCellLength(0);
    Primitive p_shuttle02 = new Primitive("Prendre navette");
    p_shuttle02.setParam(v_patient);
    p_shuttle02.setPrecond();
    p_shuttle02.setTimeCellLength(0);
m_takeShuttleBack.setSubtask(p_shuttle01, p_shuttle02);
m_takeShuttleBack.setSubtaskRelation(new Precedence(p_shuttle02, p_shuttle01),
                                     new BeginAtDaySpot(p_shuttle02, 44),
                                     new BeginAtDaySpot(p_shuttle01, 44));
m_takeShuttleBack.setExplicitOn(new GranularityType("explicitShuttle"));
```

C.2.2 Domaine 2 : journée – médication, prise de repas, autres activités

```
Variable v_patient = new Variable();

// method for saturday activities
Task t_satAct = new Task("saturdayActivities");
t_satAct.setParam(v_patient);

Method m_satAct = new Method("DoSaturdayActivities", t_satAct);
m_satAct.setParam(v_patient);
m_satAct.setPrecond();
      Task t_meds = new Task("dailyMedication");
      t_meds.setParam(v_patient);
      Task t_food = new Task("dailyFood");
      t_food.setParam(v_patient);
if (testCase == 0) {
      Task t_chess = new Task("chess");
      t_chess.setParam(v_patient);
      m_satAct.setSubtask(t_food, t_meds, t_chess);
      m_satAct.setSubtaskRelation(new Timezone(t_meds, Timezone.SATURDAY),
                                  new Timezone(t_food, Timezone.SATURDAY),
                                  new Timezone(t_chess, Timezone.SATURDAY));
}
if (testCase == 1) {
      Task t_chess = new Task("chess");
      t_chess.setParam(v_patient);
      m_satAct.setSubtask(t_meds, t_food);
      m_satAct.setSubtaskRelation(new Timezone(t_meds, Timezone.SATURDAY),
                                  new Timezone(t_food, Timezone.SATURDAY));
}
if (testCase == 2) {
      Task t_chess = new Task("chess");
      t_chess.setParam(v_patient);
      t_chess.setPreferenceRank(100);
      Task t_clinic = new Task("clinic appointment");
      t_clinic.setParam(v_patient);
      m_satAct.setSubtask(t_food, t_meds, t_clinic, t_chess);
      m_satAct.setSubtaskRelation(new Timezone(t_meds, Timezone.SATURDAY),
                                  new Timezone(t_clinic, Timezone.SATURDAY),
                                  new Timezone(t_chess, Timezone.SATURDAY),
                                  new Timezone(t_food, Timezone.SATURDAY));
}
if (testCase == 3) {
      Task t_chess = new Task("chess");
      t_chess.setParam(v_patient);
      t_chess.setPreferenceRank(100);
      Task t_clinic = new Task("clinic appointment");
      t_clinic.setParam(v_patient);
      m_satAct.setSubtask(t_food, t_meds, t_clinic, t_chess);
      m_satAct.setSubtaskRelation(new Timezone(t_meds, Timezone.SATURDAY),
                                  new Timezone(t_clinic, Timezone.SATURDAY),
                                  new Timezone(t_chess, Timezone.SATURDAY),
                                  new Timezone(t_food, Timezone.SATURDAY));
}

// method hierarchy for daily medication
Task t_dailyMed = new Task("dailyMedication");
t_dailyMed.setParam(v_patient);

Method m_dailyMed = new Method("TakeDailyMed", t_dailyMed);
m_dailyMed.setParam(v_patient);
m_dailyMed.setPrecond();
      Primitive p_takemorningMed = new Primitive("takeMedication");
      p_takemorningMed.setParam(v_patient, new Variable("Leponex"),
```

```
                                        new Variable("Tardiferon"));
      p_takemorningMed.setPrecond();
      p_takemorningMed.setTimeCellLength(1);
      p_takemorningMed.setMOBUSExportString("Prendre 2 Leponex et 1 Tardiferon");
      Primitive p_takeeveningMed = new Primitive("takeMedication");
      p_takeeveningMed.setParam(v_patient, new Variable("Leponex"));
      p_takeeveningMed.setPrecond();
      p_takeeveningMed.setTimeCellLength(1);
      p_takeeveningMed.setMOBUSExportString("Prendre 2 Leponex");
if (testCase == 0) {
      m_dailyMed.setSubtask(p_takemorningMed, p_takeeveningMed);
      m_dailyMed.setSubtaskRelation(new Timezone(p_takemorningMed, Timezone.MORNING),
                                        new Timezone(p_takeeveningMed, Timezone.EVENING),
                                        new SameDay(p_takemorningMed, p_takeeveningMed),
                                        new BeginAtDaySpot(p_takemorningMed, 32, 20),
                                        new BeginAtDaySpot(p_takeeveningMed, 76, 50));
}
if (testCase == 1) {
      m_dailyMed.setSubtask(p_takemorningMed, p_takeeveningMed);
      m_dailyMed.setSubtaskRelation(new Timezone(p_takemorningMed, Timezone.MORNING),
                                        new Timezone(p_takeeveningMed, Timezone.EVENING),
                                        new SameDay(p_takemorningMed, p_takeeveningMed),
                                        new BeginAtDaySpot(p_takemorningMed, 32, 20),
                                        new BeginAtDaySpot(p_takeeveningMed, 76, 50));
}
if (testCase == 2) {
      m_dailyMed.setSubtask(p_takemorningMed);
      m_dailyMed.setSubtaskRelation(new Timezone(p_takemorningMed, Timezone.MORNING),
                                        new BeginAtDaySpot(p_takemorningMed, 30));
}
if (testCase == 3) {
      m_dailyMed.setSubtask(p_takemorningMed);
      m_dailyMed.setSubtaskRelation(new Timezone(p_takemorningMed, Timezone.MORNING),
                                        new BeginAtDaySpot(p_takemorningMed, 30));
}


// method hierarchy for daily food
Task t_dailyFood = new Task("dailyFood");
t_dailyFood.setParam(v_patient);

Method m_dailyFood = new Method("TakeDailyFood", t_dailyFood);
m_dailyFood.setParam(v_patient);
m_dailyFood.setPrecond();
      Primitive p_breakfast = new Primitive("breakfast");
      p_breakfast.setParam(v_patient);
      p_breakfast.setPrecond();
      p_breakfast.setTimeCellLength(2);
      p_breakfast.setMOBUSExportString("Déjeuner");
      Primitive p_lunch = new Primitive("lunch");
      p_lunch.setParam(v_patient);
      p_lunch.setPrecond();
      p_lunch.setTimeCellLength(3);
      p_lunch.setMOBUSExportString("Dîner");
      Primitive p_dinner = new Primitive("dinner");
      p_dinner.setParam(v_patient);
      p_dinner.setPrecond();
      p_dinner.setTimeCellLength(4);
      p_dinner.setMOBUSExportString("Souper");
if (testCase == 0) {
      m_dailyFood.setSubtask(p_breakfast, p_lunch, p_dinner);
      m_dailyFood.setSubtaskRelation(new Timezone(p_breakfast, Timezone.MORNING),
                                        new Timezone(p_lunch, Timezone.AFTERNOON),
                                        new Timezone(p_dinner, Timezone.EVENING),
                                        new SameDay(p_breakfast, p_lunch),
                                        new SameDay(p_breakfast, p_dinner),
                                        new BeginAtDaySpot(p_breakfast, 34),
```

```
                                        new BeginAtDaySpot(p_lunch, 49),
                                        new BeginAtDaySpot(p_dinner, 72));
}
if (testCase == 1) {
      m_dailyFood.setSubtask(p_breakfast, p_lunch, p_dinner);
      m_dailyFood.setSubtaskRelation(new Timezone(p_breakfast, Timezone.MORNING),
                                        new Timezone(p_lunch, Timezone.AFTERNOON),
                                        new Timezone(p_dinner, Timezone.EVENING),
                                        new SameDay(p_breakfast, p_lunch),
                                        new SameDay(p_breakfast, p_dinner),
                                        new BeginAtDaySpot(p_breakfast, 34),
                                        new BeginAtDaySpot(p_lunch, 49),
                                        new BeginAtDaySpot(p_dinner, 73));
}
if (testCase == 2) {
      m_dailyFood.setSubtask(p_breakfast);
      m_dailyFood.setSubtaskRelation(new Timezone(p_breakfast, Timezone.MORNING),
                                        new BeginAtDaySpot(p_breakfast, 35));
}
if (testCase == 3) {
      m_dailyFood.setSubtask(p_breakfast);
      m_dailyFood.setSubtaskRelation(new Timezone(p_breakfast, Timezone.MORNING),
                                        new BeginAtDaySpot(p_breakfast, 35));
}

// method hierarchy chess
Task t_Chess = new Task("chess");
t_Chess.setParam(v_patient);

Method m_chess = new Method("PlayChess", t_Chess);
m_chess.setParam(v_patient);
m_chess.setPrecond();
      Primitive p_chess = new Primitive("chess");
      p_chess.setParam(v_patient);
      p_chess.setPrecond();
      p_chess.setTimeCellLength(4);
      p_chess.setMOBUSExportString("Jouer aux échecs");
m_chess.setSubtask(p_chess);
if (testCase == 0) {
      m_chess.setSubtaskRelation(new Timezone(p_chess, Timezone.MORNING));
}
if (testCase == 1) {
      m_chess.setSubtaskRelation(new Timezone(p_chess, Timezone.MORNING));
}
if (testCase == 2) {
      p_chess.setTimeCellLength(4);
      m_chess.setSubtaskRelation(new Timezone(p_chess, Timezone.MORNING));
}
if (testCase == 3) {
      p_chess.setTimeCellLength(8);
      m_chess.setSubtaskRelation(new Timezone(p_chess, Timezone.MORNING));
}

// method hierarchy fo clinic
Task t_Clinic = new Task("clinic appointment");
t_Clinic.setParam(v_patient);

Method m_clinic = new Method("ClinicAppointment", t_Clinic);
m_clinic.setParam(v_patient);
m_clinic.setPrecond();
      Primitive p_clinic = new Primitive("Clinic appointment with Dr. M");
      p_clinic.setParam(v_patient);
      p_clinic.setPrecond();
      p_clinic.setTimeCellLength(8);
      p_clinic.setMOBUSExportString("RDV avec Dr. M à la clinique");
m_clinic.setSubtask(p_clinic);
if (testCase == 0) {
```

```
      m_clinic.setSubtaskRelation(new Timezone(p_clinic, Timezone.MORNING),
                                  new BeginAt(p_clinic, 616));
}
if (testCase == 1) {
      m_clinic.setSubtaskRelation(new Timezone(p_clinic, Timezone.MORNING),
                                  new BeginAt(p_clinic, 616));
}
if (testCase == 2) {
      m_clinic.setSubtaskRelation(new Timezone(p_clinic, Timezone.MORNING),
                                  new BeginAt(p_clinic, 616));
}
if (testCase == 3) {
      m_clinic.setSubtaskRelation(new Timezone(p_clinic, Timezone.MORNING),
                                  new BeginAt(p_clinic, 616));
}
```

C.2.3 Domaine 3 : semaine complète

```
Variable v_patient = new Variable();

//      method for weekly activities
Task t_weeklyAct = new Task("weeklyActivities");
t_weeklyAct.setParam(v_patient);

Method m_weeklyAct = new Method("DoWeeklyActivities", t_weeklyAct);
m_weeklyAct.setParam(v_patient);
m_weeklyAct.setPrecond();
    Task t_DrawAct = new Task("weeklyDrawing");
    t_DrawAct.setParam(v_patient);
    Task t_Sport = new Task("weeklySport");
    t_Sport.setParam(v_patient);
    Task t_Cleaning = new Task("weeklyCleaning");
    t_Cleaning.setParam(v_patient);
    Task t_Shower = new Task("weeklyShower");
    t_Shower.setParam(v_patient);
    Task t_Other = new Task("weeklyOther");
    t_Other.setParam(v_patient);
    Task t_DayRoutine = new Task("weeklyDayRoutine");
    t_DayRoutine.setParam(v_patient);
m_weeklyAct.setSubtask(t_DrawAct, t_Sport, t_Cleaning, t_Shower, t_Other, t_DayRoutine);
m_weeklyAct.setSubtaskRelation();

//      method hierarchy for drawing
Task t_weeklyDrawing = new Task("weeklyDrawing");
t_weeklyDrawing.setParam(v_patient);

Method m_weeklyDrawing = new Method("DoWeeklyDrawing", t_weeklyDrawing);
m_weeklyDrawing.setParam(v_patient);
m_weeklyDrawing.setPrecond();
    Task t_drawA = new Task("drawing");
    t_drawA.setParam(v_patient, new Variable("SCENERY") );
    Task t_drawB = new Task("drawing");
    t_drawB.setParam(v_patient, new Variable("PORTRAIT"));
m_weeklyDrawing.setSubtask(t_drawA, t_drawB);
m_weeklyDrawing.setSubtaskRelation(new Timezone(t_drawA, Timezone.AFTERNOON),
                                   new Timezone(t_drawA, Timezone.MONDAY),
                                   new Timezone(t_drawB, Timezone.MORNING),
                                   new Timezone(t_drawB, Timezone.SATURDAY));

Variable v_subject = new Variable();
Task t_draw = new Task("drawing");
t_draw.setParam(v_patient, v_subject);
Method m_drawing = new Method("DoDrawing", t_draw);
m_drawing.setParam(v_patient);
m_drawing.setPrecond();
    Primitive p_draw = new Primitive("Drawing");
    p_draw.setParam(v_patient, v_subject);
    p_draw.setPrecond();
    p_draw.setTimeCellLength(8);
m_drawing.setSubtask(p_draw);
m_drawing.setSubtaskRelation();

//      method hierarchy for sports
Task t_weeklySport = new Task("weeklySport");
t_weeklySport.setParam(v_patient);
Method m_weeklySport = new Method("DoWeeklySport", t_weeklySport);
m_weeklySport.setParam(v_patient);
m_weeklySport.setPrecond();
    Primitive p_sport = new Primitive("Sport");
    p_sport.setParam(v_patient);
```

```
    p_sport.setPrecond();
    p_sport.setTimeCellLength(6);
    Primitive p_piscine = new Primitive("Piscine");
    p_piscine.setParam(v_patient);
    p_piscine.setPrecond();
    p_piscine.setTimeCellLength(7);
    Primitive p_bad = new Primitive("Badminton");
    p_bad.setParam(v_patient);
    p_bad.setPrecond();
    p_bad.setTimeCellLength(8);
m_weeklySport.setSubtask(p_sport, p_piscine, p_bad);
m_weeklySport.setSubtaskRelation(new BeginAt(p_sport, 236), new BeginAt(p_piscine, 438),
                                 new BeginAt(p_bad, 532));

//      method hierarchy for cleaning
Task t_weeklyCleaning = new Task("weeklyCleaning");
t_weeklyCleaning.setParam(v_patient);
Method m_weeklyCleaning = new Method("DoWeeklyCleaning", t_weeklyCleaning);
m_weeklyCleaning.setParam(v_patient);
m_weeklyCleaning.setPrecond();
    Task t_cleanA= new Task("cleanApart");
    t_cleanA.setParam(v_patient);
    Task t_cleanB = new Task("laundry");
    t_cleanB.setParam(v_patient);
m_weeklyCleaning.setSubtask(t_cleanA, t_cleanB);
m_weeklyCleaning.setSubtaskRelation(new Timezone(t_cleanA, Timezone.MORNING),
                                 new Timezone(t_cleanB, Timezone.WEDNESDAY));

Task t_cleanApart = new Task("cleanApart");
t_cleanApart.setParam(v_patient);
Method m_cleanApart = new Method("DoCleanApart", t_cleanApart);
m_cleanApart.setParam(v_patient);
m_cleanApart.setPrecond();
    Primitive p_cleanApart = new Primitive("Clean Appartment");
    p_cleanApart.setParam(v_patient);
    p_cleanApart.setPrecond();
    p_cleanApart.setTimeCellLength(6);
m_cleanApart.setSubtask(p_cleanApart);
m_cleanApart.setSubtaskRelation();

Task t_laundry = new Task("laundry");
t_laundry.setParam(v_patient);
Method m_laundry = new Method("DoLaundry", t_laundry);
m_laundry.setParam(v_patient);
m_laundry.setPrecond();
    Primitive p_laundry = new Primitive("Laundry");
    p_laundry.setParam(v_patient);
    p_laundry.setPrecond();
    p_laundry.setTimeCellLength(4);
    Primitive p_dresser = new Primitive("Put clothes in dresser");
    p_dresser.setParam(v_patient);
    p_dresser.setPrecond();
    p_dresser.setTimeCellLength(1);
m_laundry.setSubtask(p_laundry, p_dresser);
m_laundry.setSubtaskRelation(new Precedence(p_laundry, p_dresser),
                                 new Timezone(p_laundry, Timezone.MORNING),
                                 new Timezone(p_dresser, Timezone.AFTERNOON));

//      method hierarchy for showers
Task t_weeklyShower = new Task("weeklyShower");
t_weeklyShower.setParam(v_patient);

Method m_weeklyShower = new Method("DoWeeklyShower", t_weeklyShower);
m_weeklyShower.setParam(v_patient);
m_weeklyShower.setPrecond();
    Task t_showerA = new Task("shower");
    t_showerA.setParam(v_patient);
```

```
    Task t_showerB = new Task("shower");
    t_showerB.setParam(v_patient);
m_weeklyShower.setSubtask(t_showerA, t_showerB);
m_weeklyShower.setSubtaskRelation(new BeginAt(t_showerA, 37), new BeginAt(t_showerB, 325));

Task t_shower = new Task("shower");
t_shower.setParam(v_patient);
Method m_shower = new Method("DoShower", t_shower);
m_shower.setParam(v_patient);
m_shower.setPrecond();
    Primitive p_shower = new Primitive("Shower");
    p_shower.setParam(v_patient);
    p_shower.setPrecond();
    p_shower.setTimeCellLength(2);
m_shower.setSubtask(p_shower);
m_shower.setSubtaskRelation();

//      method hierarchy for misc activities
Task t_weeklyOther = new Task("weeklyOther");
t_weeklyOther.setParam(v_patient);

Method m_weeklyOther = new Method("DoWeeklyOther", t_weeklyOther);
m_weeklyOther.setParam(v_patient);
m_weeklyOther.setPrecond();
    Primitive p_houseMeeting = new Primitive("House meeting");
    p_houseMeeting.setParam(v_patient);
    p_houseMeeting.setPrecond();
    p_houseMeeting.setTimeCellLength(6);
    Primitive p_mealTherapy = new Primitive("Meal therapy");
    p_mealTherapy.setParam(v_patient);
    p_mealTherapy.setPrecond();
    p_mealTherapy.setTimeCellLength(8);
    Primitive p_socialCenter = new Primitive("Social Center");

    p_socialCenter.setParam(v_patient);
    p_socialCenter.setPrecond();
    p_socialCenter.setTimeCellLength(6);
m_weeklyOther.setSubtask(p_houseMeeting, p_mealTherapy, p_socialCenter);
m_weeklyOther.setSubtaskRelation(new BeginAt(p_houseMeeting, 452),
                                 new BeginAt(p_mealTherapy, 328),
                                 new BeginAt(p_socialCenter, 150));

//      method hierarchy for day routine
Task t_weeklyDayRoutine = new Task("weeklyDayRoutine");
t_weeklyDayRoutine.setParam(v_patient);

Method m_weeklyDayRoutine = new Method("DoWeeklyDayRoutine", t_weeklyDayRoutine);
m_weeklyDayRoutine.setParam(v_patient);
m_weeklyDayRoutine.setPrecond();
    Task t_sEOD = new Task("dailyRoutine");
    t_sEOD.setParam(v_patient);
    Task t_mEOD = new Task("dailyRoutine");
    t_mEOD.setParam(v_patient);
    Task t_tEOD = new Task("dailyRoutine");
    t_tEOD.setParam(v_patient);
    Task t_wEOD = new Task("dailyRoutine");
    t_wEOD.setParam(v_patient);
    Task t_ttEOD = new Task("dailyRoutine");
    t_ttEOD.setParam(v_patient);
    Task t_fEOD = new Task("dailyRoutine");
    t_fEOD.setParam(v_patient);
    Task t_ssEOD = new Task("dailyRoutine");
    t_ssEOD.setParam(v_patient);
m_weeklyDayRoutine.setSubtask(t_sEOD, t_mEOD, t_tEOD, t_wEOD, t_ttEOD, t_fEOD, t_ssEOD);
m_weeklyDayRoutine.setSubtaskRelation(new Timezone(t_sEOD, Timezone.SUNDAY),
                                      new Timezone(t_mEOD, Timezone.MONDAY),
                                      new Timezone(t_tEOD, Timezone.TUESDAY),
```

```
                                        new Timezone(t_wEOD, Timezone.WEDNESDAY),
                                        new Timezone(t_ttEOD, Timezone.THURSDAY),
                                        new Timezone(t_fEOD, Timezone.FRIDAY),
                                        new Timezone(t_ssEOD, Timezone.SATURDAY));

Task t_dailyRoutine = new Task("dailyRoutine");
t_dailyRoutine.setParam(v_patient);
Method m_dailyRoutine = new Method("DoDailyEOD", t_dailyRoutine);
m_dailyRoutine.setParam(v_patient);
m_dailyRoutine.setPrecond();
    Primitive p_dirty = new Primitive("Separate dirty clothes");
    p_dirty.setParam(v_patient);
    p_dirty.setPrecond();
    p_dirty.setTimeCellLength(1);
    Primitive p_recharge = new Primitive("Recharge agenda");
    p_recharge.setParam(v_patient);
    p_recharge.setPrecond();
    p_recharge.setTimeCellLength(3);
    Primitive p_takemorningMed = new Primitive("takeMedication");
    p_takemorningMed.setParam(v_patient, new Variable("Leponex"),
                                    new Variable("Tardiferon"),
                                    new Variable("Stomach pill"));
    p_takemorningMed.setPrecond();
    p_takemorningMed.setTimeCellLength(1);
    Primitive p_takeeveningMed = new Primitive("takeMedication");
    p_takeeveningMed.setParam(v_patient, new Variable("Leponex"),
                                    new Variable("Stomach pill"));
    p_takeeveningMed.setPrecond();
    p_takeeveningMed.setTimeCellLength(1);
m_dailyRoutine.setSubtask(p_dirty, p_recharge, p_takemorningMed, p_takeeveningMed);
m_dailyRoutine.setSubtaskRelation(new BeginAtDaySpot(p_dirty, 83),
                                    new BeginAtDaySpot(p_recharge, 84),
                                    new BeginAtDaySpot(p_takemorningMed, 36),
                                    new BeginAtDaySpot(p_takeeveningMed, 76));
```

Annexe D

Sortie du planificateur

D.1 Cas clinique, domaine 3 : semaine complète

Une sortie typique du planificateur

```
WEEKLY ACTIVITIES AND MEDICATION
Planning...
WEEK SCHEDULE:

SUNDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
  03:15:
  03:30:
  03:45:
  04:00:
  04:15:
  04:30:
  04:45:
  05:00:
  05:15:
  05:30:
  05:45:
  06:00:
  06:15:
  06:30:
  06:45:
  07:00:
  07:15:
  07:30:
  07:45:
  08:00:
  08:15:
  08:30:
  08:45:
  09:00: (PRIMITIVE takeMedication (&115 PATIENT) (&128 Leponex) (&129 Tardiferon)
         (&12A Stomach pill))
  09:15: (PRIMITIVE Shower (&53A PATIENT))
```

```
  09:30:
  09:45:
  10:00:
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  11:30:
  11:45:
  12:00:
  12:15:
  12:30:
  12:45:
  13:00:
  13:15:
  13:30:
  13:45:
  14:00:
  14:15:
  14:30:
  14:45:
  15:00:
  15:15:
  15:30:
  15:45:
  16:00:
  16:15:
  16:30:
  16:45:
  17:00:
  17:15:
  17:30:
  17:45:
  18:00:
  18:15:
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&F2 PATIENT) (&10A Leponex) (&10B Stomach pill))
  19:15:
  19:30:
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&2EE PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&5BC PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
MONDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
```

```
02:00:
02:15:
02:30:
02:45:
03:00:
03:15:
03:30:
03:45:
04:00:
04:15:
04:30:
04:45:
05:00:
05:15:
05:30:
05:45:
06:00:
06:15:
06:30:
06:45:
07:00:
07:15:
07:30:
07:45:
08:00:
08:15:
08:30:
08:45:
09:00: (PRIMITIVE takeMedication (&509 PATIENT) (&516 Leponex) (&517 Tardiferon)
       (&518 Stomach pill))
09:15:
09:30:
09:45:
10:00:
10:15:
10:30:
10:45:
11:00:
11:15: (PRIMITIVE Clean Appartment (&490 PATIENT))
11:30:
11:45:
12:00:
12:15:
12:30:
12:45:
13:00:
13:15:
13:30: (PRIMITIVE Social Center (&21 PATIENT))
13:45:
14:00:
14:15:
14:30:
14:45:
15:00:
15:15:
15:30:
15:45:
16:00: (PRIMITIVE Drawing (&55F PATIENT) (&56E SCENERY))
16:15:
16:30:
16:45:
17:00:
17:15:
17:30:
17:45:
18:00:
18:15:
```

```
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&522 PATIENT) (&532 Leponex) (&533 Stomach pill))
  19:15:
  19:30:
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&5A6 PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&4ED PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
TUESDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
  03:15:
  03:30:
  03:45:
  04:00:
  04:15:
  04:30:
  04:45:
  05:00:
  05:15:
  05:30:
  05:45:
  06:00:
  06:15:
  06:30:
  06:45:
  07:00:
  07:15:
  07:30:
  07:45:
  08:00:
  08:15:
  08:30:
  08:45:
  09:00: (PRIMITIVE takeMedication (&2D5 PATIENT) (&2DA Leponex) (&2DB Tardiferon)
         (&2DC Stomach pill))
  09:15:
  09:30:
  09:45:
  10:00:
  10:15:
  10:30:
```

```
  10:45:
  11:00: (PRIMITIVE Sport (&2BB PATIENT))
  11:15:
  11:30:
  11:45:
  12:00:
  12:15:
  12:30:
  12:45:
  13:00:
  13:15:
  13:30:
  13:45:
  14:00:
  14:15:
  14:30:
  14:45:
  15:00:
  15:15:
  15:30:
  15:45:
  16:00:
  16:15:
  16:30:
  16:45:
  17:00:
  17:15:
  17:30:
  17:45:
  18:00:
  18:15:
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&139 PATIENT) (&145 Leponex) (&146 Stomach pill))
  19:15:
  19:30:
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&A5 PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&177 PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
WEDNESDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
```

```
03:15:
03:30:
03:45:
04:00:
04:15:
04:30:
04:45:
05:00:
05:15:
05:30:
05:45:
06:00:
06:15:
06:30:
06:45:
07:00:
07:15:
07:30:
07:45:
08:00: (PRIMITIVE Laundry (&5AD PATIENT))
08:15:
08:30:
08:45:
09:00: (PRIMITIVE takeMedication (&80 PATIENT) (&9F Leponex) (&A0 Tardiferon)
       (&A1 Stomach pill))
09:15: (PRIMITIVE Shower (&59E PATIENT))
09:30:
09:45:
10:00: (PRIMITIVE Meal therapy (&2A0 PATIENT))
10:15:
10:30:
10:45:
11:00:
11:15:
11:30:
11:45:
12:00:
12:15:
12:30:
12:45:
13:00:
13:15:
13:30:
13:45:
14:00:
14:15: (PRIMITIVE Put laundry away (&5B3 PATIENT))
14:30:
14:45:
15:00:
15:15:
15:30:
15:45:
16:00:
16:15:
16:30:
16:45:
17:00:
17:15:
17:30:
17:45:
18:00:
18:15:
18:30:
18:45:
19:00: (PRIMITIVE takeMedication (&191 PATIENT) (&1A2 Leponex) (&1A3 Stomach pill))
19:15:
19:30:
```

```
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&3B3 PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&15A PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
THURSDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
  03:15:
  03:30:
  03:45:
  04:00:
  04:15:
  04:30:
  04:45:
  05:00:
  05:15:
  05:30:
  05:45:
  06:00:
  06:15:
  06:30:
  06:45:
  07:00:
  07:15:
  07:30:
  07:45:
  08:00:
  08:15:
  08:30:
  08:45:
  09:00: (PRIMITIVE takeMedication (&46E PATIENT) (&47A Leponex) (&47B Tardiferon)
         (&47C Stomach pill))
  09:15:
  09:30:
  09:45:
  10:00:
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  11:30:
  11:45:
```

```
  12:00:
  12:15:
  12:30:
  12:45:
  13:00:
  13:15:
  13:30: (PRIMITIVE Swimming (&285 PATIENT))
  13:45:
  14:00:
  14:15:
  14:30:
  14:45:
  15:00:
  15:15:
  15:30:
  15:45:
  16:00:
  16:15:
  16:30:
  16:45:
  17:00: (PRIMITIVE House meeting (&58B PATIENT))
  17:15:
  17:30:
  17:45:
  18:00:
  18:15:
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&56F PATIENT) (&577 Leponex) (&578 Stomach pill))
  19:15:
  19:30:
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&36E PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&4B2 PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
FRIDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
  03:15:
  03:30:
  03:45:
  04:00:
  04:15:
```

```
04:30:
04:45:
05:00:
05:15:
05:30:
05:45:
06:00:
06:15:
06:30:
06:45:
07:00:
07:15:
07:30:
07:45:
08:00:
08:15:
08:30:
08:45:
09:00: (PRIMITIVE takeMedication (&34B PATIENT) (&353 Leponex) (&354 Tardiferon)
       (&355 Stomach pill))
09:15:
09:30:
09:45:
10:00:
10:15:
10:30:
10:45:
11:00:
11:15:
11:30:
11:45:
12:00:
12:15:
12:30:
12:45:
13:00: (PRIMITIVE Badminton (&595 PATIENT))
13:15:
13:30:
13:45:
14:00:
14:15:
14:30:
14:45:
15:00:
15:15:
15:30:
15:45:
16:00:
16:15:
16:30:
16:45:
17:00:
17:15:
17:30:
17:45:
18:00:
18:15:
18:30:
18:45:
19:00: (PRIMITIVE takeMedication (&232 PATIENT) (&24A Leponex) (&24B Stomach pill))
19:15:
19:30:
19:45:
20:00:
20:15:
20:30:
20:45: (PRIMITIVE Separate dirty clothes (&1D9 PATIENT))
```

```
  21:00: (PRIMITIVE Recharge agenda (&1F6 PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
SATURDAY
  00:00:
  00:15:
  00:30:
  00:45:
  01:00:
  01:15:
  01:30:
  01:45:
  02:00:
  02:15:
  02:30:
  02:45:
  03:00:
  03:15:
  03:30:
  03:45:
  04:00:
  04:15:
  04:30:
  04:45:
  05:00:
  05:15:
  05:30:
  05:45:
  06:00:
  06:15:
  06:30:
  06:45:
  07:00:
  07:15:
  07:30:
  07:45:
  08:00:
  08:15:
  08:30:
  08:45:
  09:00: (PRIMITIVE takeMedication (&421 PATIENT) (&440 Leponex) (&441 Tardiferon)
         (&442 Stomach pill))
  09:15: (PRIMITIVE Drawing (&5B8 PATIENT) (&5BB PORTRAIT))
  09:30:
  09:45:
  10:00:
  10:15:
  10:30:
  10:45:
  11:00:
  11:15:
  11:30:
  11:45:
  12:00:
  12:15:
  12:30:
  12:45:
  13:00:
```

```
  13:15:
  13:30:
  13:45:
  14:00:
  14:15:
  14:30:
  14:45:
  15:00:
  15:15:
  15:30:
  15:45:
  16:00:
  16:15:
  16:30:
  16:45:
  17:00:
  17:15:
  17:30:
  17:45:
  18:00:
  18:15:
  18:30:
  18:45:
  19:00: (PRIMITIVE takeMedication (&4D0 PATIENT) (&4E7 Leponex) (&4E8 Stomach pill))
  19:15:
  19:30:
  19:45:
  20:00:
  20:15:
  20:30:
  20:45: (PRIMITIVE Separate dirty clothes (&57E PATIENT))
  21:00: (PRIMITIVE Recharge agenda (&54E PATIENT))
  21:15:
  21:30:
  21:45:
  22:00:
  22:15:
  22:30:
  22:45:
  23:00:
  23:15:
  23:30:
  23:45:
Found schedule in 0:00:00:922 (922 ms)
```

Annexe E

Relations implémentées

Suivent les relations implémentées par la présente version du planificateur. Il est possible d'en définir d'autres. Du point de vue de l'algorithme de planification, toute combinaison de relations est valide, même si celle-ci comporte des éléments contradictoires (par exemple, positionner une même tâche à deux blocs horaires différents).

`Agglomerate(task1, task2, ..., taskN)`
Les tâches spécifiées doivent être positionnées dans des blocs horaire formant une séquence continue. Les tâches peuvent être dans n'importe quel ordre à l'intérieur de cette séquence.

`BeginAt(task, position)`
La tâche spécifiée doit être positionnée à la position exacte de la grille horaire spécifiée.

`BeginAtDaySpot(task, dayPosition)`
La tâche spécifiée doit être positionnée à la position du jour de la semaine de la grille horaire spécifiée (par exemple, le bloc correspondant à 8h30, peu importe le jour de la semaine).

`Precedence(task1, task2)`
La première tâche doit être positionnée avant la seconde.

`SameDay(task1, task2)`
Les deux tâches doivent être positionnées dans le même jour de semaine (par exemple, les deux tâches doivent être positionnées le jeudi).

`Timezone(task, zone)`
La tâche doit être positionnée dans la zone spécifiée. Les zones possibles sont :

- Matin
- Après-midi
- Soir
- Nuit
- Jour de la semaine

Bibliographie

[1] Bacchus F., Kabanza F. *Planning for temporally extended goals*. Annals of mathematics and artificial intelligence, Vol. 22, pp 5-27, 1998

[2] Bacchus F., Kabanza F. *Using temporal logic to express search control knowledge for planning*. Artificial intelligence, Vol. 116, pp 123-191, 2000

[3] Baier J., Bacchus F., McIlraith S. *A heuristic search approach to planning with temporally extended preferences*. Proceedings of the 20th International joint conference on artificial intelligence, Hyderabad, pp 1808-1815, 2007

[4] Beaudry E., Kabanza F., Michaud F. *Planning for a mobile robot to attend a conference*. Proceedings of the 18th Canadian conference on artificial intelligence, Montréal, pp 48-52, 2005

[5] Fikes R., Nilsson N. *STRIPS: a new approach to the application of theorem proving to problem solving*. Artificial intelligence, Vol. 2, pp 189-208, 1971

[6] Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: elements of reusable object-oriented software*. Pearson Education, 1995

[7] Ghallab M., Nau D., Traverso P. *Automated planning theory and practice, Chapter 1 – Introduction and overview*. Morgan Kaufman, 2004

[8] Ghallab M., Nau D., Traverso P. *Automated planning theory and practice, Chapter 11 – Hierarchical task network planning*. Morgan Kaufman, 2004

[9] Giroux S., Pigot H. *Indoors pervasive computing and outdoors mobile computing for cognitive assistance and telemonitoring*. 9th International conference on computers helping people with special needs, Paris, 2004

[10] Goetz B., Peirls T., Bloch J., Bowbeer J., Holmes D., Lea D. *Java concurrency in practice*. Addison-Wesley Professional, 2006

[11] IPC - International Planning Competition : competition domains, 2002 http://planning.cis.strath.ac.uk/competition/

[12] Lesak M. D., Howieson D. B., Loring D. W. *Neuropsychological assessment, Chapter 7 – Neuropathology for neuropathologists*. Oxford Press, 2004

[13] Levinson R. *A general programming language for unified planning and control.* Artificial intelligence special issue on planning and scheduling, Vol. 76, pp 51-63, 1995

[14] Levinson R. *PEAT – The planning execution assistant and trainer*. Journal of head trauma rehabilitation, 1997

[15] Linz P. *An introduction to formal languages and Automata, Chapter 1 – Introduction to the theory of computation.* Jones and Bartlett, 2001

[16] Lussier-Desrochers D., Lachapelle Y., Pigot H., Bauchet J. *Des habitats intelligents pour promouvoir l'autodétermination et l'inclusion sociale.* Revue francophone de déficience intellectuelle, Vol. 18, pp 53-64, 2007

[17] Means W., Harold E. *XML in a nutshell, Chapter 2 – XML fundamentals*. O'Reilly, 2004

[18] Meyer B. *Object-Oriented software construction.* Prentice Hall, 1997

[19] Moreau J.F. *Évaluation d'un système d'assistance mobile conçu pour des personnes ayant des troubles cognitifs et leurs aidants, Chapitre 4 – Conception du système d'assistance mobile*. Département d'informatique, Université de Sherbrooke, 2007

[20] Nau D., Au T.-C., Ilghami O., Kuter U., Murdock W., Wu D, Yaman F. *SHOP2: an HTH planning system.* Journal of artificial intelligence research, Vol. 20, pp 379-404, 2003

[21] Nau D., Cao Y., Lotem A., Munoz-Avila H. *SHOP: simple hierarchical Ordered Planner*. Proceedings of the international joint conference on artificial intelligence, pp 968-975, 1999

[22] Paccoud B. *La conception centrée utilisateur pour les personnes avec troubles cognitifs : application à l'orthèse cognitive MOBUS, Chapitre 6 – Mise en oeuvre des*

resultats de l'évaluation. Département d'informatique, Université de Sherbrooke, 2007

[23] Pigot H., Pache D., Paccoud B., Giroux S., Savary J.-P., Stip E., Sablier J. *MOBUS : Agenda d'aide aux déplacements.* Festival of international conferences on caregiving, disability, aging and technology, Toronto, 2007

[24] Pigot H., Savary J.-P., Metzger J.-L., Rochon A., Beaulieu M. *Advanced technology guidelines to fulfill the needs of the cognitively impaired population,* 3rd International conference on smart homes and health telematic, Canada, 2005

[25] Pollack M., Brown L., Colbry D., McCarthy C.E., Orosz C., Peintner B., Ramakrishnan S., Matthews T., Engberg S., Thrun S., Dunbar-Jacob J., Pineau J., Roy N. *Pearl: a mobile robotic assistant for the elderly.* AAAI Workshop on automation as eldercare, 2002

[26] Pollack M., Brown L., Colbry D., McCarthy C.E., Orosz C., Peintner B., Ramakrishnan S., Tsmardinos I. *AUTOMINDER : an intelligent cognitive system for people with memory impairment.* Robotics and autonomous systems, Vol. 44, pp 273-282, 2003

[27] Refanidis I., Alediadis A. *SelfPlanner: planning your time!.* ICAPS 2008 Workshop on scheduling and planning applications, Sydney, 2008

[28] Reisberg B. *Signs, symptoms and course of age-associated cognitive decline.* Corkin S. et al., Aging : Alzheimer's disease: a report progress, New York, Raven Press, 1982

[29] Russell S., Norvig P. *Intelligence artificielle, Chapitre 4 – Exploration informée.* Pearson Education, 2006

[30] Russell S., Norvig P. *Intelligence artificielle, Chapitre 5 – Problèmes à satisfaction de contraintes.* Pearson Education, 2006

[31] Russell S., Norvig P. *Intelligence artificielle, Chapitre 11 – Planification.* Pearson Education, 2006

[32] Schulze H. *MEMOS: an interactive assistive system for prospective memory deficit compensation – architecture and functionality,* Proceedings of the 6th international

ACM SIGACCESS conference on computers and accessibility, New York, pp 79-82, 2005

[33] Tsang E. *Foundations of constraint satisfaction, Chapter 1 – Introduction.* Academic Press, 1993

[34] Wikipedia.org – *PDA description* http://en.wikipedia.org/wiki/Personal_digital_assistant

[35] Wikipedia.org – *Smartphone description* http://en.wikipedia.org/wiki/Smartphone

[36] Wolraich M., Schor D. *Disorders of mental development.* Wolraich M, Disorders of development and learning: a practical guide to assessment and management, St-Louis, Mosby, 1996.