Chapter 1

Résumé

1.1 Introduction

Cette thèse porte sur les techniques de description et de validation du comportement d'une catégorie de systèmes temps-réel, que nous appelons *systèmes temporisés*. Conformément à une définition généralement acceptée, les systèmes temps-réel sont des systèmes dont le fonctionnement correct dépend de la satisfaction de certaines contraintes temporelles. Parmi ces systèmes, nous désignons comme *temporisés* ceux dont le comportement est *contrôlé* ou *conditionné* par le temps, à la différence des systèmes où le temps apparat seulement par l'intermédiaire des facteurs de performance.

Une méthode de validation des systèmes temps-réel doit tenir compte à la fois des contraintes fonctionnelles et des contraintes temporelles appliquées au système. Néanmoins, pour la majorité des systèmes temps-réel, les deux aspects peuvent être traités séparément, par exemple en utilisant des modèles spécifiques pour chacun d'eux. Plusieurs classes de modèles et de méthodes d'analyse, portant sur un aspect particulier du système modélisé, sont habituellement utilisées dans l'ingénierie des systèmes temps-réel. Nous mentionnerons en particulier les *modèles d'ordonnancement* et les *modèles de performance*. Les *modèles d'ordonnancement* (voir la monographie [KRPO93]) visent à étudier les aspects de compétition pour les ressources (y compris le temps de calcul), et peuvent fournir une base pour la validation du comportement temporel d'un système. Cependant, ils ne sont pas applicables aux systèmes complexes dont le fonctionnement dépend du temps (tels que les systèmes temporisés). Les *modèles de performance* visent à décrire les systèmes d'un point de vue probabiliste, et peuvent être utilisés pour la validation des contraintes de performance; cependant, leur capacité à décrire le comportement du système reste limitée.

Dans le cas des systèmes temporisés, examinés dans cette thèse, les aspects comportementaux et les aspects temporels d'un système ne peuvent être séparés, et une méthode de validation doit porter sur un modèle hybride incluant les deux. Pour cette raison, nous nous intéressons aux modèles fonctionnels (comportementaux) d'un système, et à leur extension avec des constructions pour exprimer les informations temporelles. Plus précisément, nous étudions les *méthodes formelles* de description et de validation, s'appuyant sur des langages formalisés tels que LDS [IT99b], et sur des méthodes de validation telles que la simulation ou la vérification par *model-checking* [QS82, CES86].

Le point de départ de cette thèse est l'écart que nous avons constaté entre l'état de l'art et l'état de la pratique industrielle dans le domaine de la spécification et de la vérification des systèmes temporisés. D'une part, plusieurs langages de modélisation, tels que LDS [IT99b],

HRT-HOOD [BW95], ROOM [SGW94] ou UML avec des extensions temps-réel [Dou99, Dou98, SR98], sont utilisés dans l'industrie. Les concepts de modélisation utilisés dans ces langages ont beaucoup évolué, mais les méthodes d'analyse employées dans les outils industriels ne couvrent pas les dernières avancées de la recherche. Du côté de la recherche, il y a beaucoup de modèles abstraits, tels que les automates temporisés [ACD93, AD94], les extensions temporisées des réseaux de Petri [MF76, Sif77, Ram74] ou encore les extensions des algèbres des processus [NS91], qui ont été développés en parallèle avec des méthodes d'analyse telles que la simulation, le model-checking, et la réécriture. Cependant, l'acceptation des modèles, des méthodes et des outils académiques par l'industrie se fait lentement, à cause de leur complexité et du support limité qu'ils offrent pour la conception des systèmes complexes.

En partant de ce constat, l'objectif de la thèse est d'intégrer dans le langage LDS les techniques de modélisation et d'analyse récemment développées dans le domaine des automates temporisés, et d'étendre les outils associés. Nous avons choisi LDS car c'est un langage largement répandu dans l'industrie temps-réel. Les autres avantages de LDS sont le fait qu'il soit standardisé, qu'il bénéficie d'une sémantique formelle, et que les outils de validation basés sur LDS sont plus proches de l'état de l'art dans le domaine. Le choix des automates temporisés comme base théorique pour les extensions temporelles de LDS se justifie par le fait que de nombreux travaux de recherche récents ont concerné ce modèle, et par conséquent beaucoup de problèmes théoriques (liées à la décidabilité du modèle, aux extensions possibles, aux problèmes de model-checking, etc.) ont été étudiés.

1.1.1 Contributions de la thèse

Les résultats de cette thèse peuvent être situés sur trois niveaux:

- Au niveau du langage LDS, avec des propositions d'extension et une sémantique du temps adaptée aux besoins de l'analyse temporisée,
- Au niveau des langages d'expression de propriétés, complémentaires à LDS,
- Au niveau des méthodes et des outils de simulation et de vérification.

Nous présentons les résultats plus en détail dans la suite de cette section.

Extensions du langage LDS

Nous commenons cette thèse avec une description de l'existant, et en particulier nous analysons la faon dont LDS couvre la description des informations temporelles agissant sur le comportement des systèmes modélisés. Nous identifions ainsi certaines lacunes dans la définition de LDS, vis-à-vis des aspects temporels mentionnés.

Ce problème de modélisation est abordé dans le Chapitre 6, où nous proposons une série d'extensions du langage, capables de représenter l'information descriptive sur le temps. Les extensions proposées permettent de décrire un comportement dépendant du temps, ainsi que les hypothèses temporelles sous lesquelles le système fonctionne, ceci directement dans LDS. Ces informations peuvent ensuite être utilisées par des outils d'analyse temporelle, tels que ceux développés dans le cadre de ce travail (présentés plus loin).

Les résultats que nous présentons ici ont été décrits dans nos papiers récents [BGK+00, BGM+01]. Avec un group de partenaires industriels et universitaires, nous sommes en train de raffiner et consolider l'ensemble des extensions de LDS pour le soumettre à l'ITU en vue de leur normalisation.

Sémantique du temps en LDS

La définition standard de LDS comporte une sémantique formelle [IT99c], qui fournit une correspondance entre l'ensemble des spécifications LDS et un ensemble d'objets mathématiques, ainsi qu'une interprétation mathématique de la notion d'exécution d'un système LDS. Dans la deuxième partie du Chapitre 6, nous présentons la sémantique des extensions du langage, ainsi qu'une sémantique appropriée de la notion de temps, dans le même formalisme ASM que [IT99c]. Cette sémantique précise la définition des extensions et fait la liaison avec les méthodes d'analyse utilisées ensuite sur des spécifications étendues.

Spécification des propriétés temporelles quantitatives

L'application des méthodes de validation choisies dans ce travail nécessite un formalisme de description des propriétés temporelles quantitatives. Nous avons considéré pour ce rôle deux langages, couramment utilisés avec LDS: GOAL et MSC.

GOAL [ALH95] est un langage d'observation défini par rapport à LDS et implémenté par l'outil *Object*GEODE [TEL00a]. Dans le Chapitre 7 nous nous intéressons à la description des propriétés temporelles quantitatives avec GOAL, et nous décrivons un ensemble d'extensions de GOAL à cet effet.

Dans le cas de MSC, certaines constructions pour exprimer des contraintes temporelles existent dans la dernière version du langage, MSC-2000 [IT99a], mais ces constructions n'ont pas encore une sémantique formelle. Nous proposons une sémantique temporisée pour un sous-ensemble de MSC-2000, basée sur les automates temporisés. Nous analysons également le problème de la satisfaction d'une propriété MSC par une spécification LDS.

Les deux langages mentionnés ci-dessus sont des langages orientés événement. Pour valider des propriétés décrites avec ces deux langages, il faut d'abord leur donner une base formelle; nous définissons donc un formalisme abstrait orienté événement, basé sur les automates temporisés, que nous appelons *automate de propriétés temporelles* (TPA). Dans le Chapitre 7 nous étudions également les problèmes de vérification (model-checking) posés par l'introduction des TPA.

Méthodes et outils de simulation et de vérification

La dernière partie du travail présenté dans cette thèse concerne les méthodes de simulation et de vérification de spécifications LDS étendues et des propriétés exprimées en GOAL et MSC. Cette partie a abouti à la réalisation d'un outil, dérivé d'un produit industriel (*Object*GEODE). Du point de vue théorique, la partie importante de l'outil est l'algorithme d'exploration symbolique de l'espace d'états d'un système LDS étendu (et des propriétés annexes). Nous définissons l'algorithme par les formules de calcul des successeurs d'un état symbolique. Un algorithme similaire à été présenté antérieurement dans [Boz99]; cependant, à notre connaissance, c'est la première fois que l'ensemble de formules de calcul des successeurs est caractérisé formellement et accompagné d'une preuve mathématique.

1.1.2 Organisation du document

La première partie présente l'état de l'art dans le domaine de la spécification et de la validation des systèmes temporisés. Nous décrivons le langage LDS (Chapitre 3), les langages MSC et GOAL (Chapitre 4), et le modèle des automates temporisés (Chapitre 5).

La deuxième partie présente les extensions temporisées des langages, les méthodes d'analyse et les outils développés dans le cadre de ce travail. Le Chapitre 6 présente les extensions apportées à LDS, et étudie leur impact sur la sémantique formelle de LDS. Le Chapitre 7 introduit le formalisme des TPA's, puis les extensions et la sémantique des langages GOAL et MSC. Le Chapitre 8 porte sur les méthodes d'analyse des spécifications LDS, MSC et GOAL, et sur le développement des outils support.

La partie finale du document contient une étude de cas (Chapitre 9) et les conclusions tirés de ce travail (Chapitre 10).

1.2 Présentation de l'existant

1.2.1 Spécification des systèmes temporisés en LDS

Ce travail débute par une étude de LDS, du point de vue des constructions du langage, et du point de vue de la sémantique. Cette étude s'appuie sur la version '2000 du langage [IT99b], et sur la sémantique formelle de LDS décrite en ASM [IT99c]. L'étude réalisée couvre la majorité des concepts de LDS-2000 (relatifs à l'architecture, à la communication, à la description du comportement, et aux données), ainsi que le formalisme ASM et la sémantique statique et dynamique du langage. Dans la suite du paragraphe, nous allons présenter les conclusions de cette étude concernant l'expression du comportement *temporisé*, et la sémantique du temps, qui sont essentielles pour le reste du travail réalisé dans cette thèse.

Description du comportement temporisé en LDS-2000

LDS définit des constructions qui permettent la description du comportement temporisé. Il existe deux types de données relatifs au temps dans LDS: *Time* et *Duration*. Les valeurs du type *Time* représentent des moments sur une échelle de temps depuis l'initialisation du système, tandis que les valeurs du type *Duration* représentent des distances relatives (différences) entre moments sur l'échelle absolue. Des opérateurs spécifiques sur les valeurs de ces types (addition de temps et de durées, multiplication de durées, etc.) sont prédéfinis dans le langage.

Le temps présent (i.e. écoulé depuis l'initialisation du système) est consultable par l'intermédiaire de l'opérateur prédéfini **now**. La manière dont le temps s'écoule n'est pas définie dans LDS: la seule hypothèse qui est faite sur les valeurs de **now**, est que leur évaluation successive donne toujours des valeurs croissantes.

Ainsi le comportement dépendant du temps peut être décrit des deux manières: soit en utilisant la valeur de **now** dans des tests ou des expressions de tirage de transitions, soit en utilisant des *temporisations*.

Les *temporisations* sont des objets spéciaux du langage LDS, qui ont des attributs spécifiques comme les données (e.g. état d'activité), mais aussi un comportement prédéfini (indépendant du comportement des agents du système LDS). Une *temporisation* peut être définie par un agent LDS (avec le mot clé **timer**); l'agent peut ensuite armer la temporisation avec une date d'échéance (par l'opération **set**), la désarmer (par l'opération **reset**), ou consulter son état (par l'opération **active**). Le comportement d'une temporisation est le suivant: la temporisation est inactive tant qu'elle n'a pas été armée. Une fois que la temporisation est armée, elle devient active et attend l'arrivée de son échéance. Quand l'échéance arrive, la temporisation expire et un *signal est déposé dans la file d'attente* de son agent propriétaire. Si la temporisation est

désarmée avant que l'échéance n'arrive ou avant que le signal correspondant ne soit consommé, elle redevient inactive et tout signal correspondant est effacé de la file d'attente.

Il est important de noter le fait que l'expiration d'une temporisation produit un signal asynchrone, qui passe toujours par la file d'attente de son agent propriétaire. Par conséquent, quand un signal issu d'une temporisation est consommé, la seule hypothèse garantie par la sémantique de LDS est que l'échéance de la temporisation a eu lieu. En principe, on ne peut rien supposer à propos du temps passé depuis l'échéance. Cette hypothèse minimale est correcte du point de vue des implémentations d'un système LDS, mais présente un certain nombre d'inconvénients quand la spécification LDS est utilisée à des fins de simulation ou de vérification.

Description vs. spécification

Comme décrit dans l'introduction de la norme Z.100 [IT99b], LDS vise à la fois la *spécification* de haut niveau, et la programmation (*description*) des systèmes. Les deux objectifs du langage sont parfois conflictuels, et le côté programmation a été prioritaire dans sa définition. Pour cette raison, LDS est un langage de conception assez complet, mais il manque certaines constructions pour la modélisation de haut niveau, nécessaires dans les phases initiales de spécification d'un système. Certaines des extensions proposées dans cette thèse permettent donc la spécification abstraite:

- du comportement des canaux, avec des attributs tels que le taux de perte, les délais minimaux/maximaux, etc.
- des temps d'exécution,
- du comportement (temporisé) de l'environnement du système.

Les constructions proposées seront présentées plus en détail dans la suite du résumé.

Sémantique et raisonnement sur le temps

La sémantique du temps dans LDS est présentée en termes formels dans la thèse. Nous dressons ici un résumé des caractéristiques principales de cette sémantique:

- Les actions individuelles LDS (affectation, envoi de signal, etc.) sont atomiques et prennent un temps nul pour s'exécuter. La granularité de l'atomicité des actions composées et des transitions est l'action individuelle.
- L'évaluation des expressions n'est pas atomique. Par conséquent, la valeur de **now** peut varier pendant l'évaluation d'une expression, ce qui peut influencer sur le résultat de l'expression.
- En général, une quantité non-déterminée de temps peut s'écouler entre l'exécution de deux actions ou de deux transitions
- Une conséquence directe du point antérieur est qu'un message correspondant à une *temporisation* peut ne pas être pris en compte pendant une certaine durée (non-déterminée) après son envoi.

Avec ces hypothèses minimalistes, il est difficile de garantir une propriété sur le comportement d'un système dans le temps. D'autre part, beaucoup de comportements peu réalistes sont

considérés comme acceptables par la sémantique. Ce problème a déjà été signalé par d'autres auteurs [Boz99, MGHS96] et il est examiné en détail dans nos travaux récents [BGK+00, BGM+01].

Le problème signalé ici pose de sérieuses difficultés aux outils de simulation et de vérification basés sur LDS. Une solution souvent employée par les outils est de considérer une sémantique du temps complètement différente de celle de la norme, basée sur des suppositions réductrices, telles que: chaque action prend un temps nul, le temps est contrôlé et s'écoule seulement quand le système n'a rien à exécuter, etc. Cette solution tombe dans l'autre extrême, et peut cacher des scénarios d'exécution réalistes du système.

En complément des extensions du LDS présentées auparavant, nous proposons aussi une sémantique alternative du temps, qui résout les problèmes mentionnés ci-dessus. L'idée de cette sémantique est d'inclure des informations sur le progrès du temps, inspirées du modèle des automates temporisés, dans la spécification LDS, et de les utiliser ensuite pour contrôler le progrès du temps dans la simulation ou la vérification.

1.2.2 Spécification de propriétés en MSC et GOAL

Un aspect central de la technique de validation basée sur les modèles considérée dans cette thèse est la spécification des contraintes et des propriétés sur le modèle LDS. Nous avons choisi deux langages couramment utilisés avec LDS pour la spécification de propriétés: MSC et GOAL. Ces deux langages sont décrits en détail dans le mémoire; nous faisons ici une brève présentation des deux langages, et donnons des conclusions concernant leur utilisation pour la spécification de contraintes temporelles.

MSC

MSC est un langage normalisé par l'ITU (norme Z.120, [IT99a]), utilisé pour représenter des traces d'exécution des systèmes distribués en termes de messages échangés entre les entités du système ou avec l'environnement. Les composantes principales d'une spécification MSC sont les *instances* (représentant des entités ou des groupes d'entités d'un système) et les *messages* (qui peuvent représenter diverses modalités de communication, dépendant du système considéré). D'autres types d'évènements peuvent être spécifiés dans les traces MSC, tels que des évènements concernant les temporisations (**set**, **reset**, **timeout**), des actions ou des conditions (informelles). Le langage a aussi des constructions pour structurer (composer) les spécifications. Les types de composition possibles sont: l'alternative entre plusieurs MSC, la composition parallèle (par entrelacement d'évènements) de plusieurs MSC, la répétition ou l'exécution optionnelle d'une MSC.

La dernière version du langage (MSC-2000) propose plusieurs constructions pour exprimer des conditions sur le temps. On peut essentiellement exprimer des *contraintes relatives*, qui spécifient la durée passée entre deux évènements, et des *contraintes absolues* qui spécifient le moment auquel un évènement peut survenir. Les deux types de contraintes sont spécifiés au moyen d'une limite inférieure et d'une limite supérieure, qui sont soit des valeurs constantes de temps, soit des expressions plus complexes du type **Time**. Dans le deuxième cas, les expressions peuvent porter sur des résultats de *mesures de temps*. On peut mesurer en effet, par des constructions spécifiques de MSC-2000, soit le temps auquel un évènement survient soit le délai relatif entre deux évènements.

Dans notre travail, nous avons jugé suffisantes les constructions proposées dans MSC-2000 pour exprimer des contraintes temporelles. Il existe néanmoins plusieurs raisons pour lesquelles

la notation MSC-2000 ne peut, en l'état actuel, être utilisée pour la spécification et la vérification des propriétés temporelles des systèmes LDS:

- La sémantique de MSC-2000 n'est pas formellement définie dans la norme.
- Il n'y a pas de relation de conformité formellement définie entre des spécifications LDS et des spécifications MSC. Cette relation a été juge en dehors de l'objet de la norme Z.120.
- Certaines des constructions de MSC-2000 (notamment les mesures de temps) sont trop expressives et il n'y a pas de méthode de vérification qui puisse les prendre en compte.

Dans la thèse, nous nous sommes intéressés à ces problèmes, et nous avons proposé des solutions qui sont présentées plus loin.

GOAL

GOAL [ALH95] est un langage d'observation supporté par l'outil *Object*GEODE [TEL00a]. Pour plus de détail sur les langages d'observation le lecteur peut consulter les travaux de [Gro89].

Par définition, GOAL a un domaine d'applicabilité plus limité que les MSC, n'étant pas un langage de haut niveau pour spécifier des propriétés abstraites. Le langage est utilisé pour exprimer et vérifier des propriétés comportementales d'un système LDS, et pour guider le processus de simulation et de vérification.

GOAL est un langage orienté évènements, et basé sur des automates. La spécification d'un observateur GOAL ressemble à une machine à états d'un agent LDS. Les transitions de l'observateur sont tirées par des évènements se produisant dans la spécification LDS associée, qui peuvent être des échanges de messages, la création ou l'arrêt des agents, le tir de certaines transitions, etc. Les états d'un observateur peuvent être de trois types: *succès*, *échec* ou *ordinaire*, et correspondent à la satisfaction ou à la non-satisfaction de la propriété spécifiée par l'observateur.

Dans la thèse, nous avons mis en évidence deux problèmes relatifs à l'utilisation de GOAL comme langage de représentation/validation de propriétés temporelles quantitatives:

- L'absence de constructions permettant l'expression des conditions sur le temps.
- L'absence d'une sémantique temporisée.

Les solutions proposées dans la thèse pour résoudre ces problèmes sont présentées plus loin.

1.2.3 Spécification et vérification avec des automates temporisés

Pour compenser les points faibles de LDS, MSC et GOAL, en termes de sémantique temporisée et de méthodes d'analyse, nous nous intéressons à l'application des techniques d'automates temporisés en conjonction avec ces langages.

Le modèle des automates temporisés est un modèle de machines à états étendu avec des constructions pour la spécification des contraintes temporelles. Les éléments essentiels d'un automate temporisé sont des états discrets, des transitions, et des horloges qui mesurent le temps. Pour faciliter la spécification des contraintes complexes, un automate peut utiliser plusieurs horloges qui avancent toutes à la même vitesse, mais qui peuvent être remises à zéro ou consultées séparément.

Le comportement dépendant du temps est spécifié en imposant aux transitions des gardes qui portent sur les valeurs d'horloges. Le modèle limite les formes acceptées dans ces gardes,

pour préserver la décidabilité du modèle: seules les comparaisons d'horloges avec des constantes (entières) ou les comparaisons de différences de deux horloges avec des constantes sont valides.

La sémantique d'un automate temporisé est donnée par un graphe sémantique en temps continu: l'état dynamique d'un automate comprend un état discret (partie de la spécification de l'automate), et une valeur réelle pour chaque horloge du système. Dynamiquement, un automate peut exécuter deux types de transitions: des transitions discrètes (voir la spécification d'un automate), et des transitions temporelles qui signifient le progrès du temps. Les transitions temporelles ne servent qu'à avancer le temps, i.e. à augmenter (uniformément) la valeur de toutes les horloges. En revanche le temps ne progresse pas durant les transitions discrètes, i.e. la valeur de chaque horloge reste la même ou est remise à zéro (dans le cas d'un reset spécifié sur la transition).

Une exécution d'un automate temporisé est donc une succession (finie ou infinie) de transitions temporelles et de transitions discrètes en alternance. Pour pouvoir modéliser des actions (transitions) qui arrivent à un moment précis, le progrès du temps durant l'exécution est par définition relié à l'exécution de l'automate, au moyen des *conditions de progrès du temps*. Ces conditions dépendent de la valeur d'un attribut de chaque transition de l'automate, appelé *urgence* . L'urgence de chaque transition peut avoir les valeurs suivantes: *eager* , *delayable*, *lazy*. Brièvement, la signification des valeurs d'urgence est la suivante:

- Dès qu'une transition *eager* est tirable, le temps ne peut pas progresser. La transition eager ou toute autre transition discrète tirable doit alors être tirée.
- Quand une transition *delayable* est tirable, elle empêche le temps de progresser au-delà de la borne supérieure de sa garde. Les conditions de progrès du temps sont composées, de faon que la condition la plus restrictive s'applique.
- Les transitions *lazy* n'imposent aucune condition sur le progrès du temps.

Méthodes d'analyse et problèmes décidables

Plusieurs problèmes importants pour la validation du comportement sont décidables sur le modèle d'automates temporisés, et des méthodes d'analyse efficaces sont disponibles. On peut par exemple décider de l'atteignabilité d'un état de l'automate, ce qui implique la décidabilité de la vérification des diverses propriétés d'invariance ou de sûreté. Les problèmes de satisfaction des propriétés spécifiées dans diverses extensions de logiques temporelles (ou dans d'autres formalismes tels que les automates temporisés avec des conditions d'acceptation de Büchi) sont aussi décidables, et il existe des méthodes de vérification concrètes pour ces problèmes.

L'analyse des automates temporisés utilise des abstractions, notamment des représentations symboliques du graphe sémantique d'un automate, pour pallier au fait que le graphe sémantique est habituellement infini et non-dénombrable. Dans le mémoire de thèse, nous présentons deux abstractions, le *graphe de régions* et le *graphe de simulation* qui seront ensuite appliquées à l'analyse du langage LDS étendu.

Le modèle des automates temporisés que nous avons choisi pour notre travail impose des restrictions de modélisation, cependant il donne une limite supérieure de complexité des modèles temporisés analysables, dans le sens où plusieurs extensions de ce modèle ont été étudiées avec des résultats essentiellement négatifs concernant la décidabilité et les méthodes d'analyse applicables (voir dans le mémoire de thèse). Pour cette raison, nous considérons que les restrictions du modèle, qui vont se refléter plus tard au niveau du langage LDS étendu, sont inévitables pour préserver l'analysabilité des spécifications LDS.

Idées sur les extensions temporelles et la sémantique de LDS

Comparé avec LDS, les automates temporisés apportent plusieurs idées qui facilitent la spécification du comportement temporisé et le raisonnement temporisé basé sur le modèle:

- Les horloges et les gardes donnent un moyen flexible d'exprimer des contraintes temporelles complexes. Elles peuvent être introduites comme un complément aux constructions existantes de LDS (**now**, temporisations).
- Les *conditions de progrès du temps* limitent les comportements possibles d'un modèle, et permettent de spécifier quels sont les comportements raisonnables du point de vue temporel. Les urgences peuvent être utilisées pour spécifier des actions qui sont exécutées à un moment précis ou dans un intervalle précis de temps, ce qui n'est pas possible en LDS standard.
- Les conditions sur le temps (i.e. sur les valeurs d'horloges) ont des formes restreintes, ce qui permet l'analyse et la vérification automatique des propriétés. En revanche, en LDS standard la complexité des conditions sur **now** n'est pas contrainte, et il n'existe pas de méthodes d'analyse applicables dans le cas général.

1.3 Extensions des langages et méthodes de validation

1.3.1 Extensions de LDS

Notre travail sur l'extension de LDS comporte principalement deux parties: les extensions des constructions du langage pour décrire des informations temporelles, et l'extension /modification de la sémantique formelle.

Extensions pour la représentation des informations temporelles

Les extensions que nous proposons pour LDS sont en partie des constructions inspirées directement du modèle des automates temporisés, et en partie des extensions de plus haut niveau pour la spécification des informations tels que le temps d'exécution des actions, le temps de transmission des signaux, etc.

Horloges, gardes, urgences. Nous proposons une définition des horloges en tant que mécanisme de base pour mesurer et contraindre la progression du temps. Techniquement, les horloges sont introduites en LDS par l'intermédiaire d'un type de données (`Clock`). Les opérations habituelles (création, reset, comparaison avec un entier, différence de deux `Clocks`) sont définies sur les valeurs de ce type.

Les comparaisons de `Clocks` ou de différences de deux `Clocks` avec un entier peuvent être utilisées dans la spécification de la garde d'un transition LDS (la notion de garde existe déjà dans LDS, avec le mot clé **provided**). De plus, cette extension de LDS permet la spécification d'une *urgence* (*eager*, *delayable* ou *lazy*) pour chaque transition.

Durées d'exécution des actions. Cette extension permet la spécification de durées d'exécution des actions au moyen d'une borne inférieure et d'une borne supérieure. Comme dans le modèle des automates temporisés, les actions LDS s'exécutent dans un temps nul, mais les actions qui prennent du temps sont simulées par un état implicite symbolisant l'action en cours d'exécution, et par une transition *delayable* symbolisant la fin de l'exécution.

La spécification des canaux. Les canaux LDS standards ne perdent jamais de signaux, et les délais appliqués aux signaux transférés sont soit nuls, soit non-spécifiés. Pour valider le comportement d'un spécification LDS standard avec des hypothèses précises sur le comportement des canaux, l'utilisateur doit modifier le modèle LDS et fournir une description impérative des canaux comme agents, avec tous les inconvénients inhérents, qui sont présentés dans la thèse. De plus, à cause de la sémantique non-contrainte du temps dans LDS, le comportement précis des canaux ne peut être garanti.

L'extension proposée dans la thèse permet de spécifier un taux de perte et des bornes minimales et maximales pour les délais appliqués aux signaux transmis sur un canal. On définit deux types de délais pour les canaux: cumulatifs et non-cumulatifs. Dans le cas de délais cumulatifs, les temps d'arrivée des signaux qui précèdent un signal sont rajoutés au temps d'arrivée du signal concerné. Dans le cas de délais non-cumulatifs, le temps d'arrivée d'un signal est compris strictement entre les bornes spécifiées sur le canal, et il est contraint par les signaux précédents seulement par le fait que les canaux sont FIFO. Les canaux non-cumulatifs correspondent aux liaisons qui font un traitement en parallèle (ou en chane) des signaux, tandis que les canaux cumulatifs correspondent aux liaisons où les signaux sont transmis un par un.

Le mémoire de thèse illustre ces concepts sur un exemple réel, le protocole SpaceWire [sWG00] développé par l'Agence Spatiale Européenne, spécifié en LDS avec les extensions décrites ci-dessus.

Sémantique des extensions et du temps

La sémantique des extensions et une nouvelle sémantique du temps sont décrites dans la thèse, en utilisant le formalisme ASM. Les définitions introduites complètent la sémantique standard de LDS [IT99c]. Nous ne reprenons pas ici ces définition, mais nous soulignons les points difficiles et les choix qui ont été faits.

La sémantique des canaux et des temporisations dans LDS (standard) utilise le concept de *schedule*, qui sert pour retarder l'arrivée d'un signal (e.g. un signal correspondant à une temporisation) à son agent de destination. Nous avons étudié deux faons de traiter les temporisations et les canaux à délai (borné) dans LDS étendu: soit en utilisant les *schedules* , soit en utilisant des horloges implicites. La deuxième alternative s'avère préférable, car elle ne s'appuie pas sur une notion de temps absolu (comme c'est le cas dans les *schedules*) mais sur des mesures relatives, ce qui permet l'application des techniques d'analyse d'automates temporisés.

Un point important de la sémantique du temps que nous proposons est la contrôlabilité. En effet, le temps dans la sémantique standard de LDS est considéré comme un paramètre extérieur au système. Cela se reflète dans le fait que **now** est une fonction dite **monitored** dans la sémantique ASM standard. Pour pouvoir introduire des *conditions de progrès du temps* comme dans les automates temporisés, le temps doit être un paramètre contrôlé par le système en fonction des transitions tirables et de leur urgence.

Modifier de cette faon le statut du temps implique des nombreuses transformations dans la sémantique associée. Nous avons introduit un nouvel agent responsable de l'avancement du temps et des valeurs d'horloges, et nous avons décrit les conditions de progrès du temps en ASM, en fonction de l'état des tous les agents LDS et de l'état des canaux à délai. Par le fait qu'une condition de progrès du temps est une condition globale qui porte sur l'état des toutes les composantes d'un système, nous avons été obligés d'introduire des synchronisations supplémentaires entre les agents ASM définis par la sémantique, qui sinon sont entièrement asynchrones.

Correspondance avec les automates temporisés

Les outils de simulation et de vérification existants basés sur LDS n'utilisent pas la sémantique ASM, mais construisent directement un graphe d'états global du système en utilisant des simplifications pour des raisons d'efficacité. Cette notion de graphe d'états est en fait assez proche du modèle sémantique des automates temporisés, et nous pouvons l'étendre pour y inclure les extensions proposés pour LDS (notamment les horloges). Le résultat est un graphe d'états en temps continu qui ressemble beaucoup à celui des automates temporisés, et sur lequel nous pouvons appliquer des techniques d'analyse spécifiques. Les composantes – états et types de transitions – de ce graphe sont décrites dans le Chapitre 6 du mémoire.

1.3.2 Description et vérification des propriétés temporisées avec MSC et GOAL

Nous nous sommes intéressés aux problèmes décrits dans la section 1.2.2, qui empêchent l'utilisation de MSC et de GOAL en tant que langages de propriétés temporisées pour les systèmes LDS. Les travaux que nous avons réalisés concernent trois niveaux: les constructions introduites dans les langages, leur sémantique, et les méthodes de vérification automatique.

Automates de Propriétés Temporisés

Pour donner une base sémantique solide aux deux langages, nous avons défini un modèle abstrait de description de propriétés, calqué sur les automates temporisés, que nous appelons Automates de Propriétés Temporisés (TPA).

Un TPA est un automate temporisé équipé d'une condition d'acceptation de type Büchi. La différence entre les TPA et les variantes d'automates temporisés de Büchi (TBA) proposés dans [Alu91, Tri98] est que le modèle des TPA est orienté événement et non pas orienté état. Cela signifie qu'une propriété TPA porte sur les évènements qui on lieu dans le modèle associé, et non pas sur les états du modèle. Cette différence est importante, dans la mesure où MSC et GOAL sont tous les deux des langages orientés évènement.

Dans le mémoire nous décrivons formellement le modèle des TPA, et la relation de satisfaction entre un automate temporisé et un TPA. Nous étudions aussi le problème de la vérification de la satisfaction, et nous proposons un algorithme basé sur l'utilisation du *graphe de simulation* des automates temporisés. Cette méthode d'analyse est utilisée ensuite pour vérifier des propriétés MSC et GOAL sur des systèmes LDS étendus.

MSC

Au niveau des MSC, les deux problèmes principaux sont l'absence d'une sémantique qui prenne en compte les aspects temporels du langage, et l'absence d'une relation de satisfaction entre des spécifications LDS et des propriétés MSC.

Nous proposons une sémantique temporisée basée sur les automates temporisés, en partant de la sémantique non-temporisée basée sur des réseaux de Petri proposée dans [GPR93] et en l'étendant avec des horloges et des contraintes temporelles. Nous arrivons ainsi à traiter la plupart des contraintes exprimables en MSC-2000. Les parties du langage pour lesquelles nous ne pouvons pas donner une sémantique concernent notamment les *mesures de temps*, et l'utilisation des variables ou paramètres de type **Time**.

La composition séquentielle des MSC pose aussi des problèmes de décidabilité, déjà signalés par d'autres auteurs [MP00]. Pour pouvoir utiliser les MSC dans la vérification de propriétés, nous avons restreint la définition de la composition séquentielle, de faon à ce que le langage de traces généré par un MSC composite soit toujours régulier.

Nous proposons aussi des définitions possibles pour la relation de satisfaction entre des spécifications LDS et des propriétés MSC. En interprétant l'automate temporisé qui donne la sémantique d'un MSC comme un TPA, nous avons trouvé des correspondances entre la satisfaction des MSC et la satisfaction des TPA. Par conséquent, nous pouvons donner une méthode concrète de vérification pour les propriétés MSC, en utilisant les mêmes techniques que dans le cas des TPA.

Des travaux effectués par d'autres auteurs visent aussi à utiliser les MSC en tant que langage des propriétés, pour la vérification formelle. On notera principalement les travaux sur les Live Sequence Charts (LSC, [DH98]), mais aussi les approches proposés par des outils industriels tels que *Object*GEODE [TEL00a]. Cependant, aucun des travaux sur le sujet ne traite à notre connaissance de la partie concernant le temps.

GOAL

Dans le cas de GOAL, la notion de satisfaction et la sémantique du langage sont déjà définies dans l'outil *Object*GEODE. Le langage manque cependant de constructions pour exprimer des contraintes sur le temps, et sa sémantique doit être adaptée à la nouvelle sémantique temporisée de LDS définie dans cette thèse.

Comme constructions temporelles, nous avons proposé des concepts directement inspirés des TPA: des horloges et de gardes. La sémantique de GOAL est relativement facile à définir en termes de TPAs, du fait qu'il y a une relation directe entre les concepts des deux modèles. Même si les extensions de GOAL sont très légères, les études de cas que nous avons effectuées montrent que le langage résultant est très flexible et permet la spécification de propriétés linéaires complexes.

1.3.3 Simulation et vérification temporelles de LDS

Un des objectifs des extensions décrites dans les sections précédentes est de pouvoir valider le comportement temporel des systèmes LDS, par simulation ou par vérification de propriétés. Dans cette section nous décrivons un outil que nous avons développé à cette fin, qui se présente comme une extension de l'outil de simulation et de vérification de *Object*GEODE [TEL00a], dont il réutilise l'architecture globale et les fonctionnalités principales. L'avantage de réutiliser un environnement industriel est que l'implémentation des constructions des langages qui ne sont pas affectées par les extensions temporelles, est obtenue sans effort supplémentaire.

Fonctionnalités et architecture de l'outil

L'outil offre des fonctionnalités pour:

- La *simulation interactive ou aléatoire*. Les fonctionnalités offertes dans ce mode de fonctionnement ressemblent à celles des débuggeurs pour les langages de programmation: exécution pas à pas, conditions d'arrêt, inspection des données. Il existe aussi d'autres fonctions spécifiques: exécution inversée, sauvegarde des scénarios d'exécution, stimulation automatique des modèles ouverts, production des traces sous forme de MSC, analyse de couverture du modèle, etc.

– La *vérification par exploration exhaustive* de l'espace d'états du modèle. L'outil peut vérifier: l'absence de blocages, l'invariance de certaines conditions logiques, l'absence de certaines erreurs dynamiques (e.g. signaux non-attendus), et la satisfaction de propriétés écrites en MSC ou GOAL.

Les deux modes d'utilisation sont basés sur la construction de l'espace d'états du modèle, mais le processus de construction et la taille de l'espace sont différents dans chaque cas. La vérification d'un modèle est toujours effectuée à la volée, et par conséquent l'espace d'états n'est entièrement construit que dans certains cas.

L'outil est formé de deux modules principaux: un *compilateur* des modèles, et une *librairie* englobant les fonctionnalités génériques des simulateurs. Le compilateur prend en entrée un modèle LDS et une ou plusieurs propriétés MSC ou GOAL; il les transforme dans un format exécutable, où les transitions LDS, par exemple, deviennent des routines utilisant des primitives qui implémentent les types d'actions définis dans LDS. Ces primitives font partie de la *librairie*, qui englobe aussi des structures de données standard, et des fonctionnalités génériques (parcours de l'espace d'états, configuration du modèle, etc.).

Au final, le compilateur génère un simulateur (sous forme d'un exécutable séparé) pour chaque modèle LDS. Le simulateur construit l'espace d'états du modèle et implémente toutes les fonctionnalités de simulation et de vérification décrites auparavant.

La construction du graphe de simulation temporisé

L'espace d'états construit par l'outil est une abstraction de l'espace d'états en temps continu défini par la sémantique de LDS. Les états manipulés par le simulateur sont des états symboliques (q, S), où q est un état discret global du modèle (n'incluant aucune information sur le temps et les horloges). S est une *zone* de valeurs d'horloges atteignables dans l'état q, qui a la forme d'un polyèdre (éventuellement non-convexe et non-borné) dans l'espace des valeurs d'horloges ($\mathbb{R}^n$, où n est le nombre d'horloges actives dans q).

Les transitions de ce graphe de simulation correspondent uniquement aux transitions discrètes définies par la sémantique de LDS étendu, qui sont soit des transitions LDS *explicites*, soit des transitions *implicites* (expiration de temporisations, arrivée de signaux retardés sur les canaux). Pour plus de détail, le lecteur peut consulter la sémantique détaillée dans le mémoire. Le calcul des successeurs d'un état (q, S) après l'exécution d'une transition e se fait en deux étapes. On calcule d'abord les états directement atteignables en exécutant la transition e sur chaque état explicite contenu dans l'état symbolique (q, S). On obtient ainsi un autre état symbolique (q', S'). A partir de ce dernier, on calcule combien de temps on peut rester dans chaque état explicite contenu dans (q', S'), et on obtient ainsi l'état symbolique de destination (q', S'').

Les deux étapes décrites ci-dessus sont appelées respectivement le *calcul des successeurs discrets* et le *calcul des successeurs temporels*. Le calcul des successeurs est plus compliqué dans le cas où des propriétés GOAL ou MSC sont associées au modèle LDS, car il faut tenir compte de l'état des automates représentant ces propriétés. En particulier, il peut arriver qu'un état ait plusieurs successeurs différents par la même transition e, du fait que des conditions différentes contenues dans la propriété soient satisfaites par des parties différentes d'un état symbolique.

L'algorithme de calcul des successeurs est présenté en détail dans le mémoire. Il s'appuie sur une représentation de données spécifique (plus particulièrement sur la représentation des polyèdres S par des matrices de différences bornées – DBM [Dil89, ACD93]) et sur des formules de calcul des successeurs temporels qui réalisent des opérations élémentaires sur des polyèdres.

Une partie essentielle du travail effectué est la preuve de la validité de ces formules, présentée dans l'Annexe B.

1.4 Application et conclusion

Nous avons validé les concepts et les outils développés dans le cadre de ce travail sur un ensemble d'études de cas, incluant des cas d'école: un système de barrière de voie ferrée (utilisé précédemment dans [Alu91] et [Tri98]), un protocole de contrôle de flot (BRP, étudié aussi dans [GvdP96, HS96, Mat96, DKRT97]). Ces études de cas ont donné des bons résultats quant à l'expressivité des extensions et à la puissance des méthodes d'analyse utilisées.

Nous avons considéré aussi des études de cas plus complexes. Dans la thèse, nous présentons l'exemple d'un protocole de liaison de données (SpaceWire, [sWG00]) ; l'approche est aussi expérimentée dans le cadre d'autres projets R&D en cours, sur le protocole de multicast RMTP-2 [PMR+00, WPT99] et sur un protocole de synchronisation de flots multimédias.Ces exemples ont montré tout le bénéfice de l'approche retenue, mais aussi certaines limites des extensions que nous proposons, telles que l'impossibilité d'utiliser des mesures de temps dans des systèmes adaptatifs (e.g. contraintes de temps variables en fonction de l'évolution du système).

L'exemple présenté dans le Chapitre 9 du mémoire montre en détail la modélisation des contraintes temporelles contenues dans la norme SpaceWire avec les extensions LDS que nous avons proposées. La validation du modèle du point de vue temporel est aussi discutée, et nous montrons comment la simulation peut être exploitée pour faire des mesures de temps globales qui seront ensuite utilisées pour écrire et vérifier des propriétés en MSC et GOAL.

En conclusion, nous avons développé un ensemble d'extensions, des méthodes d'analyse et un outil support, permettant la description et la validation des systèmes temps réel avec des contraintes temporelles complexes, exprimables dans le langage LDS et les langages connexes MSC et GOAL. Les études de cas réalisées montrent que les extensions proposées sont flexibles et intuitives, et que l'expression des propriétés pour la vérification est aisée, en comparaison avec des langages mathématiques tels que les logiques temporelles. Les méthodes d'analyse développées permettent la dérivation des informations temporelles pertinentes telles que les délais minimaux/maximaux entre évènements.

Dans une perspective future, ce travail peut être continué par la recherche d'un ensemble de concepts de plus haut niveau à intégrer dans les langages de modélisation, éventuellement basés sur les concepts sémantiques proposés ici, qui peuvent apparatre de trop bas niveau et compliquer la conception des modèles. D'autres axes de recherche concernent l'amélioration des techniques de vérification (e.g. par application des méthodes de réduction de l'espace d'états), l'application des nouvelles méthodes de validation (e.g. génération automatique de tests) ou l'intégration dans de nouveaux langages tels que la notation UML.