REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEINEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS DE SETIF
UFAS, ALGERIE

MEMOIRE

Présenté à la faculté des sciences
Département D'informatique

Pour l'obtention du diplôme de

MAGISTER

Option : Ingénierie Des Systèmes Informatique

Présenté par : Lemya ACHI

Analyse de l'ordonnançabilité sur des modèles UML temps-réel

Soutenu le : 28/06/2012 **Devant la commission d'examen :**

Mr. A. Boukerram	**Maitre de conférences**		**Président**
Mr. M. MOSTEFAI	**Professeur**	**U.F.A SETIF**	**Rapporteur**
Mr. M. Alliouat	**Maitre de conférences**		**Examinateur**

Résumé

L'utilisation des outils actuels de vérification et de simulation de l'ordonnançabilité des systèmes permet la validation de l'ordonnancement des systèmes. L'étude de l'analyse d'ordonnançabilité est judicieuse surtout quand les systèmes ne tolèrent pas de fautes. La non prise en compte des aspects de l'analyse d'ordonnançabilité dans les étapes précoces du cycle de développement (la conception) de ces systèmes, rend la mise en cause du modèle plus difficile et onéreuse surtout quand l'ordonnancement est irréalisable. Grâce aux mécanismes d'extension d'UML, le profil MARTE permet de prendre en compte les aspects particuliers des systèmes temps réel et embarqué, ainsi les modèles UML peuvent être annotés par des informations supplémentaires ce qui rend la modification de la modélisation de départ faisable suite aux résultats aboutis par les outils de vérification. Le but de ce travail consiste à fournir un framework basé MARTE permettant aux concepteurs des systèmes Temps-réel de modéliser leurs systèmes afin d'analyser leurs ordonnançabilités grâce à l'outil d'analyse MAST.

Mots clés : Systèmes temps-réel et embarqués, MARTE, Analyse de l'ordonnançabilité temps-réel

---

ملخص

استخدام الأدوات الحالية للتحقق ولمحاكاة قابلية جدولة الأنظمة يسمح بإثبات صحة تقنيات جدولة الأنظمة. تعتبر دراسة تحليل قابلية جدولة مهمة خصوصا إذا تعلق الأمر بصنف الأنظمة التي لا تتسامح مع الأخطاء.عدم الأخذ بعين الاعتبار جوانب تحليل قابلية جدولة الأنظمة في المراحل المبكرة من دورة الإنشاء (التصميم) لهذه الأنظمة يجعل تصحيح النموذج أكثر صعوبة وتكلفة وخصوصا عندما تكون تقنية الجدولة غير ممكن عمليا. مع التوسع في جانب UML الذي يسمى MARTE أصبح بالإمكان الأخذ بعين الاعتبار الجوانب الفريدة الخاصة بأنظمة الوقت الحقيقي والمدمجة، ونماذج UML يمكن إثراؤها بمعلومات إضافية مما يجعل تغيير النموذج الأولي ممكنا وذلك بحسب النتائج التي تستخلصها وسائل التحقق. الهدف من هذا العمل هو تزويد هيكل مبني على MARTE، الهيكل يسمح لمصممي أنظمة الوقت الحقيقي بتحقيق نماذج لأنظمتهم من أجل تحليل قابليتها للجدولة باستخدام وسيلة التحليل MAST.

الكلمات مفتاحية : أنظمة الوقت الحقيقي والمدمجة ، MARTE، تحليل قابلية جدولة نظم الوقت الحقيقي

Abstract

The use of current tools for schedulability verification and simulation allows the validating of systems scheduling. The study of the schedulability analysis is appropriate especially when the systems do not tolerate mistakes. Missing the aspects of schedulability analysis in the early stages of the development cycle (design) of these systems makes the implication of the model more difficult especially when the scheduling politic was unfeasible. With UML profile MARTE, it is possible to specify the especial aspects of real-time and embedded systems, thus UML models can be annotated with additional information which permit to change the starting model according to the results have been achieved by the verification tools. The aim of this work is to provide designers of Real-Time systems with a framework based on MARTE which allows them to design their systems in order to analyze their schedulability using the analysis tool MAST.

Key words: Real-time and embeded systems, MARTE, Real time schedulability analysis

Remerciements

Je remercie très sincèrement le professeur M.Mostefai -Vice-recteur de la formation supérieure de post-graduation, de l'habilitation universitaire et de la recherche scientifique à l'U.F.A Sétif- d'avoir bien accepté d'encadrer ce travail, ainsi pour sa sympathie, et sa disponibilité malgré ses charges.

J'adresse mes remerciements très distinguées à Mr A.Bouamari –Chef de département Informatique à l'U.F.A Sétif- qui s'est toujours montré à l'écoute et très disponible, ainsi pour l'inspiration, l'orientation, l'aide et le temps qu'il a bien voulu me consacrer.

Je remercie Docteur A.Boukerram d'avoir accepté d'examiner ce travail et d'en avoir assuré la présidence.
Je remercie Docteur M.Alliouat d'avoir accepté d'examiner ce travail.

Je tiens à exprimer ma gratitude et mes remerciements très particuliers à Mr A.Beddiaf pour sa collaboration très importante à ce travail, je suis très reconnaissante envers lui pour son immense soutien d'un très bon ami et ses gros encouragements.

Je remercie très chaleureusement mes chères amies Nour El-Houda, Lemya, Leïla, Meriem et Aouatef, ainsi mes très chères sœurs qui étaient tout le temps à mes côtés, m'ont soutenue et encouragée.

Sans oublier mes chers parents pour leur soutien et leur patience, sans qui ce mémoire n'aurait jamais vu le jour. Ce mémoire est dédié à eux !

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

Introduction générale

Tout Système Informatique possède ses propres spécificités, étroitement dépendantes de son domaine d'utilisation, les applications informatiques peuvent être regroupées parmi trois catégories : *les Systèmes Transformationnelles*, les *Systèmes Interactives et les Systèmes Réactives*; c'est cette dernière catégorie, plus particulièrement les systèmes réactifs temps réel et embarqués, qui fait le contexte général de notre travail.
L'utilisation de tels systèmes est en expansion et on les trouve actuellement dans divers domaines d'applications, celles-ci concernent les chaînes de production, le domaine de l'aviation, le contrôle de centrales nucléaires, les modules d'exploration planétaire, et l'aide à la conduite automobile… etc.

La criticité du temps réel résulte du fait que les exécutions des tâches d'une application doivent respecter des contraintes temporelles dictées par les dynamiques du procédé qu'elles pilotent. Cette spécificité impose (plus encore que pour les autres types de systèmes informatiques) la nécessité de les modéliser le plus rigoureusement possible, et d'appliquer des techniques de vérification et de validation tout au long du processus de développement, sur les deux aspects : fonctionnel et temporel.
L'analyse de l'ordonnançabilité temps réel se focalise sur l'aspect temporel du système temps-réel, ça consiste à vérifier si un ensemble de tâches est ordonnançable par une politique d'ordonnancement donnée.
Dans le cas où l'analyse de l'ordonnançabilité produit un résultat négatif (cas d'existance de scénarios légaux où certaines tâches peuvent dépasser leurs échéances), une modification de certains paramètres du système peut avoir lieu et l'analyse se lance pour plusieurs itérations jusqu'à l'obtention d'un système ordonnançable.

Un soin particulier doit être apporté à l'étape de conception logicielle et matérielle du système temps-réel. Pour modéliser les systèmes temps réel et embarqués, et améliorer la spécification des besoins temps-réel, OMG a adopté le standard MARTE (Modeling and Analysis of Real-time and Embedded systems).
Il permet de prendre en compte les aspects particuliers des systèmes temps réel et embarqué, ainsi les modèles UML peuvent être annotés par des informations supplémentaires, ce qui rend la modification de la modélisation de départ faisable suite aux résultats enregistrés par les outils de vérification.

La problématique générale dans laquelle s'inscrit ce travail, consiste à la modélisation des systèmes temps-réel utilisant le profil MARTE pour but de pouvoir appliquer ensuite sur ces modèles l'analyse d'ordonnançabilité.

Le mémoire s'articule autour de quatre chapitres :

Introduction Générale : définit le contexte du mémoire et présente un aperçu général sur le travail.
Chapitre 1 : **Système temps réel** : L'objectif de ce chapitre est de décrire les caractéristiques principales des systèmes temps réel en mettant la lumière sur l'ordonnancement temps réel et l'analyse d'ordonnançabilité et les outils de développement et les langages de spécifications de ce genre de systèmes.

Chapitre 2 : **UML (Unified Modeling Language)** : le but de ce chapitre est de fournir d'abord des généralités sur ce fameux et riche langage de modélisation, puis, d'exposer ses capacités d'adaptation pour les domaines particuliers et précisément le domaine des systèmes temps réel. Ces capacités se matérialisent dans l'UML2.0 par un nouveau diagramme qui est le diagramme de temps, en plus des contraintes temporelles qui peuvent étendre les diagrammes de séquence, ainsi qu'un fort mécanisme d'extension qui est les profils, les stéréotypes et les valeurs marquée.

Chapitre 3 : **Profils UML pour le temps réel** : ce chapitre présente un état de l'art sur les profils UML supportant les aspects particuliers des systèmes temps réel à savoir : l'ordonnançabilité, la performance, le temps, la qualité de service.

Chapitre 4 : **Profil UML pour MARTE (Modeling and Analysis of Real-Time and Embedded Systems)**: ce chapitre s'intéresse particulièrement au profil MARTE et il présente la contribution, en décrivant la plateforme de modélisation UML dont on s'est servie (Papyrus) pour la proposition des éléments de modèles spécifiques à l'analyse d'ordonnançabilité ainsi l'outil d'analyse qu'on a opté pour faire les analyses d'ordonnançabilité. Et enfin, on se penchera sur une évaluation du Framework sur plusieurs critères tout en variant les politiques d'ordonnancement.

Conclusion générale : présente un résumé du travail et du mémoire, ainsi des perspectives pouvant faire suite à ce travail sont dégagées à la fin.

Chapitre 1 : Systèmes temps réel

Résumé : Les applications industrielles complexes et soumises à des contraintes temporelles rigoureuses, ont fait naître des systèmes temps réels. L'exactitude du comportement de ces systèmes qui sont de plus en plus omniprésents, ne s'évalue pas uniquement fonctionnellement mais également temporellement.
L'objectif de ce chapitre est donc de présenter les systèmes temps réel, de faire état de leurs spécificités et exigences, et de mettre en place les composantes principales de notre contexte de travail.

1.1. Introduction

Chaque Système Informatique possède ses propres spécificités, étroitement dépendantes de son domaine d'utilisation. Les applications informatiques peuvent être regroupées parmi les trois catégories suivantes [Fau03] :

Systèmes Transformationnelles : sont les systèmes qui permettant de produire un résultat à partir de données disponibles en entrées à l'initialisation de l'application. Leur traitement effectif dépend uniquement de ces données en entrée, il n'est plus contraint dans la durée. Les applications de calcul scientifique ou de gestion de bases de données sont des exemples représentatifs de système transformationnel.

Systèmes Interactives : sont les systèmes qui interagissent avec leur environnement. Leur exécution est contrôlée par une boucle d'attente des événements auxquels sont associés des traitements. Les résultats sont donc fournis en fonction des données produites par l'environnement du système, dans un délai de production satisfaisant des valeurs moyennes statistiques. Les progiciels de bureautique sont un exemple de système interactif.

Systèmes Réactives : sont les systèmes qui réagissent aux stimuli émis par leur environnement. Les résultats sont entièrement liés à l'environnement qui leur est connecté. À la différence des systèmes interactifs, l'environnement évolue de manière indépendante et n'attend pas la fin du traitement associé à un stimulus émis. Les systèmes de contrôle - commande sont des exemples de systèmes réactifs.

Cette dernière catégorie est souvent assimilée aux *Systèmes Temps Réel* ; De par leur nature, ces systèmes sont soumis à de nombreuses exigences non fonctionnelles, dont parmi ces exigences se trouvent les contraintes temporelles ou contraintes « temps réel ». Ces systèmes sont essentiellement caractérisés par la réactivité à l'environnement via des capteurs et des actionneurs, et par la criticité du temps, En effet cette criticité n'est pas de la même grandeur, alors que le retard de quelques millisecondes ou la perte d'une trame vidéo sur un réseau ne nuira qu'au confort d'un utilisateur, le retard de la décision de redresser un avion peut provoquer une catastrophe sur le plan humain ainsi que des pertes financières considérables.

Les domaines concernés vont des chaînes de production aux domaine de l'aviation et le contrôle de centrales nucléaires en passant par des robots de plus en plus perfectionnés, tels que les modules d'exploration planétaire, l'aide à la conduite automobile ou aux applications multimédia contraintes par le temps.

1.2. Définitions

Un système temps réel (STR) est un système informatique dont le fonctionnement est assujetti à l'évolution dynamique d'un procédé extérieur (environnement) qui lui est connecté et dont il doit contrôler le comportement (Figure 1.1). La correction d'un système temps réel dépend non seulement de la justesse des calculs mais aussi du temps auquel les résultats sont produits, autrement dit les contraintes temporelles qui doivent être satisfaites.
Plusieurs définitions existent dans la littérature, on présente les définitions qui suivent :

Définition 1 ***(James MARTIN dans Real-Time computer system)***
"Un calculateur temps réel peut être défini comme contrôlant son environnement en recevant des informations, les traitant et agissant en retournant les résultats suffisamment vite pour affecter, modifier le fonctionnement de l'environnement à chaque instant."

Définition 2 ***(SENOUILLET)***
"Le concept de temps réel s'applique aux organes qui sont inclus dans des processus dont les données et les résultats sont relatifs à des événements en cours."

Définition 3 ***(ABRIAL-BOURGNE)***
" Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée sans qu'elles soient trop vieilles pour l'intérêt qu'elles présentent, et par ailleurs, de réagir à celles-ci suffisamment vite pour que cette réaction ait un sens."

Un système est dit qu'il fonctionne en temps réel, à chaque fois qu'il sera question de contraintes de temps qui doivent être respectées, la notion de temps réel n'est donc pas liée à une notion de vitesse absolue mais à une notion de vitesse relative (à la dynamique de l'environnement), Adoptons ainsi l'adage suivant [DP91] : " **Un résultat juste mais hors délai est un résultat faux** ".

Les définitions des Systèmes Temps Réel présentées dans la littérature ont mis en évidence trois éléments distincts :

- Une ou plusieurs entités physiques constituant le *procédé*, dont le rôle est de détecter et d'agir ;
- Un contrôle informatique, nommé contrôleur ou *application temps réel* qui est le décideur des actions (ou réactions) du procédé.
- Des moyens de communication (électroniques et/ou informatiques) entre l'application et le procédé, qui constituent des transformateurs des langages externes des processus en langages internes du système informatique. Ce dernier reçoit des informations sur l'environnement du procédé à l'aide de *capteurs* et commande les changements d'état du procédé via des *actionneurs*.

1.3. Caractéristiques des systèmes temps réel

1.3.1. Contraintes temporelles

Le fonctionnement en temps réel impose sur le système des contraintes consiste à traiter l'information suffisamment vite pour que l'action résultante ait un sens, tout en restant dans les limites de fonctionnement nominal de la machine.

En d'autres termes, une machine fonctionnant en temps réel doit être capable de stocker toutes les informations qui lui parviennent, de les traiter toutes au bout d'un temps limite qui ne les rende pas périmées vis-à-vis du traitement que l'on veut leur faire subir, et enfin, en sorte que la somme des retards des traitements soient valables pour l'exploitant du système. [DP91]

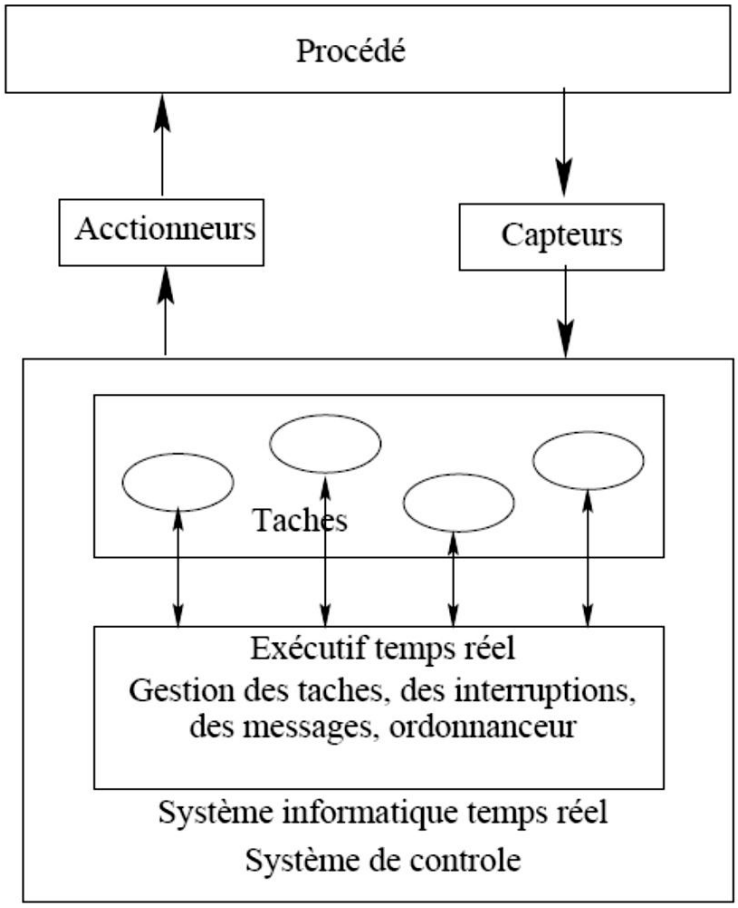

Figure 1.1. Schéma général d'une application temps réel.

Selon qu'il faille absolument respecter une contrainte lors de l'exécution du logiciel ou qu'il soit possible de la violer occasionnellement, Les contraintes temps réel sont classées dans l'une des deux catégories suivantes [TE99] :

– les contraintes temps-réel strictes " hard real-time " : sont celles qu'il faut à tout prix respecter, le non respect de ces contraintes entraîne la faute du système. Ex.: contrôle de trafic aérien, système de conduite de missile, etc.;

– les contraintes temps réel souples " soft real-time " : sont celles qu'il faut respecter autant que possible, mais qui supportent occasionnellement d'être violées, de telles contraintes sont généralement gérées par une politique " best effort ". Exemple, la perte de quelques trames d'images ou le décalage entre le son et l'image, à la projection vidéo.

1.3.2. Tâches temps-réel

Pour réduire la complexité du système et le maîtriser, il s'avère judicieux de concevoir le système sous forme d'un ensemble de tâches qui sont des programmes qui évoluent séquentiellement ou simultanément, et qui sont dédiés à traiter et piloter les composants du système Temps Réel (les actionneurs et les capteurs…).
On caractérise ainsi une Application Temps Réel comme étant une application multitâches; Mesurer périodiquement différentes grandeurs physiques (pression, température, accélération etc...) est un exemple d'une tâche temps-réel.

Le terme de tâche est le plus souvent utilisé dans le domaine du temps réel comme unité de représentation des activités concurrentes de l'architecture logique. Pour découper une application

en tâches concurrentes, on examine habituellement le parallélisme physique présent dans l'architecture support et le parallélisme logique de l'application. [CDKM99]

1.3.3. Modèle canonique des tâches temps-réel

Une application temps réel est spécifiée au moyen d'un ensemble de tâches, ces dernières peuvent être périodiques, sporadiques ou apériodiques, à contraintes temporelles strictes ou relatives.
Dans ce modèle (Figure 1.2), une tâche est définie par des paramètres chronologiques qui dénotent des délais et des paramètres chronométriques qui indiquent des dates, soit [CDKM99] :

r : sa date de réveil, c'est-à-dire le moment de déclenchement de la requête d'exécution ;

- **périodique** : quand elle se présente au système à un intervalle fixe de temps
- **apériodique** : quand la durée séparant deux arrivée est supérieure à une durée minimum de temps.
- **sporadique** : quand elle se présente aléatoirement dans le système. Ici la tâche est généralement associée à une probabilité d'arrivée.

C : sa durée d'exécution maximale quand elle dépose du processeur pour elle seule ;
D : son délai critique ou l'échéance relative, c'est le délai maximal acceptable pour son exécution ;
P (ou **T**) : sa période lorsqu'il s'agit d'une tâche périodique.
Contraintes temporelles : $0<C<D$

Autres paramètres peuvent enrichir ce modèle :
WCET : Le pire temps d'exécution, c'est le temps d'exécution (C) plus les retards.
d : Quand la tâche a des contraintes temps réel dures, l'échéance relative permet de calculer l'échéance absolue d = r + D. La transgression de l'échéance absolue cause des défauts de synchronisation.

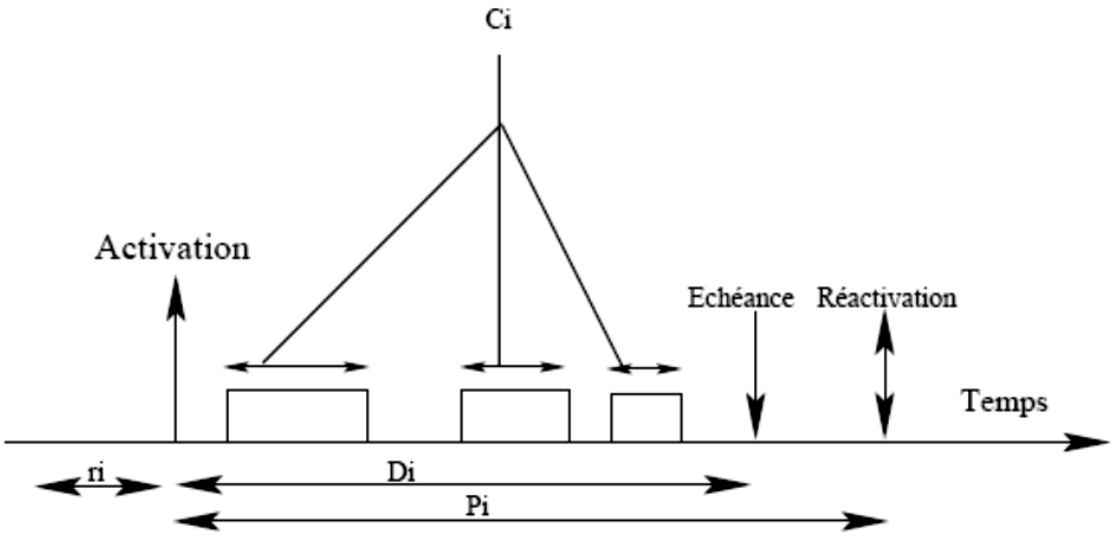

Figure 1.2. Le modèle canonique des tâches temps réel

1.3.4. Contraintes de ressources

Les tâches peuvent être vues comme des éléments du système qui consomment des ressources existantes au sein d'un même système, ces ressources peuvent être matérielles ou logiques.
Elles sont gérées en arbitrant entre les demandes que les tâches de l'application font, et les ressources effectivement disponibles dans le système.
Dans le cas de ressources actives (processeurs, réseau..), cet arbitrage consiste en un partage temporel de l'accès aux ressources. Les décisions de partage sont faites suivant un ordonnancement.

Dans le cas des ressources passives, l'objectif est de restreindre le nombre et les types d'accès (lecture/écriture) afin de maintenir la cohérence : c'est le rôle des mécanismes de synchronisation (verrous, sémaphores, conditions, moniteurs, files de messages par exemple), qui par extension permettent également aux tâches d'échanger des informations ou de se synchroniser sans qu'il y ait nécessairement de ressource à protéger (comme avec les rendez-vous par exemple).

1.3.5. Contraintes de dépendance

Les tâches, dont les comportements sont séquentiels, peuvent interagir entre elles. Il est donc nécessaire de fournir à ces tâches des moyens de [Pai06] :

- Communication et de synchronisation permettant de gérer tous les problèmes liés aux accès à des ressources communes,

  Le partage des ressources introduit une relation dynamique entre les tâches, lorsque l'ordre d'utilisation de la ressource dépend de l'ordre d'exécution des diverses tâches. La relation peut être représentée par un graphe d'allocation [CDKM99].

- L'exécution des tâches ordonnées par des critères de précédence. Cette relation entre tâches est dite statique, car elle est connue a priori et n'évolue pas. Elle est représentée par un graphe de dépendance.

Lorsque les tâches ont des dépendances statiques ou dynamiques qui les sérialisent, on introduit [CDKM99] la notion de temps de réponse global, appelé aussi délai de bout en bout dans le cas de messages; Il correspond au délai écoulé entre la date de réveil de la tâche qui reçoit le stimulus envoyé par le procédé et la fin de la dernière tâche de la série qui envoie la commande en réaction à ce stimulus.
Lorsque les tâches ne sont soumises ni à des contraintes de ressources, ni à des contraintes de précédence, elles sont qualifiées [CDKM99] de tâches indépendantes.

1.3.6. Autres contraintes

En outre des contraintes de temps ou d'accès aux ressources, la spécification de systèmes temps réel introduit parfois d'autres contraintes. Les plus courantes sont les contraintes de [Dec02]:

- Localisation ou localité : précisent si les tâches doivent s'exécuter sur un processeur donné (par exemple pour accéder à un capteur disponible uniquement sur une machine donnée, ou pour diminuer la quantité de messages réseau échangés) ;
- Anti-localité : précisent si deux tâches doivent s'exécuter sur des processeurs différents (par exemple quand il s'agit d'introduire de la redondance spatiale dans les traitements afin que le système soit tolérant aux fautes physiques).

1.4. Ordonnancement temps-réel

La spécificité du temps réel est que les exécutions des tâches d'une application doivent respecter des contraintes temporelles dictées par les dynamiques du procédé qu'elles pilotent.
La problématique générale de l'ordonnancement consiste, pour une application donnée, à déterminer la politique d'allocation des tâches sur les supports d'exécutions (processeur, contrôleur) de façon à garantir ces contraintes temporelles, puis à implémenter cette politique. Cette implémentation consiste à déterminer le mécanisme à intégrer dans l'exécutif qui lui permettra de décider de l'ordre des tâches à exécuter au fur et à mesure de l'évolution de l'application, c'est-à-dire au moment de l'observation des événements émis par le procédé. Ce mécanisme est appelé " Ordonnanceur".

1.4.1. Définition (ordonnanceur ou dispatcheur)

L'ordonnanceur est [DP91] le chef d'orchestre d'un système temps réel, son importance est d'autant plus grande qu'il est invoqué au moins à chaque fois qu'une modification intervient sur l'ensemble des tâches actives.

L'ordonnanceur a deux rôles essentiels :

– Assurer la gestion des communications de tâches de l'état bloqué à l'état éligible,

– Effectuer le choix d'une tâche dans l'ensemble des tâches éligibles.

1.4.2. Problématique de l'ordonnancement en temps réel

Le système de conduite d'une application temps réel doit piloter le procédé en ordonnançant les tâches avec deux objectifs majeurs [CDKM99] :

– En fonctionnement nominal, assurer le respect des contraintes temporelles spécifiées ;

– En fonctionnement anormal dû à des pannes matérielles ou à d'autres événements imprévus, atténuer les effets des surcharges temporelles et maintenir le procédé dans un état cohérent et sécuritaire.

Ordonnancer les tâches d'une application temps réel consiste alors à planifier l'exécution de requêtes de façon que soient respectées les contraintes de temps :

– de toutes les requêtes en fonctionnement nominal ;

– d'au moins les requêtes les plus importantes, c'est-à-dire celles qui sont nécessaires à la sécurité du procédé, en fonctionnement anormal.

Cette planification est faite par ***un algorithme d'ordonnancement*** qui fournit une description de la séquence à effectuer par les tâches, appelée séquence de planification.

Selon les applications, on cherche à satisfaire certains critères de performance comme la minimisation de temps de réponse, l'équilibrage de la charge des sites, la minimisation de la charge de communication, la minimisation du nombre ou du retard cumulé des tâches ou des messages tardifs, ou encore la minimisation du temps total d'exécution, qui présente le critère de performance qu'on utilise, au cours de notre projet, pour comparer les algorithmes d'ordonnancement dont on dispose des résultats de leurs exécutions.

1.4.3. Nature des algorithmes d'ordonnancement

Statique ou dynamique

Les ordonnanceurs statiques fondent leurs décisions d'ordonnancement sur des paramètres assignés aux tâches du système avant leur activation. À l'inverse, les ordonnanceurs dynamiques fondent leurs décisions sur des paramètres qui varient en cours de fonctionnement du système. [Dec02]

En ligne ou hors ligne

Un algorithme d'ordonnancement hors ligne construit une séquence complète de planification sur la base de tous les paramètres temporels de tâches. Cette séquence est connue avant l'exécution des tâches et peut être mise en œuvre très efficacement.

Toute fois, cette approche statique est très rigide : elle suppose que tous les paramètres, y compris les dates de réveil, soient figés et ne peut s'adapter aux changements de l'environnement.

Un algorithme d'ordonnancement en linge est capable à tout instant de l'exécution de l'application, de choisir la prochaine tâche à ordonnancer et il utilise comme informations le paramètres temporels des tâches déclenchées à cet instant. Ce choix peut être remis en cause lors de l'occurrence d'un nouveau événement sans que la date de cette occurrence n'est à être connue à l'avance.

Préemptif ou non préemptif

Un algorithme d'ordonnancement est soit préemptif soit non préemptif. Dans le premier cas, une tâche élue peut perdre le processeur au profit d'une autre tâche jugée plus urgente ou plus prioritaire;

elle passe à l'état prêt pour attendre d'être élue ultérieurement sur le même processeur ou sur un autre. Un algorithme préemptif n'est utilisable que si toutes les tâches sont préemptives.
Les algorithmes d'ordonnancement non préemptifs n'arrêtent pas l'exécution des tâches élues, il peut en résulter des temps de réponse plus longs qu'avec un ordonnancement préemptif.

Meilleur effort ou inclémence aux fautes temporelles

Pour une application à contraintes temporelles souples, on dit que la stratégie d'ordonnancement est celle du meilleur effort, si elle essaie de faire au mieux avec les processeurs disponibles.
L'application a des facultés de tolérance aux fautes temporelles. Pour une application à contraintes strictes, la stratégie d'ordonnancement doit garantir le respect des contraintes. Il y a obligation de réussite et inclémence aux fautes temporelles.

Centralisé ou réparti

L'ordonnancement est centralisé s'il s'exécute sur une architecture centralisée ou un site privilégié de l'architecture distribuée qui contient l'ensemble des paramètres des tâches. L'ordonnancement est réparti lorsque les décisions d'ordonnancement sont prises sur chaque site par un ordonnancement local après une éventuelle coopération pour effectuer un ordonnancement global.
Dans ce dernier cas peut intervenir le placement des tâches sur un site et leur migration. [CDKM99]

1.4.4. Propriétés des algorithmes

Séquence valide

Un algorithme d'ordonnancement délivre une séquence d'ordonnancement pour une configuration de tâches données. Cette séquence est valide si toutes les tâches de la configuration respectent leurs contraintes.

Configuration ordonnançable

Une configuration de tâches est ordonnançable dés qu'il existe un algorithme au moins capable de fournir une séquence valide pour cette configuration

Algorithme optimal

Un algorithme est optimal s'il est capable de produire une séquence valide pour toute configuration de tâche ordonnançable.

Test d'acceptabilité

Un algorithme en ligne crée ou modifie la séquence dynamiquement à l'arrivée de nouvelles tâches ou à la suite de dépassement d'échéance. Une tâche nouvelle peut être ajoutée à la séquence déjà formée, ou en cours de modification à la suite de suppression, et être acceptée s'il existe au moins une séquence pour laquelle toutes les tâches précédemment acceptées et cette tâche nouvelle peuvent terminer leur exécution avant leur échéance.
L'ensemble des conditions à satisfaire est appelé test d'acceptabilité. On dit parfois test de garantie car, si les tâches ne dépassent pas leurs durées d'exécution (auxquelles s'ajoutent les durées d'attente pour obtenir les ressources critiques), on peut garantir alors que les tâches ne font pas de faute temporelle.
Dans un ordonnancement réparti, le rejet d'une tâche par un test d'acceptabilité sur un site peut avoir comme conséquence d'essayer de faire migrer la tâche. [CDKM99]

1.5. Ordonnançabilité Temps-réel

Lorsqu'il s'agit de systèmes temps-réel, l'influence de l'ordonnancement sur les contraintes temporelles et celles de performance est cruciale. Ce qui exige de s'assurer du respect de telles contraintes par les techniques de l'ordonnancement, ceci conduit au besoin d'une analyse de l'ordonnançabilité ou un test de faisabilité d'un algorithme d'ordonnancement.

La maturité des techniques d'ordonnancement ont émergé plusieurs formules (qui diffère d'une technique d'ordonnancement à une autre) dont chacune, définit la condition suffisante ou nécessaire ou nécessaire et suffisante pour qu'un ensemble de tâches soit ordonnançable par une technique d'ordonnancement spécifique.

1.5.1. Définition

"*Schedulability analysis is to determine whether a specific set of tasks or a set of tasks satisfying certain constraints can be successfully scheduled (completing execution of every task by its specified deadline) using a specific scheduler.*" [Che02]

L'analyse d'ordonnançabilité consiste à vérifier si un ensemble de tâches est ordonnançable par une politique d'ordonnancement P donnée; ceci implique la vérification si cet algorithme P satisfait les contraintes temporelles de l'ensemble des tâches ;
Ceci conduit à [Hla08] vérifier si la séquence d'ordonnancement produite par P est valide.

On ne doit pas se confondre avec l'analyse de faisabilité qui consiste à vérifier si un ensemble de tâches est ordonnançable (il existe une séquence d'ordonnancement valide). L'analyse de faisabilité [Hla08] est plus générale que l'analyse d'ordonnançabilité

1.5.2. Principe

L'analyse de l'ordonnançabilité peut être utilisée à différents stades du modèle de conception. Une analyse à priori [Omg09] permet aux développeurs de détecter potentiellement des architectures temps réel infaisable et prévient des erreurs de conception coûteuses, liées particulièrement au comportement de synchronisation. D'autre part, une analyse à postériori [Omg09] d'un système existant déjà permet aux analyseurs de découvrir (avec plus d'informations quantitatives précises sur le système) des défauts temporelles, ou pour évaluer l'impact des migrations aux plateformes possibles ou des modifications au niveau des stratégies d'ordonnancement.

Dans le cas où l'analyse de l'ordonnançabilité produit un résultat négatif (cas d'existance de scénarios légaux où certaines tâches peuvent dépasser leurs échéances), une modification de certains paramètres du système peut avoir lieu et l'analyse se lance pour plusieurs itérations jusqu'à l'obtention d'un système ordonnançable.

Les techniques d'ordonnançabilité se singularisent principalement par les caractéristiques de la configuration des tâches qu'elles prennent en considération. Selon les critères retenus pour l'établissement du modèle canonique des tâches, des conditions nécessaires, suffisantes, et nécessaires et suffisantes d'ordonnançabilité ont pu être démontrées.
D'autre part, des algorithmes optimaux d'attribution des priorités aux tâches ont été définis [Fau03] pour certaines classes de problèmes. Ainsi, l'algorithme RM (pour « Rate Monotonic » qui attribue les priorités aux tâches dans l'ordre inverse de leurs périodes) est optimal pour les systèmes constitués de tâches périodiques, indépendantes, à échéance sur requête et à départs synchrones (la première activation à la date t=0 pour toutes les tâches), ordonnancés par un ordonnanceur préemptif à priorités fixes.

L'ordonnançabilité est un problème de décision et l'ordonnancement correspond à la résolution d'un problème d'optimisation. Ces deux problèmes se traduisent en théorie de l'ordonnancement par [Pai06] :
– un test d'ordonnançabilité
– l'obtention d'une séquence d'ordonnancement

1.5.3. Conditions suffisantes et conditions nécessaire

Si un test d'ordonnançabilité s'appuyant sur une condition suffisante d'ordonnançabilité produit un résultat positif, le système de tâches est définitivement ordonnançable. Par contre, si le résultat est négatif, le système peut être ordonnançable ou non.
De la même manière, si un test d'ordonnançabilité s'appuyant sur une condition nécessaire d'ordonnançabilité produit un résultat positif, l'ensemble de tâches peut ne pas être ordonnançable alors que s'il est négatif, le système de tâches est définitivement non ordonnançable.

Trois approches majeures pour la vérification analytique de l'ordonnançabilité [Pai06] :

Critères basés sur les facteurs d'utilisation et de charge
Cette analyse se base sur le facteur d'utilisation (ou de charge) pour déterminer si une configuration de tâches est ordonnançable ou non.

Critères basés sur le pire temps de réponse
Le pire temps de réponse d'une tâche sous une politique d'ordonnancement précise est le plus grand temps de réponse des instances de cette tâche pour tous les scénarios d'exécution possibles sous cette politique d'ordonnancement.

Critères basés sur la demande processeur
Test d'ordonnançabilité consistant à vérifier que toutes les requêtes devant s'exécuter dans tout intervalle de temps ne dépassent pas la capacité du processeur (la longueur de l'intervalle considéré).

1.5.3.1. Conditions d'ordonnançabilité sous des algorithmes à priorité fixe

– RM (Rate Monotonic) : algorithme à priorité fixe pour des tâches périodiques (la tâche la plus prioritaire est celle de plus petite période). Cet algorithme d'ordonnancement est optimum pour les ensembles de tâches à priorité fixe dont les périodes des tâches sont égales à leurs échéances.
Ainsi si un tel ensemble n'est pas ordonnançable par l'algorithme RM, alors il ne le sera par aucun algorithme à priorité fixe. Il existe un test simple permettant de vérifier la faisabilité de l'ordonnancement d'un ensemble de tâches par un algorithme RM. Soit un ensemble de n tâches devant être ordonnancé, avec C_i le temps d'exécution de la tâche i, et T_i sa période. Si la formule

$$\sum_{i=0}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)$$

est vérifiée alors l'ensemble de tâches est ordonnançable. Cette condition de faisabilité est souvent utilisée sous une forme un peu différente. Si on calcule

$$\lim_{n \to \infty} n(2^{\frac{1}{n}} - 1)$$

on obtient la valeur ln(2) soit environ 0.69. On retrouve alors une valeur très souvent utilisée de 69% d'utilisation du temps CPU comme test de faisabilité. Mais si ce test est suffisant, il n'est pas nécessaire, car il existe des ensembles de tâches ne remplissant pas cette condition de faisabilité mais pouvant quand même être ordonnancés par l'algorithme RM.
– DM (Deadline Monotonic) : algorithme à priorité fixe pour des tâches périodiques (la tâche la plus prioritaire est celle de petite échéance). C'est un algorithme qui ressemble à l'algorithme RM, a moins que sa condition d'optimalité est différente ; c'est-à-dire que cet algorithme d'ordonnancement est optimum pour les ensembles de tâches à priorité fixe dont les échéances sont inférieures à leurs période. Si la formule suivante

$$\sum_{i=0}^{n} \frac{C_i}{D_i} \leq n\left(2^{\frac{1}{n}} - 1\right), D_i \text{ est l'échéance de la tâche } i.$$

est vérifiée alors l'ensemble de tâches est ordonnancable.

1.5.3.2. Algorithmes à priorité dynamique

– EDF (Earlest Deadline First) : algorithme à priorité dynamique pour tâches périodiques (la tâche la plus prioritaire est celle de plus petite échéance). Comme pour l'algorithme Rate Monotonic, il existe un test de faisabilité pour l'EDF.

En reprenant la notation C_i pour le temps d'exécution de la tâche ***i*** et T_i pour sa période, le test de faisabilité pour un ensemble de n tâches peut être exprimé de la manière suivante :

$$\sum_{i=0}^{n} \frac{C_i}{T_i} \leq 1$$

Comme l'exprime le test de faisabilit2 donné ci-dessus, l'algorithme EDF peut, dans les meilleurs des cas, obtenir 100% d'utilisation du temps CPU, mais cela masque l'un des principaux défauts de cet algorithme. Si ces performances sont bonnes lors d'une utilisation 'normale', elles ont malheureusement tendance à s'effondrer lors des périodes de surcharge, notamment à cause du fait que les échéances manquées ne sont pas prévisibles et ne correspondent pas forcément aux tâches les moins ((importantes)) du système.

– LLF (Least Laxity Time) : algorithme à priorité dynamique pour des tâches périodique. En effet, c'est un algorithme variant de l'algorithme EDF où la tâche la plus prioritaire est celle dont marge est la plus petite. La marge d'une tâche ***i*** est calculée comme suit :

$$Marge_i = DealLine_i - Temps\ d'exécution\ restant_i$$

Cet algorithme est optimal sur 1seul processeur et meilleur qu'EDF en multiprocesseur, par contre, il est difficile à implanter car il nécessite de maintenir à jour le temps de calcul en plus qu'il demande un grand nombre de changement de contexte en préemptif

1.6. Développement des systèmes temps réel

La complexité liée aux architectures temps réel rend la conception et le développement des logiciels temps réel de plus en plus difficiles. Cette complexité a conduit les ingénieurs à utiliser une méthode de développement en étapes, le cycle en V. Il est un fait établi que le nombre de lignes de code des applications ne cesse d'augmenter. Le temps consacré par les ingénieurs et techniciens à l'écriture de ces lignes de code et à leur débogage représente la part la plus importante du budget consacré à la réalisation d'une application. Dans un premier temps, on décrit le cycle en V utilisé par de nombreux ingénieurs. Afin de tenir compte des contraintes de coût de plus en plus fortes, on présente ensuite une méthode de réduction de ce cycle aboutissant à des temps de développement considérablement réduits.

1.6.1. Cycle de développement d'un système temps réel

La conception des systèmes temps réel nécessite la résolution d'un grand nombre de problèmes matériels et logiciels. Afin de gérer au mieux cette complexité les méthodes de développement sont généralement basées sur un principe de hiérarchisation permettant de décrire le système comme un ensemble de sous-systèmes plus simples donc plus faciles à concevoir. Ce principe est appliqué dans le cycle de développement le plus utilisé actuellement pour la conception des systèmes temps réel. Ce cycle de développement appelé cycle en V permet selon une hiérarchisation descendante par étape, d'aboutir à la conception détaillée d'une application à partir d'une description abstraite. A chacune des étapes est associée une étape de validation. Le cycle de

développement va donc consister à réaliser, une par une, toutes les étapes de la branche descendante, puis à valider chacune de ces étapes en remontant dans la hiérarchie. La validation d'une étape de conception entraîne la remontée vers une étape de validation de niveau hiérarchique supérieur. La non validation d'une étape de conception entraîne la réexécution de cette étape et de toutes les étapes inférieures ainsi que leur revalidation [GMSB96] [Len99][Per90].

Pour la réalisation du logiciel temps réel on peut distinguer trois grandes étapes de conception : la modélisation, la spécification et l'implantation. A ces trois étapes correspondent respectivement trois étapes de validation : les tests unitaires, l'intégration et la validation du système (Figure 1. 3).

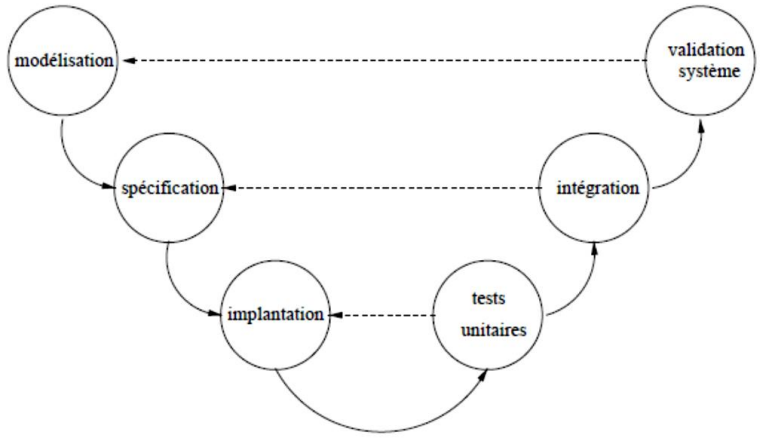

Figure 1.3. Cycle de développement d'un système temps réel.

Modélisation : Cette première étape réalisée par l'automaticien consiste à déterminer et à décrire sous forme d'équations mathématiques, le comportement que devra avoir le système.

Spécification : C'est une description "haut niveau" du logiciel. Elle permet de définir les algorithmes à mettre en oeuvre et définir les contraintes temporelles sur les opérations qui constituent les algorithmes.

Implantation : C'est la réalisation proprement dite du logiciel. Elle consiste à écrire les programmes qui doivent implanter la spécification sur l'architecture matérielle.

Tests unitaires : Cette première étape de validation est souvent longue et fastidieuse. Il s'agit ici de déboguer et de s'assurer que chacune des opérations, que les fonctions de communication inter-processeur, etc. fonctionnent correctement. Cette étape est souvent réalisée conjointement avec l'implantation : une opération est souvent testée dès qu'elle a été écrite.

Intégration : Ici, tous les éléments du logiciel sont assemblés. On vérifie alors que le système se comporte comme défini par le modèle. Si ce n'est pas le cas, la spécification est remise en cause, l'implantation est modifiée.

Validation système : Cette étape consiste à vérifier que le comportement de l'application est conforme au cahier des charges. Si cette étape ne valide pas le système, le modèle est remis en cause, il doit être modifié, affiné. Le cycle de développement entier est ré-exécuté.

1.6.2. Réduction du cycle de développement

Le cycle en V, grâce à la hiérarchisation des problèmes à résoudre, a montré son efficacité dans de nombreuses applications. Néanmoins, la complexité croissante des applications induit un coût de développement élevé. Afin de garantir la compétitivité de telles applications, il est nécessaire de minimiser les temps de développement, c'est-à-dire réduire le cycle de développement en V. Concevoir un système temps réel en suivant un cycle de développement en V consiste à raffiner un cahier des charges lors d'itérations successives du cycle jusqu'à obtenir le système automatisé recherché (Figure 1.4). Le temps de développement d'une application temps réel est donc proportionnel au temps de réalisation de chaque étape du cycle, mais aussi du nombre de réitérations du cycle. L'utilisation d'outils de développement issus de méthodes formelles est un bon moyen pour réduire à la fois le nombre de réitérations du cycle en V et le temps de réalisation d'un cycle. En effet, l'intérêt de ces méthodes est de reposer sur des langages de spécification rigoureux fondés sur un ensemble de règles mathématiques qui autorisent des vérifications et de

la génération automatique d'un code sain conforme à la spécification. Les gains que l'on peut espérer de tels méthodes et outils se situent à plusieurs niveaux du cycle de développement :
– la sémantique des langages de spécification des systèmes temps réel tend à se rapprocher le plus possible de la sémantique des outils de modélisation. Ainsi l'étape de spécification est grandement simplifiée, les erreurs dues au passage du modèle à la spécification sont réduites. La validation de la spécification lors de l'étape d'intégration est accélérée car la spécification converge plus rapidement vers le modèle en nécessitant moins d'itérations du cycle spécification - implantation - tests unitaires - intégration ;

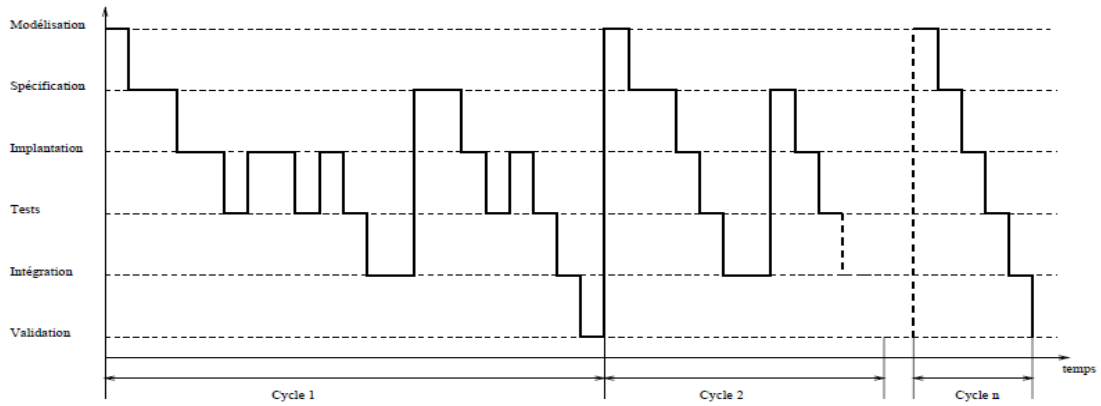

Figure 1.4. Itérations du cycle en V.

– des outils de vérification permettent de détecter des erreurs de spécification. Ce point est particulièrement important car plus une erreur est détectée tôt moins elle engendrera de réitérations du cycle ;
– la génération automatique de code permet de s'affranchir de l'écriture d'un grand nombre de lignes de code et de réutiliser les développements d'autres applications. On réduit considérablement les tests unitaires. La spécification ayant été vérifiée, l'intégration n'est plus nécessaire ;
– la traçabilité de l'application est améliorée car généralement les méthodes formelles intègrent des outils de génération automatique de documentation, le code bas niveau est souvent simple et commenté ; il est alors très facile de retrouver une erreur sur la spécification à partir d'une erreur sur le code bas niveau.
L'utilisation d'outils de simulation réduit aussi considérablement le nombre d'itérations du cycle en V en permettant de détecter des erreurs éventuelles dans les lois de commande, avant toute spécification et implantation.

Le langage de spécification intègre des méthodes de simulation autorisant leur utilisation lors de la modélisation. Ainsi, les étapes de modélisation et de spécification peuvent être réunies en une seule étape. Les vérifications et les simulations que l'on peut alors réaliser permettent de réduire les erreurs de modèle. On espère ainsi obtenir la validation du système dès la première itération du cycle de développement. La génération du code est automatique et s'appuie sur un ensemble minimal de librairie de fonctions de très bas niveau portables sur différentes architectures cibles. Les tests unitaires sont à réaliser une seule fois pour chaque type d'architecture. Le coût de développement et de tests unitaires est donc partagé entre toutes les applications qui utilisent la librairie. La génération de code et les librairies peuvent être dans le meilleur des cas certifiées. On apporte ainsi la preuve que l'implantation est conforme à la spécification, qui a elle-même été vérifiée.Ce nouveau cycle en V réduit doit permettre d'aboutir dans le meilleur des cas à une seule itération du cycle (Figure 1.5) comparable à un cycle de développement en cascade sans remontée [GMSB96] pour lequel chaque étape est validée avant de passer à la suivante. L'objectif de cette thèse est de fournir une méthode de conception de systèmes temps réel embarqués se rapprochant le plus possible de ce cycle en V réduit.

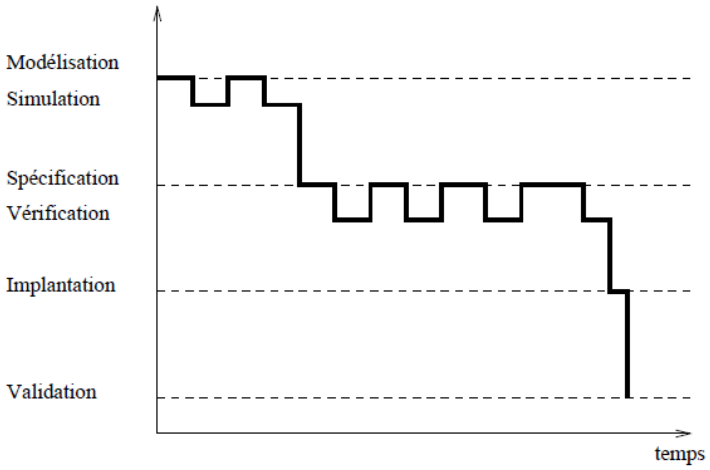

Figure 1.5. Itération du cycle en V réduit idéal.

1.7. Spécification des systèmes temps réel

Dans un système temps réel, les aspects matériel et logiciel sont fortement liés. Pour réaliser l'architecture matérielle du système, il est nécessaire de connaître les algorithmes devant y être implantés afin de la dimensionner correctement (choix des capteurs et actionneurs, précision des convertisseurs, puissance des processeurs, etc.). De même, pour réaliser le logiciel il est nécessaire de connaître non seulement les algorithmes ainsi que les contraintes temporelles sur l'exécution de ces algorithmes, mais il est aussi nécessaire de connaître l'architecture matérielle afin de générer du code exécutable adapté à l'architecture (jeu d'instructions des processeurs, accès capteurs et actionneurs, protocoles des médias de communication, etc.). La spécification est la base servant de point de départ à la conception du matériel et du logiciel. Elle doit donc décrire à un haut niveau les algorithmes, l'architecture matérielle et les contraintes temporelles d'exécution des algorithmes sur le processeur.

1.7.1. Spécification des algorithmes

Un système automatisé est composé d'un processus et d'un système temps réel en perpétuelle interaction avec celui-ci. Pour prendre en compte l'aspect infiniment répétitif de ces interactions (le nombre d'interactions ne peut être borné, il peut donc être considéré comme infini), il est nécessaire d'étendre la notion classique d'algorithme. Ainsi dans le cadre de la conception des applications temps réel, un algorithme est une séquence finie d'opérations, réalisables en un temps fini sur un support matériel fini, répétée infiniment et dont l'exécution est déclenchée par les événements d'entrée. Dans un système temps réel complexe deux types d'algorithmes applicatifs distincts sont à prendre en compte : la commande et le contrôle. Les algorithmes de commande décrivent les actions qui doivent être appliquées au processus. Les algorithmes de contrôle déterminent, en fonction des événements d'entrées, quelles sont les actions à appliquer au processus à un instant donné.

1.7.1.1. Algorithmes de commande

Les algorithmes de commande sont généralement constitués d'un ensemble de fonctions généralement issues de la théorie du signal, des images ou de l'automatique et dont la finalité est de réaliser des transformations sur des signaux. Les calculs impliqués dans ces algorithmes peuvent être très simples (addition et soustraction de nombres entiers) aussi bien que très complexes (fonctions trigonométriques, logarithme, transformée de Fourrier appliquées à des nombres en représentation à virgule flottante). Quelle que soit cette complexité, les algorithmes de commande nécessite souvent la mémorisation du passé de certains signaux..

1.7.1.2. Algorithmes de contrôle

Le contrôle est nécessaire pour répondre principalement à trois besoins : le changement de paramètres de certaines lois de commande, le séquencement et le parallélisme entre actions et le changement de modes de fonctionnement.

1.7.2. Spécification des contraintes temporelles

1.7.2.1. Contrainte de cadence

Par définition, un système temps réel est un système informatique dont le logiciel est soumis à des contraintes spécifiques de temps, les contraintes temps réel. Il est nécessaire de pouvoir spécifier ces contraintes. Leur prise en compte à l'implantation influe sur l'ordre d'exécution des calculs qui constituent le logiciel que doit exécuter le CPU. Pour un système temps réel, une contrainte sur la période d'échantillonnage permet de fixer l'intervalle de temps minimum qui sépare l'arrivée de deux événements d'entrée. Cette contrainte permet d'imposer au système réactif un rythme minimal d'entrée des événements. Le système réactif doit être capable de traiter tous les événements dont l'intervalle entre les dates d'arrivée est au moins égale à la période d'échantillonnage. Ce type de contraintes est appelé contrainte de cadence. A chacun des sous-processus qui composent le processus on associe un algorithme de contrôle-commande. L'exécution de chacun de ces sous-algorithmes doit respecter sa propre contrainte de cadence correspondant à la fréquence d'échantillonnage définie pour ses entrées. Ainsi dans les systèmes complexes il est généralement nécessaire de pouvoir spécifier plusieurs contraintes de cadence.

1.7.2.2. Contrainte de latence

Les opérations d'échantillonnage et de blocage peuvent être modélisées par un simple retard et les algorithmes de commande peuvent être modélisés par une fonction de transfert à laquelle est associée un retard pur équivalent au temps de calcul de l'algorithme. La somme de ces deux retards est le temps de réponse du système temps réel. Pour garantir que ce temps de réponse ne dépassera jamais une certaine valeur critique à partir de laquelle les performances du système ne seraient plus satisfaisantes ou à partir de laquelle le système serait instable, il est nécessaire de borner le temps de calcul. Cette borne est appelée contrainte de latence. De manière générale, elle correspond à l'intervalle de temps maximum pendant lequel le système réactif doit produire les actions engendrées par les événements d'entrée (Figure 1.6).

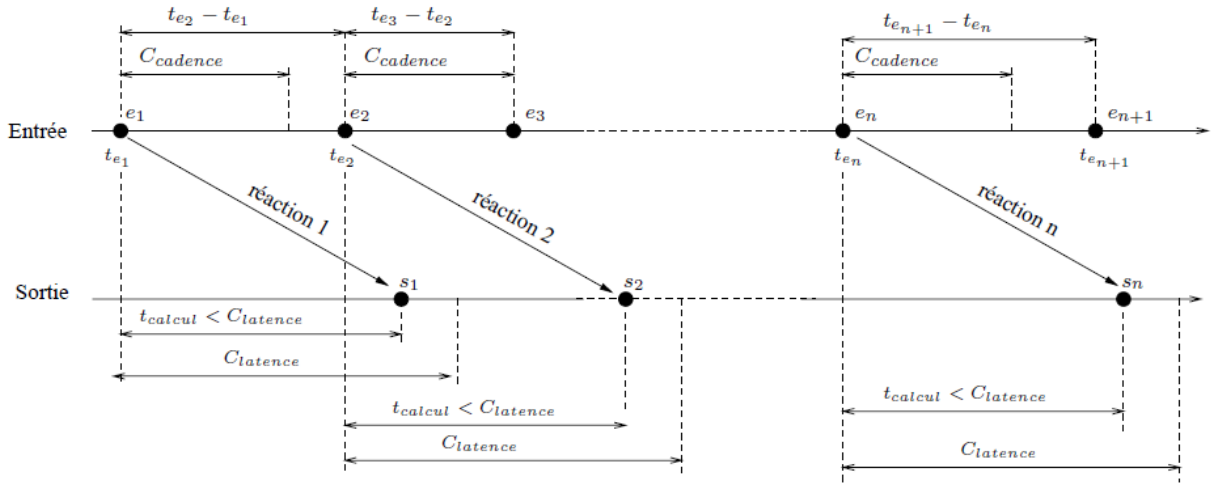

Figure 1.6. Latence.

1.7.3. Spécification des contraintes matérielles

La spécification du logiciel ne peut être complètement indépendante de l'architecture matérielle car les lois de commande ont été définies pour certains capteurs et certains actionneurs. L'utilisation d'une architecture disposant d'autres capteurs et d'autres actionneurs entraînerait obligatoirement une modification des lois de

commande et donc des algorithmes. Dans le cadre d'un cycle de développement en V réduit il est nécessaire de spécifier à la fois les algorithmes et l'architecture de manière à permettre une implantation automatique des algorithmes sur l'architecture. Lors de cette implantation sur une architecture distribuée les opérations de l'algorithme doivent être affectées à un processeur, c'est la **distribution**.
La distribution des opérations de gestion des capteurs et actionneurs ne peut être arbitraire car chaque capteur et chaque actionneur sont gérés par un microprocesseur. Il est donc nécessaire de spécifier ce lien afin que l'opération de gestion de chaque capteur et de chaque actionneur soit affectée au processeur sur lequel le capteur et l'actionneur correspondant sont reliés. De plus, pour pouvoir générer les communications inter-processeur, il est nécessaire de connaître la topologie du réseau du calculateur, c'est-à-dire le type de média de communications (lien, bus SAM, bus RAM) ainsi que les connexions des processeurs à ces médias. Enfin, pour pouvoir générer un code exécutable optimisé, il est nécessaire de connaître les principales caractéristiques des composants utilisés (temps de calcul des opérations sur les processeurs, taux de transfert sur les médias de communication, etc.).

1.8. Langages de spécification des systèmes temps réel

La programmation des systèmes temps réel est un domaine de l'informatique en forte expansion. On peut expliquer ce phénomène, d'une part, par le nombre et la complexité croissants des applications et, d'autre part, par la jeunesse de cette science. De nombreux laboratoires de recherche et de nombreuses entreprises s'emploient activement à combler les lacunes des méthodes de conception des systèmes temps réel. C'est pourquoi durant ces dernières années un grand nombre d'outils d'aide à la spécification est apparu sur le marché. Néanmoins, ceux-ci ne répondent pas à tous les besoins. On trouve généralement deux classes de langages de spécification, formelle et non formelle. Dans la première classe on trouve une variété de langages, comme : RT-Lotos. Pour un langage de spécification soit qu'il est formel ou non, il possède au moins deux caractéristiques importantes dans le cadre de la spécification des applications temps réel embarquées. La première reflète la manière dont il permet de décrire les algorithmes. On peut ainsi classer les langages en quatre catégories : impératif, déclaratif, logique et objet. La deuxième caractéristique importante est la manière dont ces langages intègrent la notion de temps. Selon ce critère on peut classer les langages en deux grandes familles : les langages synchrones et les langages asynchrones.

1.9. Conclusion

En résumé, le processus de développement des systèmes temps réels est une tâche délicate qui consiste en le raffinement successif et progressif des spécifications. Les langages de spécification formelle se fait un fort outil de spécification grâce à sa base mathématique très solide permettant de gérer et de vérifier les aspects complexes classiques des systèmes temps réel, ainsi que la génération automatique de code. Tandis que confier cette tâche à un langage non formel comme l'UML, s'avère un vrai défi malgré sa richesse sémantique et d'expressivité et ses forts mécanismes d'extensibilité. Dans le chapitre qui suit on introduit l'UML de point de vue support du temps réel.

Chapitre 2: UML (Unified Modeling Language)

Résumé : UML est le langage standard de modélisation visuelle objet qui peut être très utile en tant que la langue de communication du système de conception. Cependant, UML comme un langage de modélisation des systèmes temps réel, a ses limites. Malgré qu'il comporte essentiellement une syntaxe et sémantique riche, mais cela n'est pas assez.

2.1. Généralités sur UML

2.1.1. Introduction

Les techniques de programmation n'ont cessé de progresser depuis l'époque de la programmation en langage binaire (cartes perforées, switch) à nos jours. Cette évolution a toujours été dictée par le besoin de concevoir et de maintenir des applications toujours plus complexes. La programmation par cartes perforées, switch ou câblage (de 1800 à 1940) a ainsi fait place à des techniques plus évoluées, comme l'assembleur (1947), avec l'arrivée de l'ordinateur électronique né des besoins de la guerre. Des langages plus évolués ont ensuite vu le jour comme Fortran en 1956 ou Cobol en 1959. Jusque là, les techniques de programmation étaient basées sur le branchement conditionnel et inconditionnel (goto) rendant les programmes importants extrêmement diciles à développer, à maîtriser et à maintenir. La programmation structurée (Pascal en 1970, C en 1972, Modula et Ada en 1979, . . .) a alors vu le jour et permis de développer et de maintenir des applications toujours plus ambitieuses. L'algorithmique ne se su sant plus à elle seule à la fin des années 1970, le génie logiciel est venu placer la méthodologie au cœur du développement logiciel. Des méthodes comme Merise (1978) se sont alors imposées. La taille des applications ne cessant de croître, la programmation structurée a également rencontré ses limites, faisant alors place à la programmation orientée objet (Simula 67 en 1967, Smalltalk en 1976, C++ en 1982, Java en 1995, . . .). La technologie objet est donc la conséquence ultime de la modularisation dictée par la maîtrise de la conception et de la maintenance d'applications toujours plus complexes. Cette nouvelle technique de programmation a nécessité la conception de nouvelles méthodes de modélisation. UML (Unified Modeling Language en anglais, soit langage de modélisation objet unifié) est né de la fusion des trois méthodes qui s'imposaient dans le domaine de la modélisation objet au milieu des années 1990 : OMT, Booch et OOSE. D'importants acteurs industriels (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys etc.) s'associent alors à l'effort et proposent UML 1.0 à l'OMG (Object Management Group) qui l'accepte en novembre 1997 dans sa version 1.1. La version d'UML en cours en 2008 est UML 2.1.1 qui s'impose plus que jamais en tant que langage de modélisation standardisé pour la modélisation des logiciels.

2.1.2. Vues d'UML

UML propose différents modèles pour représenter les différents points de vue de la modélisation. Les cinq vues sont :

- la vue logique (intégrité de conception) : perspective abstraite de la solution (classes, relations, machines états-transitions, etc.) ;
- la vue des composants (intégrité de gestion du code) : perspective physique de l'organisation du code (modules, composants, concepts du langage ou de l'environnement d'implémentation) ;

- la vue des processus (intégrité d'exécution) : perspective sur les activités concurrentes et parallèles (tâches et processus) ;
- la vue de déploiement (intégrité de performance) : répartition du système (logiciel) à travers un réseau ;
- la vue des cas d'utilisation (intégrité de conception), qui guide et justifie les autres : moyen rigoureux et systématique pour guider la modélisation (cas d'utilisation, scénarios, collaborations d'objets, machines à états).

2.1.3. Diagrammes d'UML

UML n'est pas une méthode (i.e. une description normative des étapes de la modélisation) : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information. Aux graphiques sont bien sûr associés des textes qui expliquent leur contenu. UML est donc un **métalangage** car il fournit les éléments permettant de construire le modèle qui, lui, sera le langage du projet. Il est impossible de donner une représentation graphique complète d'un logiciel, ou de tout autre système complexe, de même qu'il est impossible de représenter entièrement une statue (à trois dimensions) par des photographies (à deux dimensions). Mais il est possible de donner sur un tel système des vues partielles, analogues chacune à une photographie d'une statue, et dont la conjonction donnera une idée utilisable en pratique sans risque d'erreur grave. UML 2.0 comporte ainsi treize types de diagrammes représentant autant de vues distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

- Diagrammes structurels ou diagrammes statiques (UML Structure)
  - diagramme de classes (Class diagram)
  - diagramme d'objets (Object diagram)
  - diagramme de composants (Component diagram)
  - diagramme de déploiement (Deployment diagram)
  - diagramme de paquetages (Package diagram)
  - diagramme de structures composites (Composite structure diagram)
- Diagrammes comportementaux ou diagrammes dynamiques (UML Behavior)
  - diagramme de cas d'utilisation (Use case diagram)
  - diagramme d'activités (Activity diagram)
  - diagramme d'états-transitions (State machine diagram)
  - diagramme de séquence (Sequence diagram)
  - diagramme de communication (Communication diagram)
  - diagramme global d'interaction (Interaction overview diagram)
  - diagramme de temps (Timing diagram)

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Les plus utiles pour la maîtrise d'ouvrage sont les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions. Les diagrammes de composants, de déploiement et de communication sont surtout utiles pour la maîtrise d'oeuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique.

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système. C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre.

Diagramme de classes

Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Il représente l'architecture conceptuelle du système : il décrit les classes que le système utilise, ainsi que

leurs liens, que ceux-ci représentent un emboîtage conceptuel (héritage) ou une relation organique (agrégation).

Diagramme d'objets

Le diagramme d'objets permet d'éclairer un diagramme de classes en l'illustrant par des exemples. Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

Diagramme d'états-transitions

Le diagramme d'états-transitions représente la façon dont évoluent (i.e. cycle de vie) les objets appartenant à une même classe. La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

Diagramme d'activités

Le diagramme d'activités n'est autre que la transcription dans UML de la représentation du processus telle qu'elle a été élaborée lors du travail qui a préparé la modélisation : il montre l'enchaînement des activités qui concourent au processus.

Diagramme de séquence et de communication

Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur. Il indique les objets que l'acteur va manipuler et les opérations qui font passer d'un objet à l'autre. On peut représenter les mêmes opérations par un diagramme de communication, graphe dont les nœuds sont des objets et les arcs (numérotés selon la chronologie) les échanges entre objets. En fait, diagramme de séquence et diagramme de communication sont deux vues différentes mais logiquement équivalentes (on peut construire l'une à partir de l'autre) d'une même chronologie. Ce sont des diagrammes d'interaction.

2.1.4. Présentations des modèles UML

La présentation d'un modèle UML se compose de plusieurs documents écrits en langage courant et d'un document formalisé : elle ne doit pas se limiter au seul document formalisé car celui-ci est pratiquement incompréhensible si on le présente seul. Un expert en UML sera capable dans certains cas de reconstituer les intentions initiales en lisant le modèle, mais pas toujours ; et les experts en UML sont rares. Voici la liste des documents qui paraissent nécessaires :

Présentation stratégique : elle décrit pourquoi l'entreprise a voulu se doter de l'outil considéré, les buts qu'elle cherche à atteindre, le calendrier de réalisation prévu, etc.

Présentation des processus de travail par lesquels la stratégie entend se réaliser : pour permettre au lecteur de voir comment l'application va fonctionner en pratique, elle doit être illustrée par une esquisse des écrans qui seront affichés devant les utilisateurs de terrain.

Explication des choix qui ont guidé la modélisation formelle : il s'agit de synthétiser, sous les yeux du lecteur, les discussions qui ont présidé à ces choix.

Modèle formel : c'est le document le plus épais et le plus difficile à lire. Il est préférable de le présenter sur l'Intranet de l'entreprise. En effet, les diagrammes peuvent être alors équipés de liens hypertextes permettant l'ouverture de diagrammes plus détaillés ou de commentaires.

On doit présenter en premier le diagramme de cas d'utilisation qui montre l'enchaînement des cas d'utilisation au sein du processus, enchaînement immédiatement compréhensible ; puis le diagramme d'activités, qui montre le contenu de chaque cas d'utilisation ; puis le diagramme de séquence, qui montre l'enchaînement chronologique des opérations à l'intérieur de chaque cas d'utilisation. Enfin, le diagramme de classes, qui est le plus précis conceptuellement mais aussi le plus difficile à lire car il présente chacune des classes et leurs relations (agrégation, héritage, association, etc.)

2.1.5. Eléments de modèle

- **Chaîne (string) :** suite de caractères.
- **Nom (name) :** chaîne utilisée pour identifier un élément d'un modèle.
- **Etiquette (label) :** chaîne attachée à un symbole graphique.

- **Mot-clé (keyword):** qui a pour format d'un mot-clé : «mot-clé»
- **Expression (expression) :** chaîne qui évalue une valeur (d'un type particulier).
- **Collection (collection) :** regroupement d'objets (terme générique qui ne précise pas la nature du regroupement).
- **Cardinalité (cardinality) :** nombre d'éléments dans une collection.
- **Multiplicité (multiplicity) :** spécification d'échelle qui précise les cardinalités autorisées.
- **Type primitif (primitive type) :** type prédéfini (booléen, expression, liste, chaîne, point, nom, multiplicité, temps, non interprété).
- **Contrainte (constraint) :** relation sémantique sur des éléments de modélisation qui spécifie une proposition devant être vérifiée. le format d'une contrainte est : { contrainte }
- **Propriété (property) :** elle représente une caractéristique d'un élément de modélisation et qui peut recevoir des valeurs.
- **Valeur marquée (tagged value) :** définition explicite d'une propriété en précisant ses nom/valeur.
- **Commentaire (comment) :** chaîne attachée à un élément de modélisation, mais sans sémantique.
- **Note (note) :** information textuelle (commentaire bien évidemment mais aussi contrainte, valeur marquée, corps d'une méthode ou d'autres chaînes valuées) associée à un (ou plusieurs) élément(s) d'un modèle.
- **Dépendance (dependency) :** relation d'obsolescence (unidirectionnelle) entre deux éléments de modélisation.
- **Dichotomie type/instance (type-instance correspondence) :** séparation de l'essence de l'élément de modélisation (sa spécification) et de la manifestation avec ses valeurs (sa réalisation) de ce type ; cela concerne la séparation classe/objet, association/lien, paramètre/valeur, opération/appel, etc.
- **Dichotomie type/classe (type/class dichotomy) :** séparation de la spécification d'un élément de modélisation et de la réalisation de cette spécification.
- **Sous-système (subsystem) :** ensemble d'éléments de modélisation groupés qui constituent une spécification du comportement présenté par les autres éléments du modèle.
- **Modèle (model) :** abstraction (sémantiquement fermée) d'un système (physique), contenant des éléments de modélisation.
- **Paquetage (package) :** regroupement (avec encapsulation) d'éléments de modélisation, selon un critère logique (et non fonctionnel).
- **Stéréotype (stereotype) :** extension de la sémantique d'un élément du méta-modèle. Son format est: « stéréotype » (ou par une icône).

2.1.6. Mécanismes d'extension

Ils permettent de personnaliser et d'étendre l'UML. En effet, grâce à ces mécanisme, l'UML peut étendre ses éléments de modélisation en leur rajoutant de nouvelles propriétés ou en modifiant celles existantes ou de créer de nouveaux élément de modélisation:

- **Les stéréotypes:** ce sont utilisés pour classifier ou marquer les éléments de modélisation ou pour introduire de nouveaux types d'éléments de modélisation. Chaque stéréotype définit un ensemble de propriétés qui seront héritées par éléments de modélisation qui appliquent ce stéréotype. Donc les stéréotypes sont des méta-modèles (ils ne servent pas à créer des modèles, mais plutôt de nouveaux éléments de modélisation bien personnalisés pour un domaine d'application bien spécifié)
- **Les propriétés:** ce sont des caractéristiques d'un élément de modélisation.
- **Les contraintes:** ce sont de propriétés qui ont pour vocation la spécification des sémantiques ou des conditions qui doivent être satisfaites pour les éléments de modélisation.
- **Les valeurs marquées:** ce sont des propriétés qui ont pour vocation la spécification des valeurs des attributs.

Utilisant ces mécanismes d'extension, le méta-modèle d'UML peut être étendu et personnalisé pour un domaine spécifique.

2.2. UML et le cycle de vie des logiciels

La problématique que pose la mise en œuvre d'UML est simple : comment passer de l'expression des besoins au code de l'application. A maintes reprises, UML n'est qu'un langage de modélisation, ce n'est pas une méthode. En e et, UML ne propose pas une démarche de modélisation explicitant et encadrant toutes les étapes d'un projet, de la compréhension des besoins à la production du code de l'application. Une méthode se doit de définir une séquence d'étapes, partiellement ordonnées, dont l'objectif est de produire un logiciel de qualité qui répond aux besoins des utilisateurs dans des temps et des coûts prévisibles. Bien qu'UML n'est pas une méthode, ses auteurs précisent néanmoins qu'une méthode basée sur l'utilisation UML doit être :

Pilotée par les cas d'utilisation : La principale qualité d'un logiciel étant son utilité, c'est-à dire son adéquation avec les besoins des utilisateurs, toutes les étapes, de la spécification des besoins à la maintenance, doivent être guidées par les cas d'utilisation qui modélisent justement les besoins des utilisateurs.

Centrée sur l'architecture : L'architecture est conçue pour satisfaire les besoins exprimés dans les cas d'utilisation, mais aussi pour prendre en compte les évolutions futures et les contraintes de réalisation. La mise en place d'une architecture adaptée conditionne le succès d'un développement. Il est important de la stabiliser le plus tôt possible.

Itérative et incrémentale : L'ensemble du problème est décomposé en petites itérations, définies à partir des cas d'utilisation et de l'étude des risques. Les risques majeurs et les cas d'utilisation les plus importants sont traités en priorité. Le développement procède par des itérations qui conduisent à des livraisons incrémentales du système.

Dans tous les cas, il faut garder à l'esprit qu'une méthode n'est pas une formule magique. Le fait de produire des diagrammes UML selon un ordre établi n'est en aucun cas une garantie de réussite. Une méthode ne sert qu'à canaliser et ordonner les étapes de la modélisation. La valeur n'est pas dans la méthode mais dans les personnes qui la mettent en œuvre.

Un tel cycle de vie du processus de développement logiciel par une approche RUP (Rational Unified Process) va comprendre les étapes suivantes :

2.2.1. Identification des besoins et spécification des fonctionnalités

a. Identification et représentation des besoins : diagramme de cas d'utilisation

Les cas d'utilisation sont utilisés tout au long du projet. Dans un premier temps, on les crée pour identifier et modéliser les besoins des utilisateurs (Figure 2.1). Ces besoins sont déterminés à partir des informations recueillies lors des rencontres entre informaticiens et utilisateurs. Il faut impérativement proscrire toute considération de réalisation lors de cette étape. Durant cette étape, vous devrez déterminer les limites du système, identifier les acteurs et recenser les cas d'utilisation. Si l'application est complexe, vous pourrez organiser les cas d'utilisation en paquetages. Dans le cadre d'une approche itérative et incrémentale, il faut affecter un degré d'importance et un coefficient de risque à chacun des cas d'utilisation pour définir l'ordre des incréments à réaliser. Les interactions entre les acteurs et le système (au sein des cas d'utilisation) seront explicitées sous forme textuelle et sous forme graphique au moyen de diagrammes de séquence. Les utilisateurs ont souvent beaucoup de difficultés à exprimer clairement et précisément ce qu'ils attendent du système. L'objectif de cette étape et des deux suivantes est justement de les aider à formuler et formaliser ces besoins.

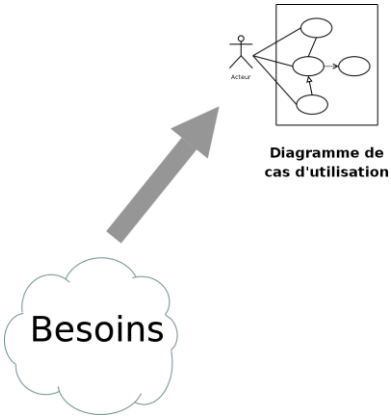

Figure 2.1. Les besoins sont modélisés par un diagramme de cas d'utilisation.

b. Spécification détaillée des besoins : diagrammes de séquence système

Dans cette étape, on cherche à détailler la description des besoins par la description textuelle des cas d'utilisation et la production de diagrammes de séquence système illustrant cette description textuelle (Figure 2.2). Cette étape amène souvent à mettre à jour le diagramme de cas d'utilisation puisque on est toujours dans la spécification des besoins. Les scénarii de la description textuelle des cas d'utilisation peuvent être vus comme des instances de cas d'utilisation et sont illustrés par des diagrammes de séquence système. Il faut, au minimum, représenter le scénario nominal de chacun des cas d'utilisation par un diagramme de séquence qui rend compte de l'interaction entre l'acteur, ou les acteurs, et le système. Le système est ici considéré comme un tout et est représenté par une ligne de vie. Chaque acteur est également associé à une ligne de vie. Lorsque les scénarii alternatifs d'un cas d'utilisation sont nombreux et importants, l'utilisation d'un diagramme d'états-transitions ou d'activités peut s'avérer préférable à une multitude de diagrammes de séquence.

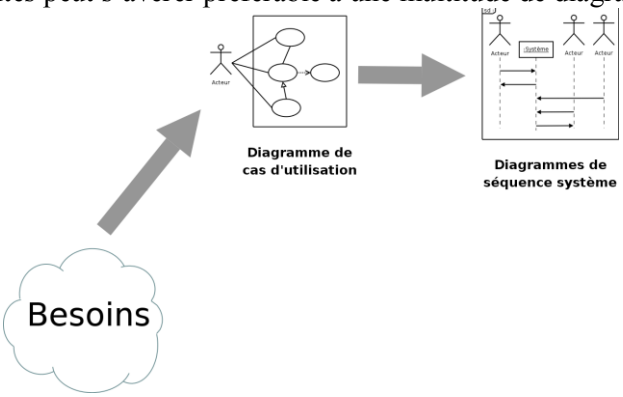

Figure 2.2. Les diagrammes de séquence système illustrent la description textuelle des cas d'utilisation.

c. Maquette de l'IHM de l'application (non couvert par UML)

Une maquette d'IHM (Interface Homme-Machine) est un produit jetable permettant aux utilisateurs d'avoir une vue concrète mais non définitive de la future interface de l'application (Figure 2.3). La maquette peut très bien consister en un ensemble de dessins produits par un logiciel de présentation ou de dessin. Par la suite, la maquette pourra intégrer des fonctionnalités de navigation permettant à l'utilisateur de tester l'enchaînement des écrans ou des menus, même si les fonctionnalités restent fictives. La maquette doit être développée rapidement afin de provoquer des retours de la part des utilisateurs.

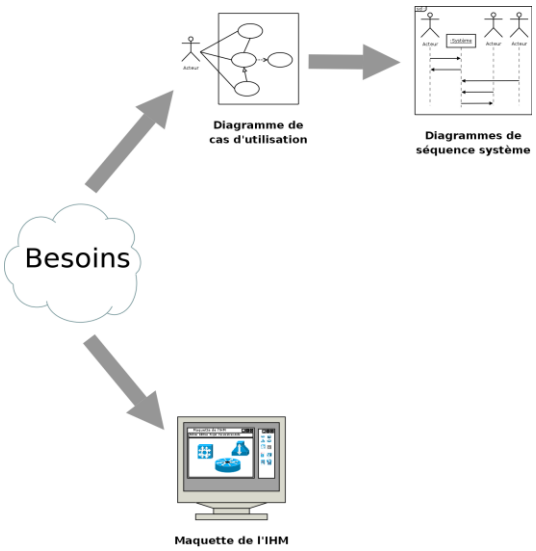

Figure 2.3. Une maquette d'IHM facilite les discussions avec les futurs utilisateurs.

2.2.2. Phases d'analyse

d. Analyse du domaine : modèle du domaine

La modélisation des besoins par des cas d'utilisation s'apparente à une analyse fonctionnelle classique. L'élaboration du modèle des classes du domaine permet d'opérer une transition vers une véritable modélisation objet. L'analyse du domaine est une étape totalement dissociée de l'analyse des besoins. Elle peut être menée avant, en parallèle ou après cette dernière. La phase d'analyse du domaine permet d'élaborer la première version du diagramme de classes (Figure 2.4) appelée modèle du domaine. Ce modèle doit définir les classes qui modélisent les entités ou concepts présents dans le domaine (on utilise aussi le terme de métier) de l'application. Il s'agit donc de produire un modèle des objets du monde réel dans un domaine donné. Ces entités ou concepts peuvent être identifiés directement à partir de la connaissance du domaine ou par des entretiens avec des experts du domaine. Il faut absolument utiliser le vocabulaire du métier pour nommer les classes et leurs attributs. Les classes du modèle du domaine ne doivent pas contenir d'opération, mais seulement des attributs. Les étapes à suivre pour établir ce diagramme sont:

– identifier les entités ou concepts du domaine ;
– identifier et ajouter les associations et les attributs ;
– organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant l'héritage ;
– le cas échéant, structurer les classes en paquetage selon les principes de cohérence et d'indépendance.

L'erreur la plus courante lors de la création d'un modèle du domaine consiste à modéliser un concept par un attribut alors que ce dernier devait être modélisé par une classe. Si la seule chose que recouvre un concept est sa valeur, il s'agit simplement d'un attribut. Par contre, si un concept recouvre un ensemble d'informations, alors il s'agit plutôt d'une classe qui possède elle-même plusieurs attributs.

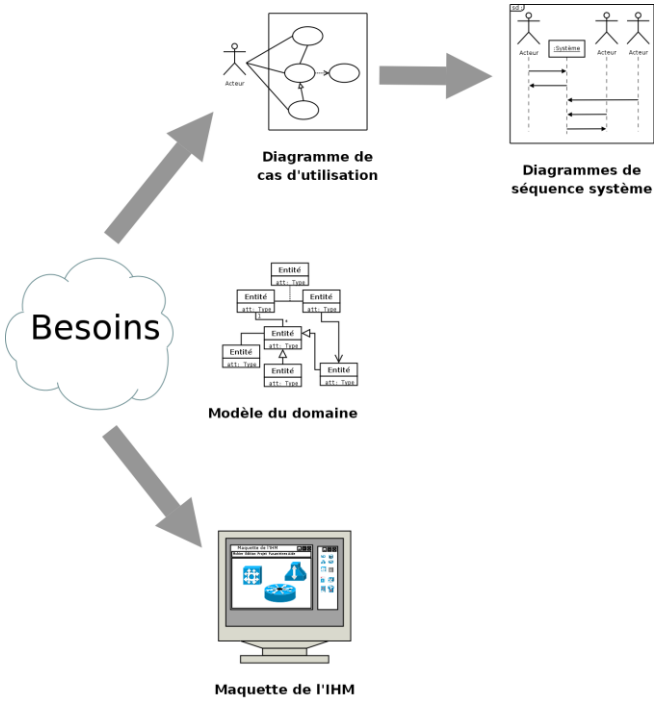

Figure 2.4. La phase d'analyse du domaine permet d'élaborer la première version du diagramme de classes.

e. Diagramme de classes participantes

Le diagramme de classes participantes est particulièrement important puisqu'il effectue la jonction entre, d'une part, les cas d'utilisation, le modèle du domaine et la maquette, et d'autre part, les diagrammes de conception logicielle que sont les diagrammes d'interaction et le diagramme de classes de conception. Les diagrammes de conception logicielle n'apparaissent pas encore sur Figure 2.5. Il n'est pas souhaitable que les utilisateurs interagissent directement avec les instances des classes du domaine par le biais de l'interface graphique. En effet, le modèle du domaine doit être indépendant des utilisateurs et de l'interface graphique. De même, l'interface graphique du logiciel doit pouvoir évoluer sans répercussion sur le cœur de l'application. C'est le principe fondamental du découpage en couches d'une application. Ainsi, le diagramme de classes participantes modélise trois types de classes d'analyse, les dialogues, les contrôles et les entités ainsi que leurs relations.

Les classes de dialogues – Les classes qui permettent les interactions entre l'IHM et les utilisateurs sont qualifiées de dialogues. Ces classes sont directement issues de l'analyse de la maquette. Il y a au moins un dialogue pour chaque association entre un acteur et un cas d'utilisation du diagramme de cas d'utilisation. En général, les dialogues vivent seulement le temps du déroulement du cas d'utilisation concerné.

Les classes de contrôles – Les classes qui modélisent la cinématique de l'application sont appelées contrôles. Elles font la jonction entre les dialogues et les classes métier en permettant aux différentes vues de l'application de manipuler des informations détenues par un ou plusieurs objets métier. Elles contiennent les règles applicatives et les isolent à la fois des dialogues et des entités.

Les classes entités – Les classes métier, qui proviennent directement du modèle du domaine, sont qualifiées d'entités. Ces classes sont généralement persistantes, c'est-à-dire qu'elles survivent à l'exécution d'un cas d'utilisation particulier et qu'elles permettent à des données et des relations d'être stockées dans des fichiers ou des bases de données. Lors de l'implémentation, ces classes peuvent ne pas se concrétiser par des classes mais par des relations, au sens des bases de données relationnelles. Lors de l'élaboration du diagramme de classes participantes, il faut veiller au respect des règles suivantes :

– Les entités, qui sont issues du modèle du domaine, ne comportent que des attributs.

– Les entités ne peuvent être en association qu'avec d'autres entités ou avec des contrôles, mais, dans ce dernier cas, avec une contrainte de navigabilité interdisant de traverser une association d'une entité vers un contrôle.

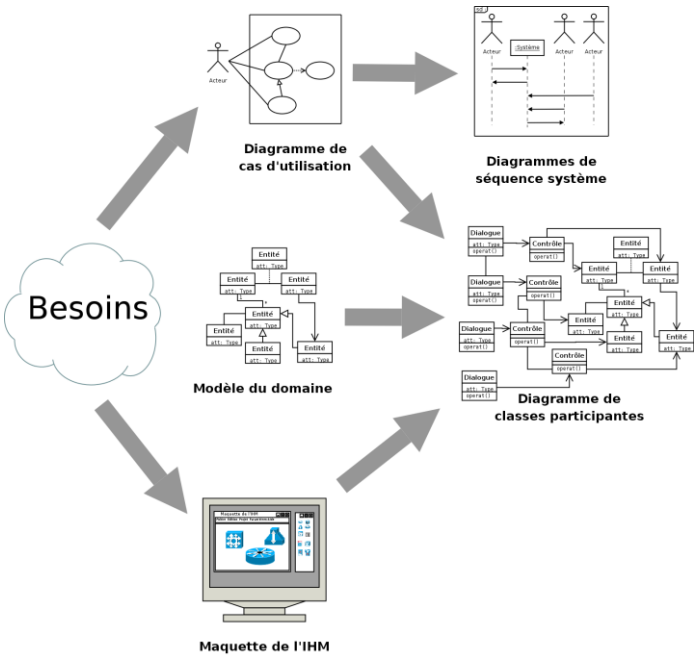

Figure 2.5. Le diagramme de classes participantes effectue la jonction entre les cas d'utilisation, le modèle du domaine et les diagrammes de conception logicielle.

– Les contrôles ne comportent que des opérations. Ils implémentent la logique applicative (i.e. les fonctionnalités de l'application), et peuvent correspondre à des règles transverses à plusieurs entités. Chaque contrôle est généralement associé à un cas d'utilisation, et vice versa. Mais rien n'empêche de décomposer un cas d'utilisation complexe en plusieurs contrôles.
– Les contrôles peuvent être associés à tous les types de classes, y compris d'autres contrôles. Dans le cas d'une association entre un dialogue et un contrôle, une contrainte de navigabilité doit interdire de traverser l'association du contrôle vers le dialogue.
– Les dialogues comportent des attributs et des opérations. Les attributs représentent des informations ou des paramètres saisis par l'utilisateur ou des résultats d'actions. Les opérations réalisent (généralement par délégation aux contrôles) les actions que l'utilisateur demande par le biais de l'IHM.
– Les dialogues peuvent être en association avec des contrôles ou d'autres dialogues, mais pas directement avec des entités.
– Il est également possible d'ajouter les acteurs sur le diagramme de classes participantes en respectant la règle suivante : un acteur ne peut être lié qu'à un dialogue.
Certaines classes possèdent un comportement dynamique complexe. Ces classes auront intérêt à être détaillées par des diagrammes d'états-transitions. L'attribution des bonnes responsabilités, aux bonnes classes est l'un des problèmes les plus délicats de la conception orientée objet. Ce problème sera affronté en phase de conception lors de l'élaboration des diagrammes d'interaction et du diagramme de classes de conception. Lors de la phase d'élaboration du diagramme de classes participantes, le chef de projet a la possibilité de découper le travail de son équipe d'analystes par cas d'utilisation. L'analyse et l'implémentation des fonctionnalités dégagées par les cas d'utilisation définissent alors les itérations à réaliser. L'ordonnancement des itérations étant défini par le degré d'importance et le coefficient de risque affecté à chacun des cas d'utilisation.

f. Diagrammes d'activités de navigation

Les IHM modernes facilitent la communication entre l'application et l'utilisateur en offrant toute une gamme de moyens d'action et de visualisation comme des menus déroulants ou contextuels, des palettes d'outils, des boîtes de dialogues, des fenêtres de visualisation, etc. Cette combinaison possible d'options d'affichage, d'interaction et de navigation aboutis aujourd'hui à des interfaces de plus en plus riches et puissantes. UML offre la possibilité de représenter graphiquement cette activité de navigation dans l'interface en produisant des diagrammes dynamiques. On appelle ces diagrammes des diagrammes de navigation. Le concepteur a le choix d'opter pour cette modélisation entre des diagrammes d'états-transitions et des diagrammes d'activités. Les diagrammes d'activités constituent peut-être un choix plus souple et plus judicieux. Les diagrammes d'activités de navigation sont à relier aux classes de dialogue du diagramme de classes participantes (Figure

2.6). Les différentes activités du diagramme de navigation peuvent être stéréotypées en fonction de leur nature : « fenêtre », « menu », « menu contextuel », « dialogue », etc. La modélisation de la navigation à intérêt à être structurée par acteur.

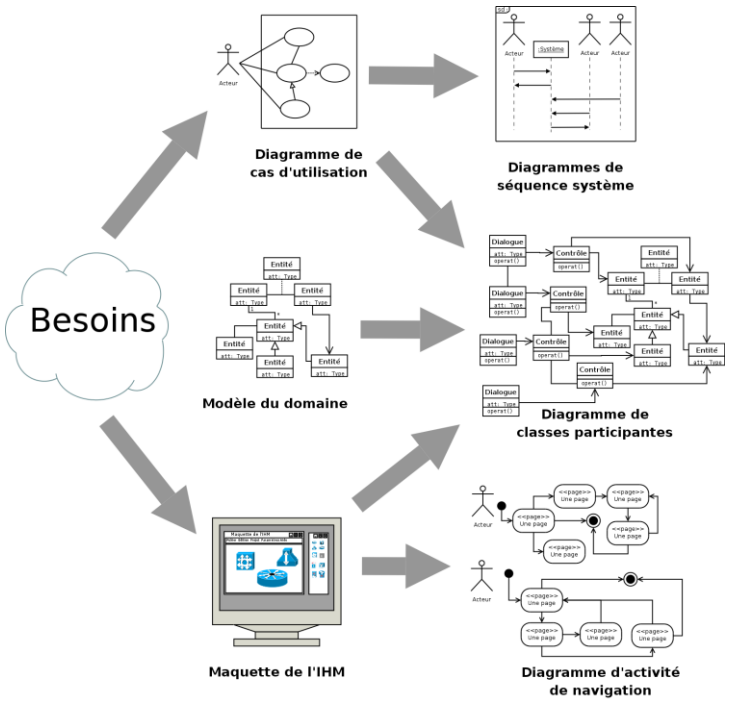

Figure 2.6. Les diagrammes d'activités de navigation représentent graphiquement l'activité de navigation dans l'IHM.

2.2.3. Phase de conception

g. Diagrammes d'interaction

Maintenant, il faut attribuer précisément les responsabilités de comportement, dégagée par le diagramme de séquence système aux classes d'analyse du diagramme classes participantes. Les résultats de cette réflexion sont présentés sous la forme de diagrammes d'interaction UML (Figure 2.7). Inversement, l'élaboration de ces diagrammes facilite grandement la réflexion. Parallèlement, une première ébauche de la vue statique de conception, c'est-à-dire du diagramme de classes de conception, est construite et complétée. Durant cette phase, l'ébauche du diagramme de classes de conception reste indépendante des choix technologiques qui seront faits ultérieurement. Pour chaque service ou fonction, il faut décider quelle est la classe qui va le contenir. Les diagrammes d'interactions (i.e de séquence ou de communication) sont particulièrement utiles au concepteur pour représenter graphiquement ces décisions d'allocations des responsabilités. Chaque diagramme va représenter un ensemble d'objets de classes différentes collaborant dans le cadre d'un scénario d'exécution du système. Dans les diagrammes d'interaction, les objets communiquent en s'envoyant des messages qui invoquent des opérations sur les objets récepteurs. Il est ainsi possible de suivre visuellement les interactions dynamiques entre objets, et les traitements réalisés par chacun d'eux. Avec un outil de modélisation UML (comme Rational Rose ou PowerAMC), la spécification de l'envoi d'un message entre deux objets crée effectivement une opération publique sur la classe de l'objet cible. Ce type d'outil permet réellement de mettre en œuvre l'allocation des responsabilités à partir des diagrammes d'interaction. Par rapport aux diagrammes de séquences système, on remplace ici le système, vu comme une boîte noire, par un ensemble d'objets en collaboration. Ces objets sont des instances des trois types de classes d'analyse du diagramme de classes participantes, à savoir des dialogues, des contrôles et des entités. Les diagrammes de séquences élaborés dans cette section doivent donc toujours respecter les règles édictées précédemment. Ces règles doivent cependant être transposées car, pour que deux objets puis interagir directement, il faut que :

– les classes dont ils sont issus soient en association dans le diagramme de classes participantes1;

– l'interaction respecte la navigabilité de l'association en question.

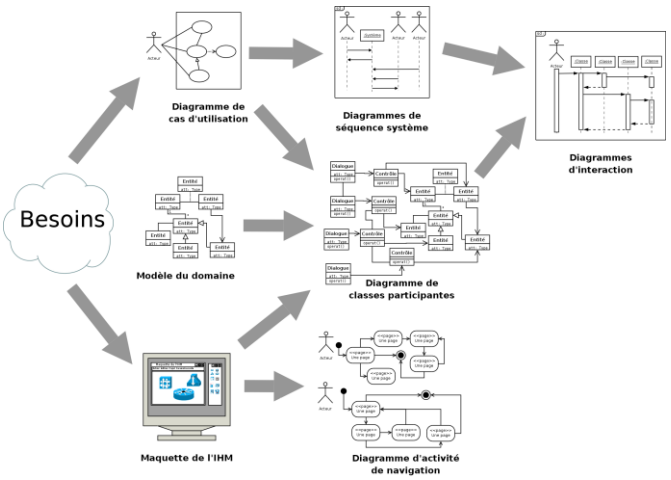

Figure 2.7. Les diagrammes d'interaction permettent d'attribuer précisément les responsabilités de comportement aux classes d'analyse.

h. Diagramme de classes de conception

L'objectif de cette étape est de produire le diagramme de classes qui servira pour l'implémentation (Figure 2.8). Une première ébauche du diagramme de classes de conception a déjà été élaborée en parallèle du diagramme d'interaction. Il faut maintenant le compléter en précisant les opérations privées des différentes classes. Il faut prendre en comptes les choix techniques, comme le choix du langage de programmation, le choix des différentes librairies utilisées (notamment pour l'implémentation de l'interface graphique), etc. Pour une classe, le couplage est la mesure de la quantité d'autres classes auxquelles elle est connectée par des associations, des relations de dépendances, etc. Durant toute l'élaboration du diagramme de classes de conception, il faut veiller à conserver un couplage faible pour obtenir une application plus évolutive et plus facile à maintenir. L'utilisation des design patterns est fortement conseillée lors de l'élaboration du diagramme de classes de conception. Parfois, les classes du type entités ont intérêt à être implémentées dans une base de données relationnelle.

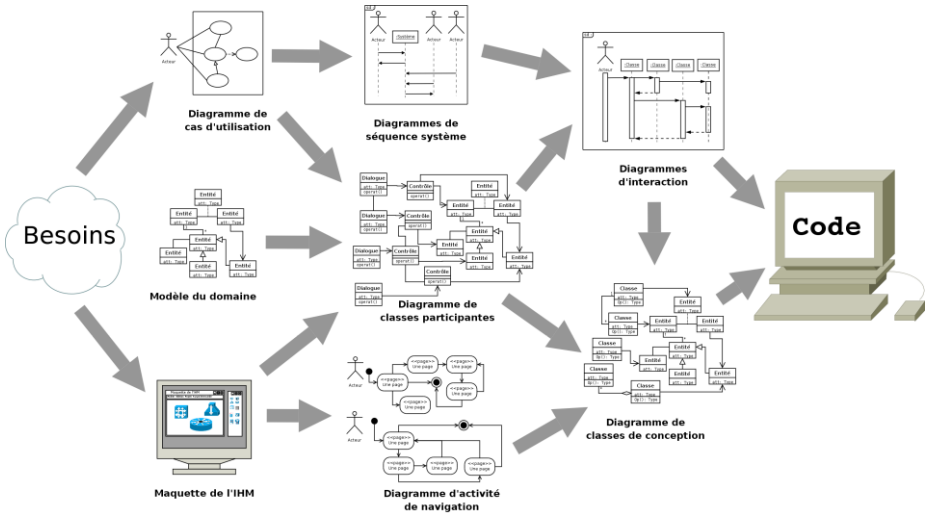

Figure 2.8. Chaîne complète de la démarche de modélisation du besoin jusqu'au code.

2.3. Les caractéristique temps réel dans UML 2.0

2.3.1. Introduction

Dans les systèmes temps réel, l'exactitude d'un calcul ne dépend pas uniquement de l'obtention des bons résultats, mais les plus importants est de les obtenir à temps. En raison des contraintes de performance, généralement la conception de logiciels temps-réel est établie conjointement avec le matériel et même le système d'exploitation. Le système d'exploitation choisi pour ces applications pourrait être un système

d'exploitation temps réel système (RTOS), spécialement conçu pour programmer des tâches temps réel. Dans les dernières années, le traitement temps réel semble être la partie essentielle d'un système d'exploitation, et l'ordonnanceur pourrait être considéré comme la composante la plus importante d'un système temps-réel. Et ainsi, la conception de ce type de systèmes peut être considérée comme l'une des tâches les plus intéressantes qu'un ingénieur logiciel peut effectuer.la modélisation orientée objet est naturellement équipée pour capturer les différentes caractéristiques et exigences des systèmes. Cependant, les techniques orientées objet actuelles ne fournissent pas les outils et les services appropriés afin de concevoir toutes les caractéristiques des systèmes temps réel, en particulier ceux concernant les contraintes (contraintes temporelles, contraintes de ressources ...) et exigences non-fonctionnelles (QoS, exigences de sécurité ...). L'UML est un standard de l'industrie qui fournit des notations pour décrire l'objet modèles orientés. Aujourd'hui, la version 2.0 d'UML est couramment utilisée comme un langage de description qui est plus complet que d'autres langages de point de vue support de modélisation de systèmes complexes, étant donné qu'il est également adapté pour modélisation de systèmes temps réel [BK03], [Dou98a], [Gom00]. UML 2.0 a été également utilisé dans un grand nombre de systèmes temps-critique et de ressources critiques [Omg05]. Dans ce qui suit, on verra ce qu'UML 2.0 a apporté comme support pour le temps réel.

2.3.2. Notion de temps

UML est un langage graphique de modélisation orientée objet qui permet de spécifier et de documenter les artefacts des systèmes logiciels. UML est largement utilisé pour exprimer le polyvalent des modèles de conception de logiciels. Les systèmes logiciels temps réel présentent par ailleurs, certaines caractéristiques spécifiques. En plus des contraintes de temps, une importante caractéristique de logiciels temps-réel provient de l'interaction avec l'environnement. Cette interaction est intrinsèquement non déterministe et fortement concurrentes. UML 2.0 présente quelques caractéristiques qui soutiennent des aspects temps réel [Ber03], [GT03]. Par exemple, il supporte la modélisation de la concurrence en fournissant certains concepts, y compris les objets actifs, les états composites concurrents et des opérations concurrentes. Afin d'exprimer contraintes de temps, UML 2.0 fournit deux types de données: **Time** et **TimeExpression**. Ces expressions temporelles peuvent être utilisées soit dans des diagrammes d'état ou de diagrammes de séquence comme le montre Figure 2.9. Par ailleurs, UML 2.0 introduit un nouveau schéma appelé Diagramme de temps (Timing Diagramme) pour permettre le raisonnement sur le temps et de visualiser les conditions ou les changements d'état au cours du temps. Figure 2.10 illustre un exemple du diagramme de temps. UML est, cependant, mieux adapté au contexte des systèmes temps réel grâce à ses mécanismes hautement extensible, en particulier les profils (on les verra dans le prochain chapitre).

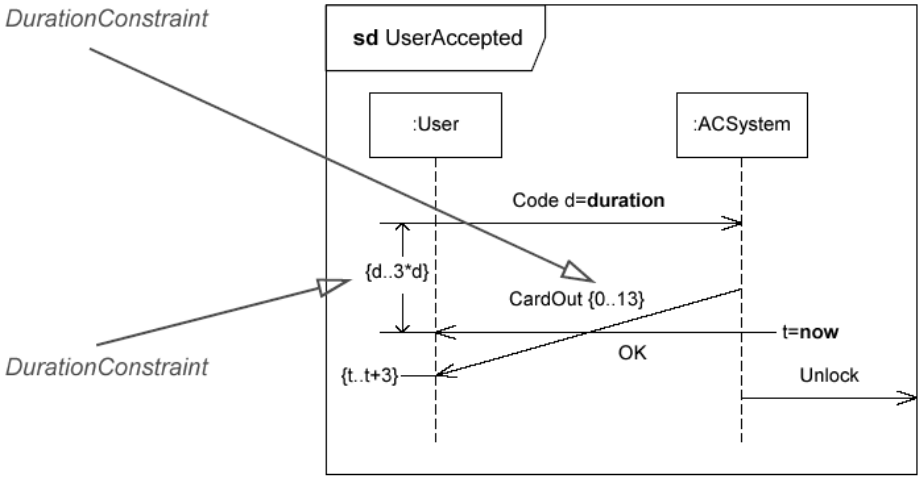

Figure 2.9. Diagramme de séquence avec des contraintes temporelles.

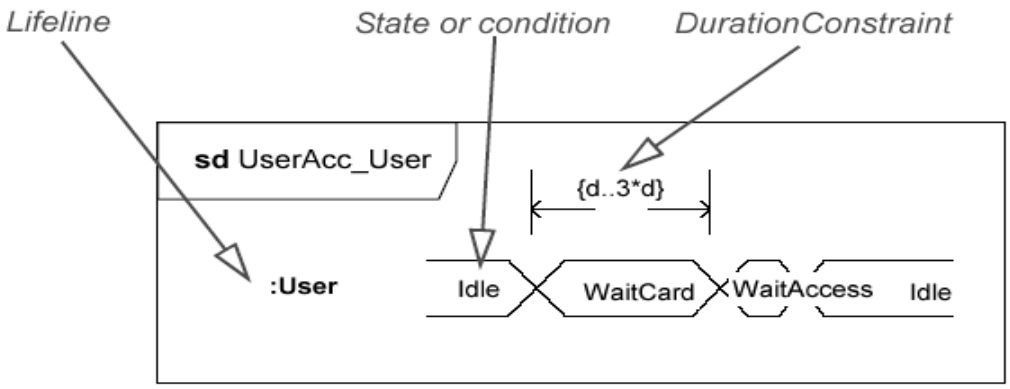

Figure 2.10. Exemple d'un diagramme de temps.

Les principaux facteurs qui affectent l'ordonnançabilité du système temps réel incluent [SR04]:

- L'occurrence de l'événement, si l'événement est périodique ou apériodique.
- Echéance d'événement, si l'échéance est inférieure, égale ou supérieure à sa période.
- Interaction des tâches, si les tâches ont besoin de se synchroniser entre eux ou non.

En raison de la nécessité de prendre en charge l'analyse d'ordonnançabilité aussi tôt que possible dans le processus de développement logiciel, son intégration avec le standard UML est une question très importante, car il pourrait permettre : de détecter les architectures temps réel irréalisables à un stade précoce du développement, d'éviter des erreurs de conception coûteuses, et de permettant aux développeurs de vérifier les performances temps réel de la conception tout au long du cycle de vie du logiciel [Mar03].

2.4. Conclusion

UML permet de fournir des modèles où des informations liées aux aspects temporels peuvent être annotées dessus. Ainsi, les développeurs logiciels peuvent être en mesure de spécifier et de concevoir leurs systèmes temps réel avec des modèles UML qui pourrait être la description formelle du système, de sorte que les différents modèles pourraient être analysés afin d'assurer qu'ils accomplissent toutes les propriétés nécessaires dans le système. Les diagrammes de temps et de séquence sont les deux types de diagramme d'interaction les plus adéquats pour modéliser l'ordonnancement des tâches. Ainsi, il est recommandé d'utiliser les deux schémas lors de l'analyse d'ordonnançabilité. Bien que le diagramme de temps ne montre pas des informations autres que ceux disponibles dans le diagramme de séquence annoté, à savoir la synchronisation absolue des évènements, le changement d'état et la synchronisation relative plus claire et plus lisible que sur le diagrammes de séquence, même lorsque les contraintes temporelles explicites sont ajoutées. Car les échéanciers délais sont d'une importance critique, par opposition à la séquence de messages, il se peut qu'il est donc préférable d'utiliser des diagrammes de temps au lieu des diagrammes de séquence. Cependant, chaque diagramme fournit des points de vue différents pour un même scénario et les deux pourraient être très utiles. Cependant, bien que les modèles résultants soient assez complets et qu'ils facilitent la communication et la compréhension, ces diagrammes ne sont pas suffisants pour indiquer si on a un ordonnancement réalisable ou non, car on aura besoin d'appliquer des modèles de faisabilité mathématiques sur les diagrammes UML. En dépit de ses capacités temps réel [Dou98b], [JMEP00], [KS01], [SR98], UML a quelques limitations ainsi, parce qu'il a des manques dans les notations et la sémantique nécessaires pour représenter plusieurs aspects qui sont d'intérêt particulier pour les développeurs de systèmes temps réel [BRS02], [LQV01]. Cependant, il dispose de mécanismes pour apporter des améliorations dans la phase de conception grâce à ses capacités d'extension, autrement dit le mécanisme de profil UML fournit un moyen de spécialisation des concepts définis dans le standard UML.

Chapitre 3 : Profils UML pour le Temps Réel

Résumé : Les systèmes temps réel (RTS) ont des contraintes temporelles strictes ainsi que des ressources limitées. La satisfaction de contraintes temporelles des RTS est nécessaire à leur correction. Pour réduire les coûts en raison de la découverte tardive de défauts de conception et / ou des violations de contraintes de temps des RTS ainsi que pour accélérer leur développement afin de faire face aux exigence du marché, il est important de valider, à un stade précoce du processus de développement, les propriétés fonctionnelles et non fonctionnelles d'un RTS. En outre, la complexité des RTS ne cesse pas d'augmenter ce qui rend leur conception très difficile. UML comme une interface graphique langage de modélisation orienté objet, est adapté pour prendre en charge cette complexité. Il support l'analyse prédictive et quantitative à travers ses profils temps réel. L'objectif de ce chapitre est d'examiner les profils les plus importants d'UML pour le temps réel.

3.1. Introduction

Aujourd'hui les systèmes temps réel sont utilisées dans une large variété d'applications, y compris avionique, l'automobile, la surveillance des patients, pour ne citer que quelques-uns. Un système en temps réel (RTS) est contraint de mener à bien ses fonctions sous contraintes temporelles (stricte). En ce qui concerne ce dernier, les RTS sont souvent classés en des systèmes temps réel durs ou souples. Dans la première classe, un délai non respecté est synonyme de désastre conséquences comme la perte de vie ou des biens. Les RTS Souples, comme un système de lecture vidéo qui tolère un certain retard d'échéance occasionnel. les RTS sont généralement intégrés dans un environnement non déterministe et fortement concurrent, par exemple, une système de contrôle de croisière à l'intérieur d'une voiture. Cet environnement génère des flux d'événements asynchrones et simultanés ayant des caractéristiques différentes, c'est à dire, périodiques, sporadiques, etc. Un RTS devrait réagir à ces événements en temps opportun. Par conséquent, les RTS doivent être simultanés afin de maximiser leur réactivité ainsi que d'utiliser efficacement leurs ressources informatiques. Par ailleurs, un RTS est généralement un système distribué qui tire parti des intérêts des réseaux temps réel tels que les bus CAN ou FDDI. Enfin, les RTS sont souvent utilisé pour surveiller et/ou contrôler des systèmes très critiques qui nécessitent de hauts niveaux de disponibilité et de fiabilité. Ces caractéristiques rendent les RTS très complexes et leur conception devient un vrai défi. Par conséquent, un langage de modélisation adéquat pour les RTS devrait permettre de gérer leur complexité inhérente ainsi que supporter l'analyse quantitative afin de valider leurs propriétés non fonctionnelles.

Le paradigme objet est très efficace pour faire face à la complexité des logiciels. L'UML qui est le langage standard orientée objet de modélisation pour les logiciels d'ordre général, attire de plus en plus l'intention de la communauté temps réel. Il est donc intéressant d'examiner comment UML est adapté au contexte temps réel. Une caractéristique importante d'UML provient de ses hauts mécanismes d'extensibilité: les stéréotypes, les valeurs marquées et les profils. Ceux-ci permettent d'adapter UML aux spécificités des domaines particuliers et de supporter des analyses spécifiques. Par conséquent, de nombreux profils UML ont été proposés dans le milieu universitaire, industriel et /ou par les organismes de normalisation afin d'accommoder les caractéristiques des logiciels temps réel. Dans ce chapitre on s'intéresse à aborder les fonctionnalités d'UML dans l'expression des exigences temps réel grâce à certains profils spécialement conçus pour ce contexte.

3.2. Profil UML pour l'ordonnançabilité, la performance et le temps (SPT)

Le profil UML pour la modélisation du temps réel, officiellement appelé le profil UML pour ordonnançabilité, Performance et Temps (UML / SPT), a été adopté par l'OMG en 2002 [Omg05]. Ceci a augmenté l'intérêt de l'utilisation de la technologie orientée objet et UML, en particulier pour modéliser et construire des systèmes temps-réel [SR04]. UML / SPT est un framework qui sert à modéliser des concepts, tels que la qualité du service (QoS), les ressources, le temps et la concurrence, ainsi de supporter l'analyse prédictive et quantitative des modèles UML. En fait, il fournit à l'utilisateur ou le modélisateur un ensemble de stéréotypes et des valeurs marquées afin d'annoter les modèles UML. L'analyse quantitative (l'ordonnançabilité et l'analyse des performances) peut ensuite être appliquée à ces modèles UML (prédictifs). La structure de l'UML / SPT, comme illustré dans Figure 3.1, est composée d'un nombre de sous-profils:

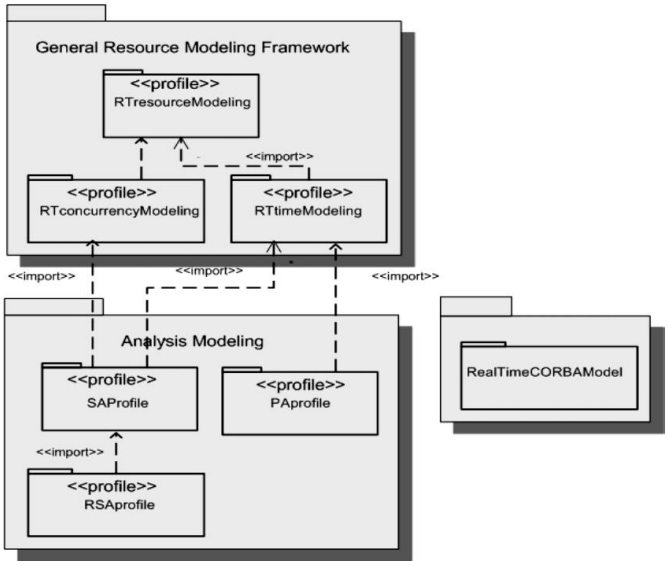

Figure 3.1. Structure du profil UML/SPT.

- Le paquetage du modèle général des ressources (GRM) est le noyau d'UML / SPT. Il est encore divisé en trois sous-paquetages:
  - RTresourceModeling pour les concepts de base de la qualité de service et des ressources.
  - RTConcurrencyModeling pour la modélisation de la concurrence.
  - RTtimeModeling pour le temps et la modélisation des mécanismes liés au temps.
- Le paquetage de modélisation d'analyse définit les sous-profils d'analyse, y compris:
  - PAprofile pour l'analyse des performances.
  - SAprofile pour l'analyse d'ordonnançabilité.

Le standard UML / SPT est trop grand pour être entièrement couvert dans cette section, par conséquent, on vient d'illustrer les caractéristiques qui sont directement liés à la modélisation du temps réel. En ce qui concerne la modélisation du temps, le sous-profil RTtimeModeling définit un méta-modèle représentant le temps, comme illustré dans Figure 3.2, et les mécanismes liés au temps, comme l'illustre Figure 3.3. Le profil fournit un ensemble de stéréotypes et des valeurs marquées que le modélisateur pourrait s'appliquer à des éléments de modélisation UML pour spécifier les valeur du temps « RTtime », les mécanismes liés au temps « RTclock », « RTtimer », « RTtimeout » ou des contraintes de temps « RTdelay », « RTintervale ». Par exemple, Figure 3.4 illustre un diagramme de séquence UML de modélisation du comportement d'un système de contrôle d'un ascenseur en réaction à l'événement du capteur d'arrivée. Ce diagramme de séquence est annoté avec des stéréotypes UML / SPT pour modéliser la périodicité de l'événement, les contraintes temporelles sur les actions des différentes composantes du système etc.

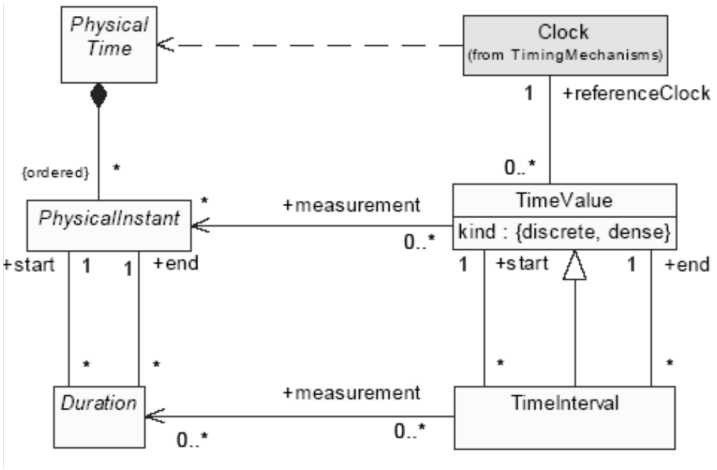

Figure 3.2. Modélisation du temps en UML/SPT[Omg05].

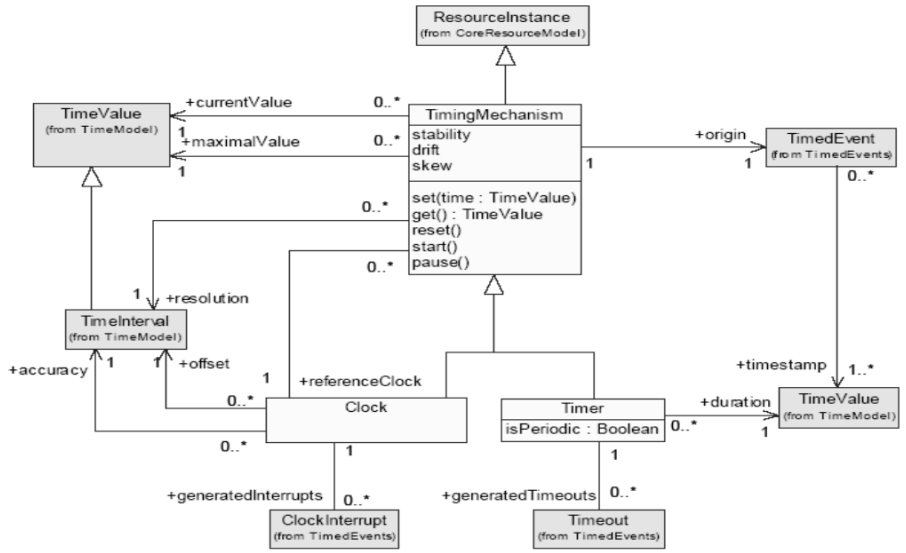

Figure 3.3. Mécanismes de synchronisation en UML/SPT[Omg05].

Le sous-profil SAprofile est conçu pour supporter l'analyse d'ordonnançabilité des modèles d'UML. Figure 3.5 montre le méta-modèle défini dans UML / SPT pour les principaux concepts impliqués dans l'analyse d'ordonnançabilité: le moteur d'exécution, les threads (tâches ou processus), les ressources partagées, les événements externes et la réponse du système à des événements extérieurs. En conséquence de ces concepts, un ensemble de stéréotypes et leur valeurs marquées est défini en UML / SPT. Un échantillon de ces derniers est présenté dans le tableau 3.1. L'application de ces stéréotypes sur un diagramme de collaboration est illustrée dans Figure 3.6.

Tableau 3.1. Stéréotypes communs d'UML/SPT pour l'analyse d'ordonnançabilité

Stéréotype	Concept temps réel	Elément de modèle UML
« SASituation »	Situation temps réel	Digramme de Collaboration et séquence
« SATrigger »	Evènement	Message
« SA Response »	Réponse	Méthode, stimulus, action
« SA Action »	Action	Méthode, stimulus, action
« SASchedulable »	Tâche, thread	Instance, objet, nœud
« SAResource »	Ressource	Instance, objet, nœud
« SAEngine »	CPU	Object, classe, nœud

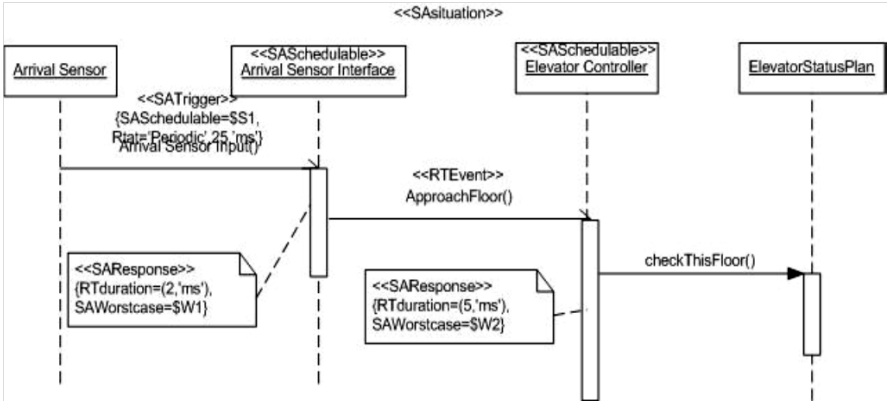

Figure 3.4. Diagramme de séquence annoté par les stéréotypes d'UML/SPT[Omg05].

La capacité d'entamer une analyse quantitative aux toutes premières phases du processus de développement logiciel est importante pour réduire le coût. En conséquence, de nombreuses activités de recherches sont dédiées à l'intégration de la modélisation conceptuelle des logiciels avec l'analyse de performance [BMIS04]. En particulier, UML / SPT est au cœur de nombreuses initiatives de recherche actives visant l'intégration de l'analyse quantitative prédictive aux toutes premières étapes du processus de développement logiciel. Ces travaux de recherche peuvent être classés en deux voies principales:

1-Le suivi de performance visant la dérivation des modèles de performance, y compris les réseaux des files d'attente [BM05], les réseaux de file d'attente en couches ou les réseaux des files d'attente étendus [XWP03], ou d'autres formalismes tels que les réseaux de Pétri stochastiques généraux. Ces modèles permettent la réalisation de l'analyse des performances sur les modèles UML.

2-La piste d'ordonnançabilité, d'autre part, comprend des initiatives de nombreuses recherches qui visent à intégrer la théorie d'ordonnancement avec une conception orienté objet temps réel [KCH01], [KCH00], [SKW00], et [SK00].

Par ailleurs, d'autres activités de recherche se concentrent sur les fondamentaux de la modélisation des concepts en UML / SPT en abordant ses capacités et ses limites [WP04].

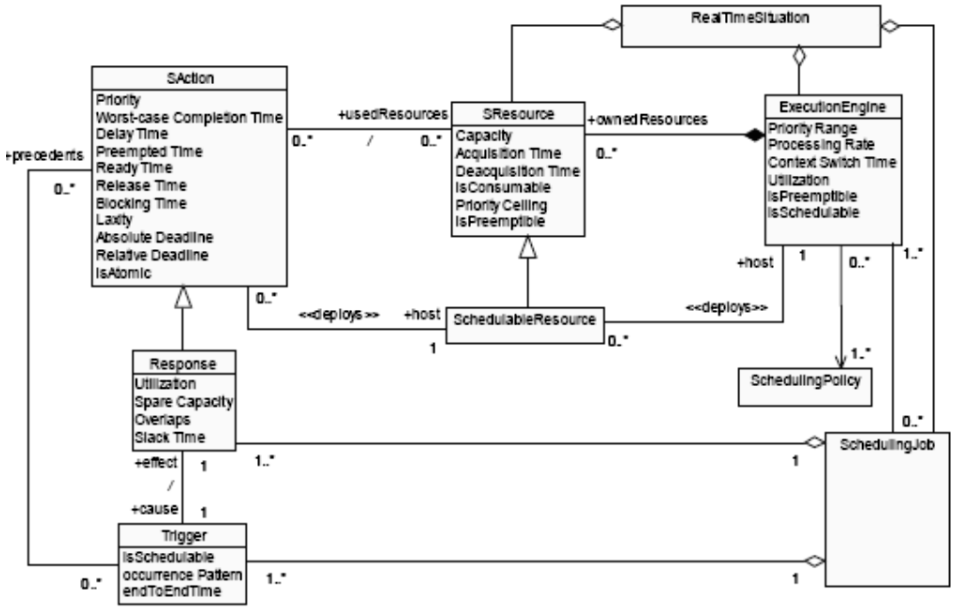

Figure 3.5. Méta-modèle d'analyse d'ordonnançabilité d'UML/SPT[Omg05].

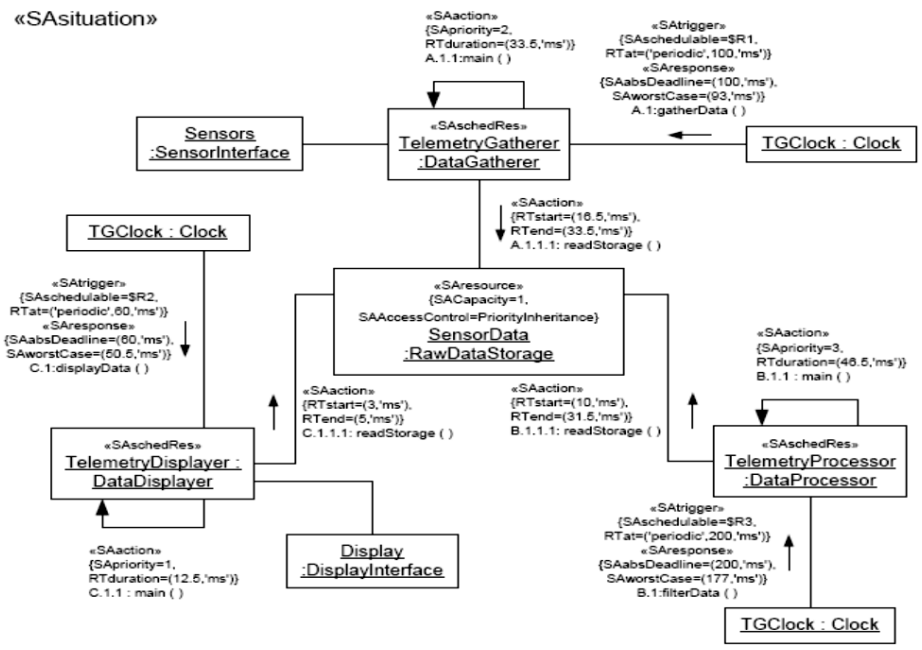

Figure 3.6. Diagramme de collaboration annoté pour l'analyse d'ordonnançabilité [Omg05].

3.3. Profil UML pour la qualité de service (QoS)

Le profil UML pour la modélisation de la qualité du service et les caractéristiques de tolérance aux pannes et les mécanismes (UML / QoS) ont été adoptés par l'OMG en 2004 [Omg04]. Il s'agit d'un profil UML émergent qui vise à capturer la notion de qualité de service au sens large. Il permet la définition d'une variété ouverte de qualités de service et de propriétés [dM03]. On considère que le profil UML / QoS est pertinent pour la modélisation de logiciels temps réel. C'est parce que il est introduit pour compléter le profil précédant UML / SPT. Cependant, tandis que UML / SPT est adaptée à la performance et l'analyse d'ordonnançabilité, UML / QoS permet au concepteur de définir un ensemble d'exigences de qualité de service et d'effectuer toute analyse spécifique qui pourrait être pertinente pour l'aspect sécurité critique des logiciels temps-réel. Cela a été démontré dans [BP04], où un modèle de qualité a été défini pour conduire une analyse de dépendance et de performance des systèmes embarquée d'automatisation. En bref, UML / QoS pourrait également être utilisé pour annoter des diagrammes UML. En revanche du UML / SPT, UML / QoS propose une procédure de trois étapes:

- La définition des caractéristiques de qualité de service: Les nouvelles caractéristiques de QoS définis par l'utilisateur pourrait tirer parti, grâce à la spécialisation, du catalogue des caractéristiques généraux de QoS définis en UML / QoS. Ce catalogue est organisé, comme le montre Figure 3.8, en les catégories suivantes : performance, fiabilité, sécurité, intégrité, Cohérence, débit, la latence, l'efficacité, la demande, la fiabilité et Disponibilité. En particulier, Figure 3.7 illustre la catégorie qui comprend les caractéristiques de la latence de QoS. Ce dernier pourrait s'adapter pour le contexte temps réel. Notons que les caractéristiques de QoS sont des classes templates ayant des paramètres. Ce dernier doit être instancié à l'étape suivante.
- Définition du modèle de qualité: Les paramètres des caractéristiques de QoS doivent être attribués à des valeurs réelles. Cela se fait à travers la définition d'une classe liée aux caractéristiques de qualité. Le modèle UML contenant les informations de liaison et les classes liées est appelé le modèle de qualité. Un exemple d'un modèle de qualité est illustré dans Figure 3.9.

- La dernière étape est l'annotation des modèles UML à l'aide des exigences de qualité de service. Ceci est illustré par Figure 3.10.

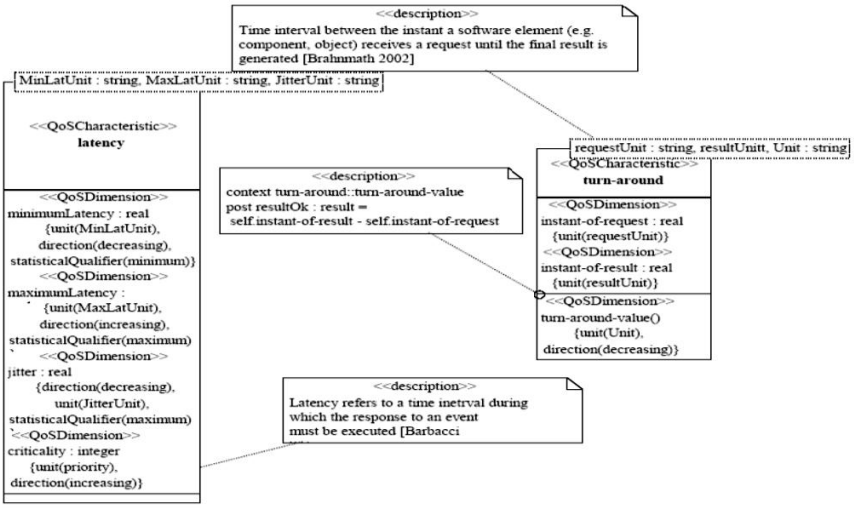

Figure 3.7. Catégorie des caractéristiques QoS de la latence[Omg04].

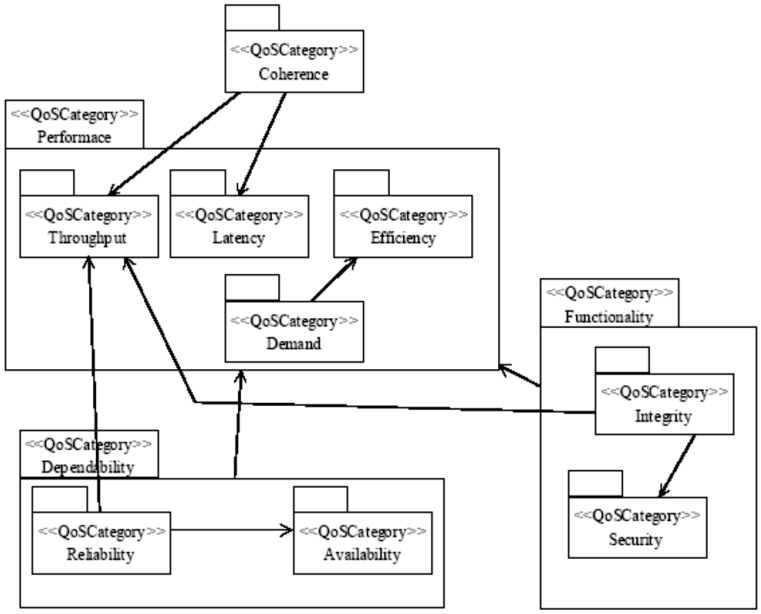

Figure 3.8. Catalogue des catégories des caractéristiques QoS [Omg04].

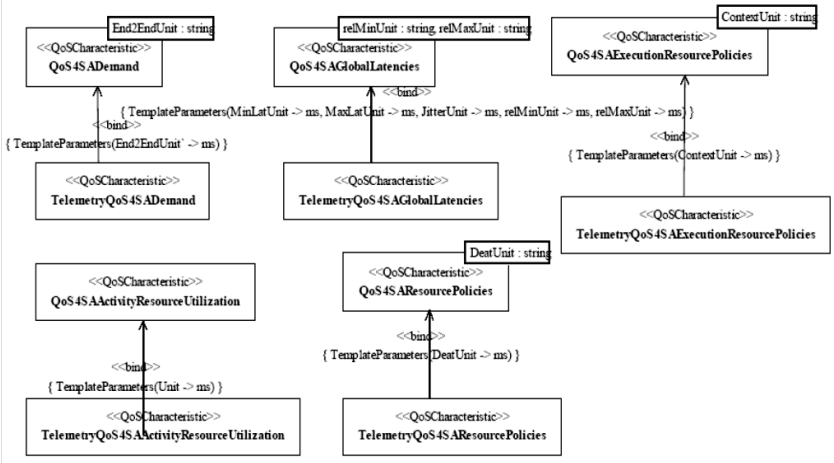

Figure 3.9. Exemple d'un modèle de qualité [Omg04].

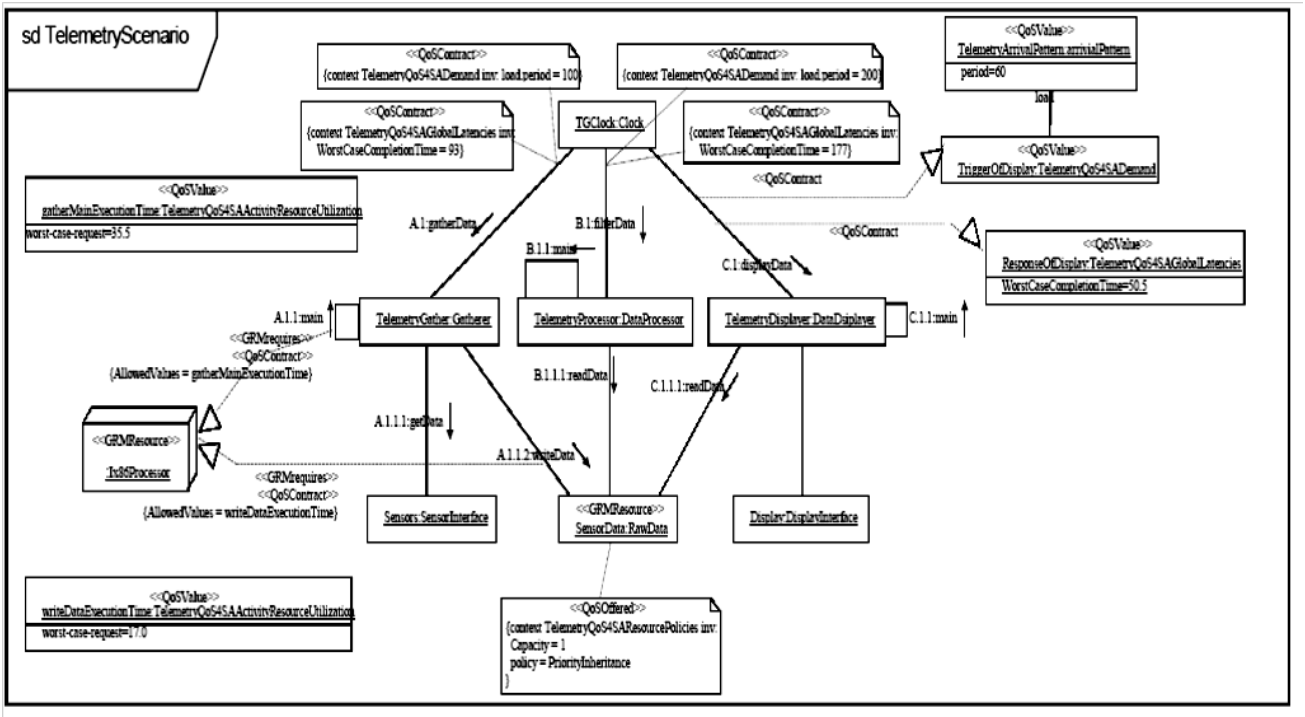

Figure 3.10. Diagramme de collaboration UML annoté par QoS [Omg04].

3.4. Profil UML-RT

UML-RT est un profil temps réel développé par Rational Software [SR98]. Il utilise les mécanismes d'extensibilité intégré d'UML pour capturer les concepts dans le langage ROOM (Real-Time Object Oriented Modeling) [SGW94]. Au contraire des deux profils précédents : UML / SPT et UML / QoS, UML-RT n'est pas uniquement destiné à annoter un modèle de conception avec des informations permettant une analyse quantitative. En effet, UML-RT permet au concepteur de produire des modèles sur des systèmes complexes, dirigés par les évènements et éventuellement systèmes temps-réel distribués. Toutefois, UML-RT ne supporte pas le temps et la modélisation des contraintes temporelles. UML-RT est pris en charge par un outil appelé RationalRT qui permet la génération de code automatique par la compilation et l'édition de liens des modèles. UML-RT comprend des constructions pour modéliser la structure et le comportement des systèmes temps réel:

- Modélisation de la structure : UML-RT fournit au concepteur des entités appelées capsules, qui sont des objets communicants actifs. Les capsules s'interagir par l'envoi et la réception des messages grâce à des interfaces appelées ports. Par ailleurs, une capsule peut avoir une structure interne composée de d'autres capsules communicantes et ainsi de suite. Cette décomposition hiérarchique permet la modélisation des systèmes complexes. Figure 3.11 illustre le modèle de structure d'un système de contrôle d'un ascenseur en utilisant UML-RT.
- Modélisation du comportement : le comportement est modélisé par un automate d'état fini étendu, et il est visualisé en utilisant des diagrammes d'état UML. Ces automates d'état sont hiérarchiques car un état peut être décomposé en d'autres automates d'état fini. La réception du message déclenche une transition dans l'automate. Les actions peuvent être associées à des transitions ou des entrées et/ou des sorties d'un état. Par exemple, Figure 3.12 illustre un modèle de comportement d'un composant contrôleur d'un système de contrôle de l'ascenseur représenté dans Figure 3.11.

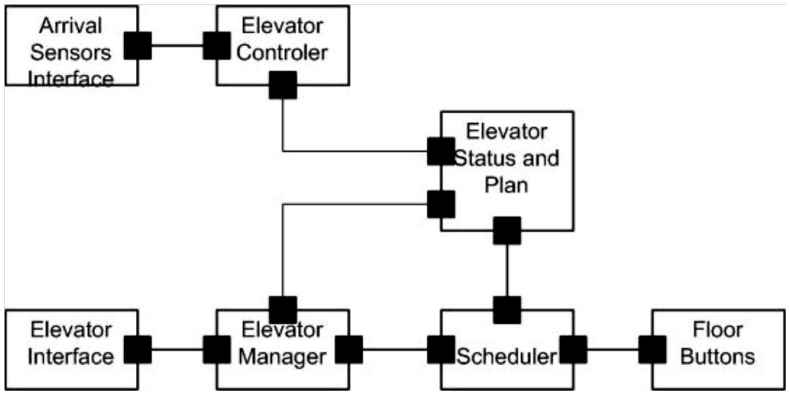

Figure 3.11. Modèle UML-RT d'un système de contrôle d'ascenseur.

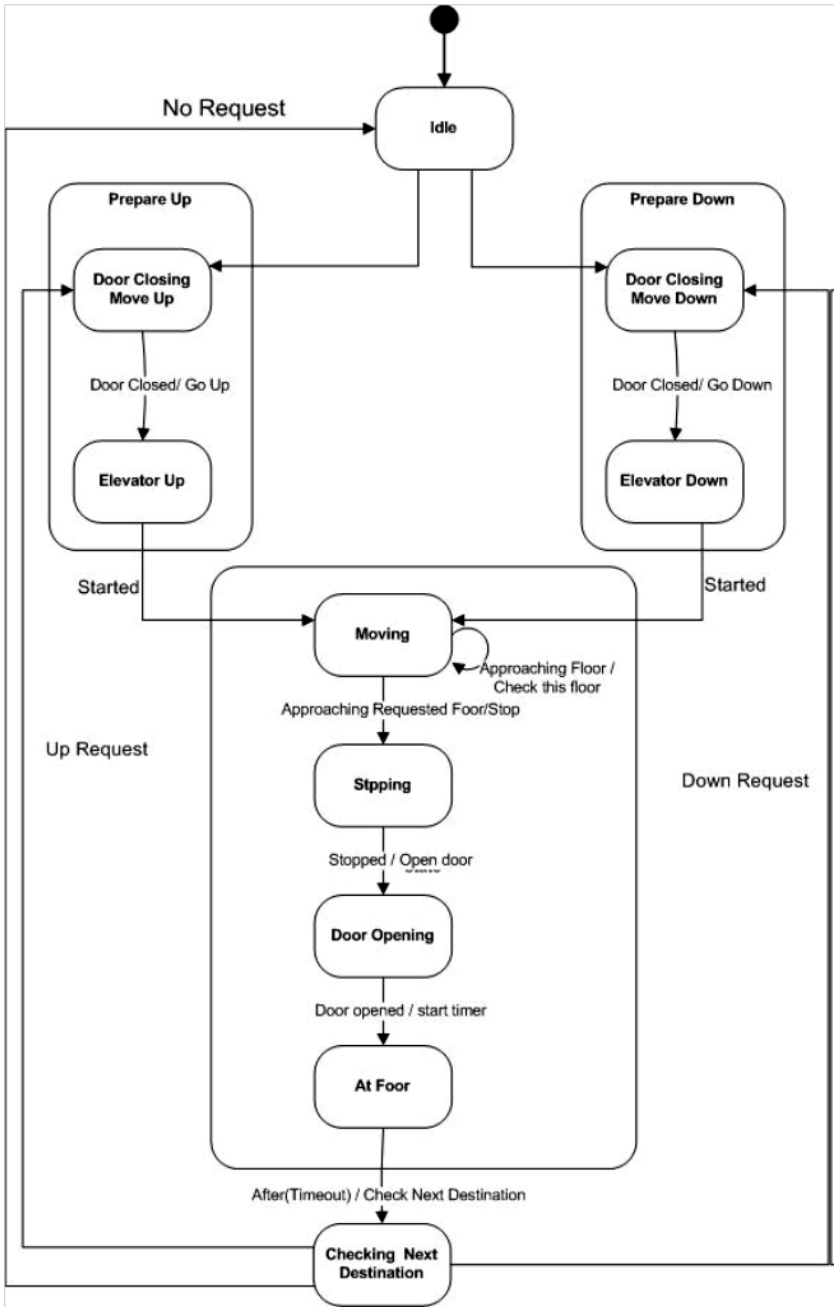

Figure 3.12. Automate d'état fini d'un contrôleur d'ascenseur.

De même pour les deux précédents profils UML, UML-RT manque de bases formelles. Il consiste cependant la base d'un travail de recherche très actif sur l'analyse d'ordonnançabilité appliqués sur des modèles de

conception de logiciels temps réel. Alors que l'outil RationalRT permet une génération automatique de code, et il ne prend pas en considération les contraintes temporelles. Par conséquent, la recherche présentée dans [SK00] a été une première tentative d'intégrer la théorie d'ordonnancement temps réel avec la conception orientée objet ciblant les systèmes temps réel. Ce travail a montré comment la théorie d'ordonnancement de priorité fixe pourrait être appliquée pour concevoir des modèles UML-RT. Enfin, une approche légèrement différente pour appliquer les techniques d'ordonnancement temps réel pour obtenir des implémentation temps réel à partir des modèles UML-RT est donnée dans [GS04], [GH05], [KWS03] et [MWS+05].

3.5. Profil TURTLE (Timed Uml RT-Lotos Environment)

C'est un profil UML visant la validation formelle des systèmes temps-réel complexes [ACLS04]. TURTLE utilise mécanismes d'extensibilité d'UML pour améliorer la structuration UML et la puissance des expressions comportementales. En outre, TURTLE a une forte base formelle et sa sémantique formelle est exprimée en RT-LOTOS. Cela permet une validation formelle ainsi que la simulation des modèles UML. Comme UML-RT, TORTLE n'annote pas les modèles UML en utilisant des stéréotypes mais il fournit des concepts et des opérateurs pour exprimer le modèle lui-même. TURTLE essentiellement permet de décrire la structure / architecture ainsi que le comportement du système en utilisant une extension des diagrammes de classes / objets et d'activités d'UML. Les principales extensions apportées par TURTLE sont les suivantes:

- Extensions structurelles: TORTLE introduit le concept de TClass qui possède des attributs spéciaux appelés Gates (voir Figure 3.13). Ils sont utilisés par les instances de TClass appelées TInstances, pour communiquer. En outre, TORTLE introduit des stéréotypes appelés les opérateurs de composition (voir Figure 3.14) qui sont utilisés pour exprimer explicitement le parallélisme, les relations de synchronisation et de séquence entre les TClasses.

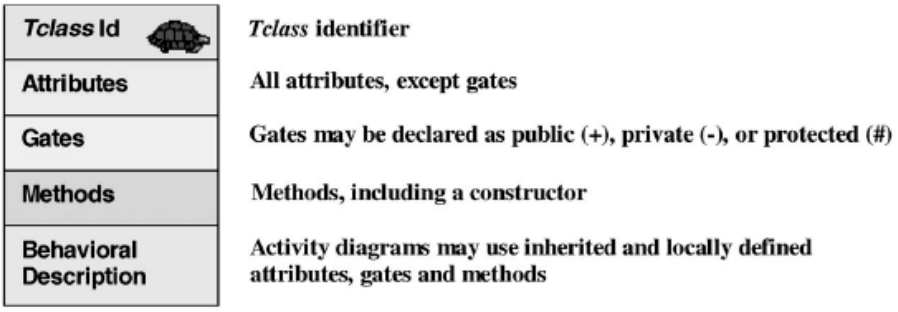

Figure 3.13. Structure TClass de TURTLE [ACLS04].

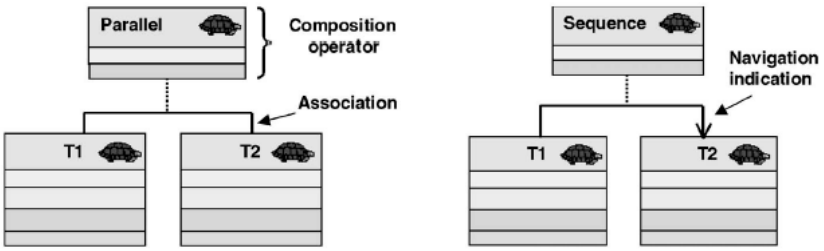

Figure 3.14. Des opérateurs de composition de TURTLE [ACLS04].

- Extensions de comportement: Le comportement d'un TClass est exprimé en utilisant les diagrammes d'activité étendus avec des opérateurs logiques et temporels (voir Figure 3.15). Ces opérateurs permettent d'exprimer la synchronisation sur les portes par des échanges de données. Par ailleurs, TURTLE permet d'exprimer le non-déterminisme temporel ainsi de différentes sortes de délais (déterministe, non déterministe).

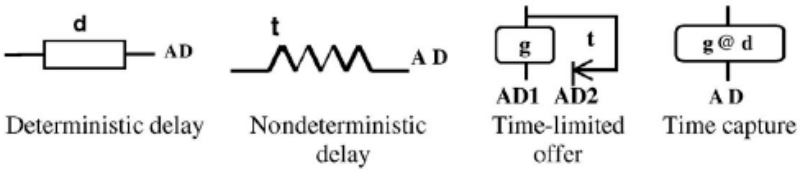

Figure 3.15. Des opérateurs temporels de TURTLE [ACLS04].

TURTLE est supporté par une boîte d'outils composée de TTool [Tto11] et RTL [Rtl98]. Ces sont utilisées par le concepteur pour construire une conception, exécuter la simulation et d'effectuer l'analyse d'accessibilité pour la validation du système. Enfin, TORTLE a été récemment étendu pour répondre aux exigences des systèmes distribués critiques. L'objectif est de permettre la définition des composants et leur déploiement, ainsi d'étudier leurs propriétés à des stades précoces du processus de développement logiciel. Ceci est fait utilisant une définition formelle des diagrammes de déploiement, qui sont les plus appropriés pour la description de l'architecture distribuée. Par conséquent, TORTLE a été étendu pour prendre en compte les diagrammes de déploiement. Le profil obtenu est appelé TURTLE-P [ASK05], qui traite de la description concrète des architectures de communication. TURTLE-P permet la validation formelle des composants et des diagrammes de déploiement grâce à ses bases formelles en RT-LOTOS.

3.6. SDL combiné avec UML

La recommandation Z.109 de ITU-T ; SDL combiné avec UML [Itu00], est en fait un profil UML pour SDL. Il définit une spécialisation d'un sous-ensemble d'UML qui est destiné à fournir au concepteur une combinaison d'UML et de SDL. Fondamentalement, Z.109 donne un modèle UML pour les principaux concepts de SDL et fournit les stéréotypes correspondants.

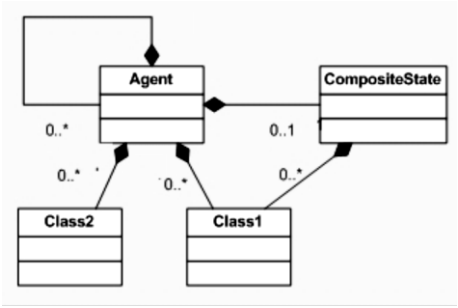

Figure 3.16. Modèle UML des concepts basiques de SDL.

Dans la suite, on met en évidence les principaux concepts de SDL ayant un stéréotype correspondant:

- **Agent :** un système SDL agent est composé d'agents connectés par des canaux. Un agent a un automate et une structure interne composée hiérarchiquement d'autres agents. Par ailleurs, un agent peut être un processus, un bloc ou un système. Z.109 fournit le modèle UML représentant ces concepts comme illustré dans Figure 3.16. En particulier, un type d'agent correspond à une classe d'objets actifs et son genre est stéréotypé en : « system », « bloc » ou « process ».
- **Gates et interface :** Les agents communiquent via des portes en envoyant des signaux ou en appelant des procédures. Ces derniers représentent leur interface. Ces interfaces correspondent simplement à des interfaces UML stéréotypées en : « signal » et « procedure ».
- **Automate :** un automate SDL correspond à un automate UML.
- **Paquetage :** un paquetage SDL correspond à un paquetage UML.

Enfin, ce profil a été mis en œuvre dans l'outil de Telelogic CASE tool Telelogic TAU 3.5 [Tel11].

3.7. Autre profils

Autre que les profils mentionnés en haut, il y en a d'autres proposés du domaine académique:

- OMEGA-RT [GOO03] qui est Un profil compatible UML/SPT qui constitue une partie du projet OMEGA [Ome11]. C'est un framework de modélisation temps réel basé UML permettant l'analyse, la vérification du temps ainsi qu'il gère aussi les aspects d'ordonnancement, en particulier, il fournit un ensemble de primitives d'évènements temporisés exprimés à l'aide d'automates temporisés.
- Profil OCL[FM02] se base sur une extension du méta-modèle OCL 2.0 pour la spécification des contraintes temps réel utilisant la spécification OCL. La sémantique formelle est basée sur les formules logiques temporelles exprimées en CTL, ce qui permet la vérification formelle des propriétés.

3.8. Discussion

Dans cette section, on compare entre les profils en fonction des critères susmentionnés, incluant la base formelle, l'expressivité et le support d'outil. On peut distinguer, deux groupes de profils: le premier groupe comprend UML / SPT, UML / QoS et UML-RT. Ces profils manque de rigueur, et de base formelle. En particulier pour les UML / SPT, cette limitation importante a été souligné dans [GO04]. Le deuxième groupe de profils présente quelques sémantiques formelles telles que RT-LOTOS pour TURTLE et SDL pour Z.109. Quant à l'expressivité des profils, on constate que les profils de l'OMG, à savoir UML / SPT et UML / QoS, introduisent des modèles pour les concepts de temps, de ressources et les caractéristiques de la qualité de service respectivement en utilisant les stéréotypes pour appliquer des annotations aux modèles de conception UML, tandis que les autres; UML-RT, TORTLE et Z.109, n'expriment pas explicitement les contraintes temporelles, mais ils introduisent de nouveaux éléments et constructions de modélisation tels les capsules dans UML-RT, et TClass dans TURTLE pour construire le modèle lui-même. L'objectif ultime des profils UML pour le temps réel est d'intégrer UML dans un framework permettant la conception, l'analyse et la synthèse des logiciels temps-réel. Cette initiative vise à réduire le coût et à s'adapter aux exigences du temps de mise en marché. Alors qu'il existe déjà un certain support d'outil pour la majorité des profils ci-dessus, un framework intégré complet cependant n'a pas encore vu le jour. Cela est particulièrement vrai pour le standard UML / SPT de l'OMG, et ceci peut être du à un problème méthodologique dans la spécification du standard, parce qu'il n'est mentionné comment utiliser ce profil. L'approche de modélisation multi-vue d'UML, où les différents schémas sont utilisés pour capturer des exigences différentes du système, ainsi que l'aspect transversal des annotations UML / SPT à travers le modèle UML peut conduire à un problème de cohérence grave. En outre, un moyen efficace et correcte de correspondance des annotations UML / SPT à un modèle approprié pour l'analyse d'ordonnançabilité manque encore. Le tableau 3.2 résume la comparaison des profils UML pour les différents systèmes temps réel.

Tableau 3.2. Profils UML pour les systèmes temps réel

Profil	propriétaire	Support d'outil	système	Analyse	Expressivité	Sémantiques formelles
UML/SPT	OMG	Rhapsody (iLogix) RT Studio (Artisan)	Systèmes TR génériques	Performance Ordonnançabilité	Temps Ressource Concurrence	/
UML/QoS	OMG	/	Systèmes TR génériques	Définie par l'utilisateur	Caractéristiques QoS	/
UML/RT	IBM/Rational	RationalRT	Systèmes TR génériques	Ordonnançabilité	Capsule Port Messages Async	/
UML/TURTLE	Académie	TTool RTL	Systèmes distribués critiques	/	Opérateur de synchronisation, de parallélisme et de délai	RT-LOTOS
Z.109	ITU-T (Industrie)	Telelogic Tau 3.5	Systèmes critiques de télécom-munication	/	Concepts SDL	SDL
OMEGA-RT	Académie	OMEGA Tool set	Systèmes TR	Synchronisation Ordonnancement	Evénements de temporisation	Automates temporisés
Profil OCL	Académie	/	Systèmes TR génériques	Vérification	Contraintes temps réel	CTL

3.9. Conclusion

Avec la constante augmentation de la complexité des systèmes temps réel actuels, il s'avère crucial d'utiliser un langage de modélisation adéquat. UML est devenu un langage de modélisation graphique et orienté objet largement utilisé et ceci surtout pour le développement des logiciel d'ordre général. Dans ce chapitre, on a vu comment UML s'est adapté pour répondre aux exigences particulières des systèmes temps réel liées principalement au propriétés non-fonctionnelles de ces systèmes, et ceci grâce au profil d'UML. notons que le UML/SPT d'OMG a attiré beaucoup d'attention de beaucoup d'activités de recherche visant à intégrer l'analyse quantitative, la performance, et l'analyse d'ordonnançabilité dans les étapes précoces du processus de développement logiciel. Pendant que l'analyse de performance basée modèles a conduit à des résultats intéressants, l'analyse d'ordonnançabilité et son intégration dans un framework basé sur UML/SPT reste toujours délicate.

Chapitre 4 : Profil UML pour MARTE (Modeling and Analysis of Real-Time and Embedded Systems)

Résumé : Depuis l'adoption du standard UML, notamment sous sa deuxième version, ce langage de modélisation a été très largement testé industriellement pour le développement de systèmes embarqués temps réel (SETR). Grâce à cette expérience, UML apparaît aujourd'hui comme un langage de modélisation très utile, couvrant de multiples besoins, toutefois il ne permet pas de répondre à toutes les spécificités métiers liées au développement de systèmes temps réels embarqués. Le manque d'artefacts pour la quantification du temps, pour la modélisation des ressources d'exécution critiques et concurrentes, et pour la description rigoureuse d'une sémantique d'exécution empêchait jusqu'à maintenant sa plus large utilisation dans ces domaines. Ceux-ci nécessitent des langages couvrant aussi bien les besoins liés à la conception que ceux liés à l'analyse des systèmes. Pour répondre à ces exigences, le consortium OMG a d'ores et déjà normalisé des extensions à son langage de modélisation universel UML : l'extension SPT (Schedulability, Performance and Time) et l'extension QoS&FIT (Modeling Quality of Service and Fault-Tolerance Characteristics & Machanisms). Ces extensions ne couvrant pas tous les besoins d'un développement dirigé par les modèles, l'OMG a récemment émis un nouvel appel à propositions pour une extension successeur de UML/SPT, nommée MARTE (UML Profile for **M**odeling and **A**nalysis of **R**eal-**T**ime and **E**mbedded systems) et en conséquence le consortium ProMarte a répondu à cet appel. Cette récente extension a été adoptée le 29 juin 2007.

4.1. Motivation

UML est un langage ou un standard de modélisation d'ordre général qui peut être spécialisé ou étendu pour traiter des domaines ou des besoins spécifiques. Ces extensions là s'appellent les profils

> *Un profil d'UML fournit un mécanisme d'extension générique pour la personnalisation des modèles UML pour des domaines particuliers. Le mécanisme d'extension permettra d'affiner la sémantique standard de manière strictement additive, de sorte qu'ils ne peuvent pas contredire la sémantique standard.*
>
> *Les profils sont définis en utilisant les stéréotypes, les définitions des balises et les contraintes qui sont appliquées aux éléments spécifiques des modèles, tels que les classes, les attributs, les opérations et les activités. Un profil est une collection d'extensions telles qui personnalise UML pour un domaine particulier (par exemple, de l'aérospatiale, la santé, le financiers) ou une plate-forme (J2EE, .NET).*

Le domaine des systèmes logiciels temps-réel et embarqué est un domaine où les extensions UML sont tenus de fournir une expression aussi précise que les phénomènes spécifiques à ce domaine (par exemple, les mécanismes d'exclusion mutuelle, la concurrence, les spécifications d'échéances, etc.)

Auparavant l'OMG a déjà adopté un profil UML à cet effet, appelé le profil UML pour l'ordonnançabilité, la Performance et le Temps « Schedulability, Performance and Time» (SPT). Il a fourni des concepts pour faire face à l'analyse d'ordonnançabilité basée modèles, axé principalement sur l'analyse du taux monotone, et aussi

des concepts pour l'analyse de performance basé modèles qui se sont fondés sur la théorie d'attente. En outre, les SPT a également fourni un cadre pour représenter le temps et ses mécanismes. Toutefois, l'expérience pratique avec SPT a révélé des lacunes dans le profil en termes de sa puissance expressive et sa flexibilité.

Par ailleurs, lorsque la nouvelle version largement révisée d'UML qui est l'UML2, a été adopté par l'OMG, il est devenu nécessaire de mettre à niveau le profil SPT. Par conséquent, une nouvelle demande de propositions a été émise par l'OMG en quête d'un nouveau profil UML pour les systèmes temps-réel et embarqués. Ce profil a été nommé MARTE (l'abrégée de «Modélisation et analyse des systèmes temps-réel et embarqués»).

L'objectif était de combler les lacunes d'SPT, ainsi de fournir un alignement avec d'autres profils OMG normalisées qui permettent la spécification non seulement de contraintes de temps réel mais également d'autres caractéristiques liées aux systèmes embarqués, telles que la capacité de mémoire et la consommation d'énergie. MARTE est également nécessaire pour supporter la modélisation et l'analyse des architectures basées composants, ainsi qu'une variété de paradigmes (asynchrones, synchrones, et chronométré).

En réponse à cette demande de proposition, certain membres de l'organisation OMG a collaboré à une seule soumission conjointe. Après plusieurs années d'activité intense, ce groupe, appelé le consortium ProMARTE, est arrivés pour obtenir la version officielle du standard MARTE 1.0 par l'OMG.

4.2. Introduction

Le profil UML pour MARTE rajoute de nouvelles capacités à UML pour le développement dirigé par les modèles pour les systèmes Temps Réel et Embarqués (STRE). Cette extension, appelée le profil UML pour MARTE (MARTE tout court), fournit un support pour les phases de spécification, de conception et de vérification / validation. Ce nouveau profil est destiné à remplacer le profil existant UML pour l'ordonnançabilité, Performance et Temps.

MARTE consiste à définir des concepts de base pour les descriptions dirigées par les modèles des systèmes temps réel et embarqués. Ces concepts de base sont ensuite affinées à la fois pour la modélisation et l'analyse des besoins. Les phases de modélisation fournissent un soutien nécessaire dès la spécification jusqu'à la conception détaillée des caractéristiques temps réel et embarqués des systèmes. MARTE traite également l'analyse dirigée par les modèles. En ce sens, le but n'est plus de définir de nouvelles techniques d'analyse des systèmes temps-réel et embarqués, mais pour les supporter. Ainsi, il fournit des moyens pour annoter les modèles avec des informations nécessaires pour effectuer des analyses spécifiques. Ainsi MARTE se concentre sur la performance et l'analyse d'ordonnançabilité, mais aussi il définit un cadre général pour l'analyse quantitative, qui vise à affiner / spécialiser n'importe quel autre type d'analyse.

On cite, entre autre, les avantages apporter par ce nouveau profil :

- Il fournir une approche commune de la modélisation des aspects matériels et logiciels d'un STRE en vue d'améliorer la communication entre les développeurs.
- Il permet l'interopérabilité entre les outils de développement utilisés pour la génération de la spécification, la conception, la vérification, le code, etc.
- Il favoriser la construction de modèles qui peuvent être utilisés pour faire des prévisions quantitatives concernant les caractéristiques temps réel et embarqués des systèmes en tenant compte à la fois des caractéristiques matérielles et logicielles.

4.3. Architecture

MARTE est structuré autour de deux exigences principales, l'une est la modélisation des caractéristiques des systèmes temps-réel et embarqués et l'autre est l'annotation des modèles d'application afin de supporter l'analyse des propriétés du système. L'architecture MARTE est représentée par les trois principaux diagrammes de paquetage : le **paquetage fondation**, le **paquetage du modèle de conception** et le **paquetage du modèle d'analyse,** comme montré dans Figure 4.1. Un quatrième paquetage optionnel contient les profils définis dans les annexes MARTE, ainsi qu'un modèle des bibliothèques prédéfinies qui peuvent être utilisées par les modélisateurs pour leurs applications temps réel et embarqués.

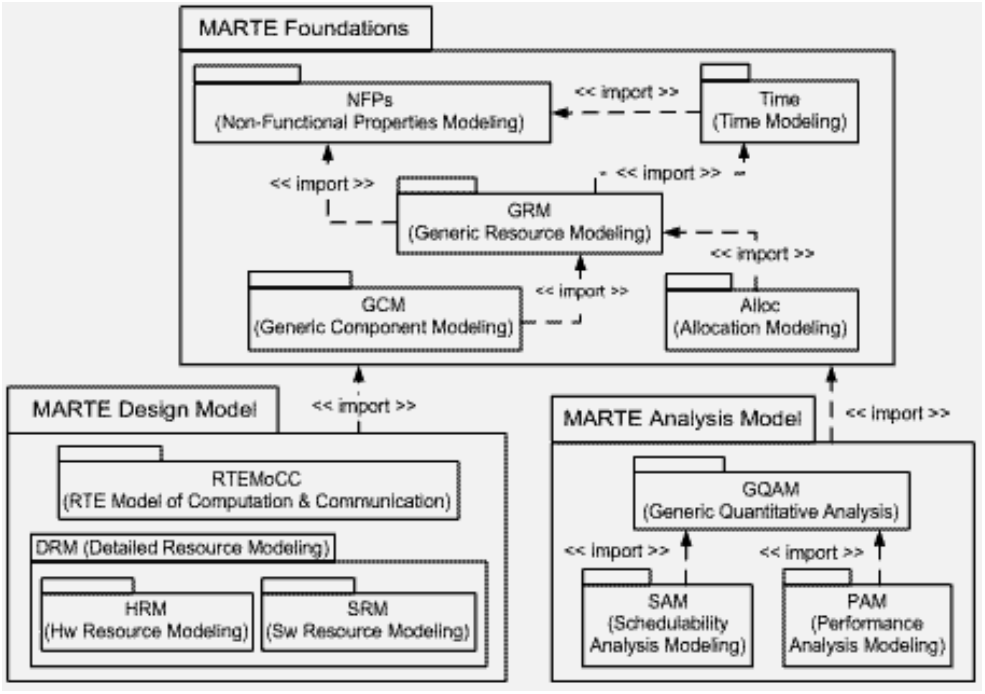

Figure 4.1. Architecture du profil MARTE. [Omg09]

4.3.1. Paquetage Fondation de MARTE

Ce package définit tous les concepts de base nécessaires à la conception et l'analyse des systèmes temps-réel et embarqués. Il fournit aux développeurs de modèles les moyens pour modéliser les éléments suivants:

- **Propriétés non fonctionnelles (NFPs):** Ce sous-paquetage offre des moyens pour spécifier les propriétés non-fonctionnelles des systèmes temps réel comme l'utilisation de mémoire, la consommation d'énergie, etc. Il explique également comment ces NFPs peuvent être attachés aux éléments de modèle. En bref, ce profil spécifie un cadre général permettant d'annoter les éléments du modèle UML avec les NFPs.
- **Modélisation du temps:** Ce sous-paquetage permet la modélisation du temps et des structures liées au temps. L'objectif principal de MARTE est de fournir des concepts de base et avancés de modélisation du temps. Comme un profil successeur du profil SPT, le profil MARTE doit supporter un temps métrique avec une référence implicite au temps physique [AMS07]. Toutefois, MARTE supporte les modèles de temps de général comme «physique», «logique» et « multiforme ». Ces concepts avancés liés au temps, pourraient être utilisées pour développer les différents modèles de calcul et de communication (MoCC).
- **Modélisation des ressources génériques (GRM):** Ce sous-paquetage offre tous les stéréotypes et les valeurs marquées nécessaires pour représenter des ressources comme les médiums de communication,

les ressources de calcul, les ressources de stockage etc. En outre, il inclut des fonctionnalités qui sont nécessaires pour faire face à la modélisation des plates-formes d'exécution avec différents niveaux d'abstraction et de modélisation matérielle (comme exemple, les canaux de communication physiques) et logicielle (à titre d'exemple, les systèmes d'exploitation temps réel) des plates-formes. Le sous-paquetage GRM avec le sous-paquetage de modélisation du temps peut être utilisés pour spécifier des contraintes de temps, et lorsqu'il est utilisé avec le sous-paquetage NFPs il peut spécifier la qualité de service.

- **Modèle de composant générique (GCM):** Ce sous-paquetage est utile pour appliquer des stratégies de composants de base dans le domaine des systèmes temps-réel et embarqués. Le sous-paquetage GCM n'est rien, mais les raffinements appliqués aux classes UML structurées au dessus desquelles un support des bloques SysML a été ajouté. Ainsi, ce modèle fournit un dénominateur commun pour les différents modèles de calcul et de communication (MoCCs) [Omg07]. Ce sous-package permet de fournir un modèle aussi général que possible, qui est indépendant d'une sémantique d'exécution spécifique, sur lequel les caractéristiques temps réel peuvent être appliquées ultérieurement.
- **Modélisation des allocations (Alloc):** Le profil MARTE permet aux concepteurs de modéliser à la fois les applications et les plateformes d'exécution. Un élément d'application dans MARTE pourrait être un service, un calcul ou une fonction d'un système d'exploitation temps réel (RTOS). Une plate-forme d'exécution est un ensemble de ressources reliées représentant l'architecture matérielle. Il se compose de <<HW_ Resource>> tels que des ressources de stockage (RAM, ROM ou cache), les moyens de communication (bus et d'E / S) et des ressources informatiques (processeur, accélérateur matériel), etc. Après la modélisation de l'architecture logicielle et matérielle, il est important de répartir les tâches de l'application sur la plateforme d'exécution dans les applications temps réel embarquées. La répartition joue un rôle important quant à la performance. MARTE fournit un mécanisme appelé «allocation», qui permet aux concepteurs de spécifier la répartition. Une allocation MARTE est une association entre une application et une plateforme MARTE d'exécution. Le concept principal de l'allocation <<Allocate>> est utilisé pour associer des éléments d'un contexte logique, les éléments du modèle d'application, ou les éléments du modèle d'application, à des éléments nommés décrits dans un contexte plus physique, ou à des éléments du modèle de la plateforme d'exécution [BMP 07] respectivement.

Les concepts définis en utilisant ce paquetage sont ensuite affinées dans les deux paquetages suivants pour supporter les exigences de modélisation et d'analyse des systèmes temps réel.

4.3.2. Paquetage du modèle de conception de MARTE

Ce paquetage établit la conception basée modèles depuis l'analyse des besoins jusqu'à la spécification, la conception et l'implémentation. Il fournit des concepts de haut niveau pour la modélisation des caractéristiques à la fois quantitatives et qualitatives des systèmes temps réel/protocoles. En outre, elle fournit également des moyens pour une description détaillée des ressources logicielles et matérielles utilisées pour l'exécution d'une application.

- **Modèle RTE de calcul et de Communication (RTEMoCC):** L'objectif principal de ce sous-paquetage est de fournir des concepts de modélisation de haut niveau, qui permettent les fonctionnalités de modélisation quantitative comme l'échéance et la période, ainsi que les caractéristiques qualitatives qui sont liées au comportement, comme la communication et la concurrence.
- **Modélisation détaillée des ressources (DRM):** Ce sous-paquetage se spécialise aux concepts génériques fournis par le sous-paquetage GRM. Il offre des artefacts de modélisation spécifiques pour spécifier à la fois les supports d'exécution logiciels et matériels. Le sous-paquetage DRM se compose

de deux sous-paquetages : celui de modélisation de ressources logicielles (SRM) et celui de modélisation des ressources matérielles (HRM).

- **Modélisation des ressources logicielles (SRM):** Ce sous-paquetage spécifie un ensemble d'artefacts de modélisation qui facilite la description de la structure des supports d'exécution comme le RTOS (système d'exploitation temps réel). Un RTOS est utilisé comme support d'exécution dans les approches multitâches. Ces dernières sont utilisées pour la conception largement répandue de logiciels d'applications temps réel. Le sous-paquetage SRM n'est pas dédié à la définition d'une nouvelle API (Application Programming Interface) pour RTOS plutôt elle offre des artefacts pour modéliser les bibliothèques de l'API RTOS.
- **Modélisation des ressources matérielles (HRM):** Ce sous-paquetage est utilisé pour spécifier les éléments d'architecture détaillée des plateformes, c'est-à-dire, que son but est de décrire les supports d'exécution matériels avec différents niveaux de détails et de vues car ils sont essentiels pour répondre aux spécifications d'application. Le sous-paquetage HRM sert à la description du matériel existant et la conception de nouvelles plateformes matérielles. Le HRM se compose de deux vues, une vue logique et une vue physique, ces deux vues sont complémentaires l'une à l'autre. Elles fournissent l'abstraction du matériel de deux façons différentes qui pourraient être tout simplement fusionnées. La vue physique classe les ressources matérielles basant sur les propriétés physiques d'une part et d'autre part, la vue logique classifie les ressources matérielles basant sur les propriétés fonctionnelles [Omg07].

Une fois la modélisation des ressources matérielles et la modélisation des ressources logicielles soient terminées, elles seront combinées pour supporter l'exécution entière de l'application.

4.3.3. Paquetage du modèle d'analyse de MARTE

Ce paquetage offre des abstractions spécifiques et des annotations pertinentes qui pourraient être lues par des outils d'analyse. Ce paquetage est destiné à fournir des évaluations fiables et précises en utilisant des analyses quantitatives formelles fondées sur des modèles mathématiques [Omg07]. Ce paquetage est divisé en trois sous-paquetages, à savoir:

- **Modélisation d'analyse quantitative générique (GQAM):** Ce sous-package fournit des concepts génériques pour l'analyse quantitative. La modélisation générique fournit des concepts de modélisation de base et les NFPs basée sur le cadre d'annotation NFPs, ce qui simplifie le paquetage et le rend plus facile aux concepteurs de rajouter de nouvelles analyses en fonction de leur exigence. Le sous-paquetage GQAM offre des domaines spécialisés, dont les analyses sont fondées sur le comportement des logiciels, comme d'ordonnançabilité et la performance et aussi d'autres NFPs comme l'énergie, la sécurité, la disponibilité, la mémoire et la fiabilité. En d'autres termes, on peut dire que l'objectif principal du GQAM est de révéler l'utilisation des ressources basées sur le comportement du système. Cette fonctionnalité fait croire que MARTE jouera un rôle important quand il s'agit de la modélisation des ICPs (Industrial Communication Protocols). Les constructions définies à l'aide de ce sous-paquetage sont ensuite affinées dans les autres sous-paquetages pour l'ordonnançabilité et l'analyse de performance.
- **Modélisation d'analyse d'ordonnançabilité (SAM):** Ce sous-paquetage traite l'analyse d'ordonnançabilité. Il aide à prévoir, si le système étudié pourrait rencontrer certaines contraintes temporelles, comme le manque de ration, les échéances, etc. L'ordonnancement influence sur la synchronisation et les performances d'une manière cruciale quand il s'agit des applications temps réel comme les ICPs et par conséquent, l'analyse d'ordonnançabilité devient importante pour le calcul des bornes garanties des temps de réponse et les charges de traitement des ressources. Ce sous-paquetage fournit un groupe d'annotations communes pour l'analyse d'ordonnançabilité.

- **Modélisation analyse de performance (PAM):** Ce sous-paquetage offre un support pour l'analyse des performances, c'est à dire qu'il offre des moyens pour déterminer si un système peut fournir des performances adéquates dans des conditions comportementales non-déterministes, fondée sur des valeurs statistiques. Le sous-paquetage PAM comprend à la fois, une analyse «single-case», pour un ensemble particulier de paramètres d'entrée et aussi l'analyse «multi-case», par exemple « l'analyse de sensibilité» qui aide à identifier les situations dangereuse de surcharge, le paramètre opérationnel idéal, etc, et « l'analyse des capacités » qui identifie les capacités de conception ou de configuration.

N.B.: NFPs = Non-Functional Properties, GRM = Generic Resource Modeling, GCM = Generic Component Model, Alloc = Allocation modeling, RTEMoCC = RTE Model of Computation & Communication, DRM= Detailed Resource Modeling, SRM = Software Resource Modeling, HRM = Hardware Resource Modeling, GQAM = Generic Quantitative Analysis Modeling, SAM = Schedulability Analysis Modeling, PAM = Performance Analysis Modeling.

4.4. MARTE et l'analyse d'ordonnançabilité

Marte propose un ensemble minimal d'annotations communes pour l'analyse d'ordonnançabilité basée sur les modèles. Cet ensemble minimal fournit suffisamment d'informations pour réaliser les analyses d'ordonnançabilité courante. Marte est constitué d'un modèle de domaine conceptuel et d'un profil UML concret. Le modèle du domaine définit les concepts clés et les relations entre ces différents concepts utilisés lors de la description des systèmes temps réel, tandis que le profil spécifie comment les éléments du modèle de domaine sont implémentés par l'intermédiaire des stéréotypes et comment ils étendent les méta-classes UML.

4.4.1. Le modèle de domaine pour l'analyse d'ordonnancement

Le modèle de domaine pour la modélisation de l'analyse d'ordonnancement est structuré comme dans SPT, en utilisant le concept de contexte temps réel (real time situation). Du point de vue prédictif, les modèles d'analyse d'ordonnançabilité sont intrinsèquement basés sur les instances. Ainsi un contexte temps réel représente une situation spécifique du système qui travaille dans un mode et une configuration particuliers. Néanmoins, des modèles descriptifs de haut niveau peuvent également être construits en utilisant les sous-profils RTEM, des modèles d'analyse concrets pouvant ensuite être instanciés pour des analyses spécifiques.

Un contexte temps réel répertorie les informations nécessaires pour réaliser une analyse spécifique. A partir d'un contexte temps réel et de ses éléments, un outil peut suivre les liens dans un modèle, afin d'extraire les informations nécessaires à l'analyse. Un contexte temps réel est décrit par des modèles séparés, associés par trois préoccupations génériques (Figure 4.2):

- Le contexte de charge de travail (Workload Situation) : un ensemble de réponses du début à la fin (end-to-end responses) déclenchées par des stimuli externes (par exemple, des évènements de l'environnement) ou internes (par exemple un minuteur) (Figure 4.3).
- L'exécution comportementale (behavior execution) : une description de la chaine d'actions exécutées en réponse de la charge de travail, incluant l'accès aux ressources partagées et leurs services (Figure 4.4).
- La plateforme des ressources (platform resources) : une architecture concrète des ressources matérielles et logicielles utilisées (Figure 4.5).

La séparation des préoccupations de modélisation a permis l'organisation du modèle de domaine en sous-parties significatives.

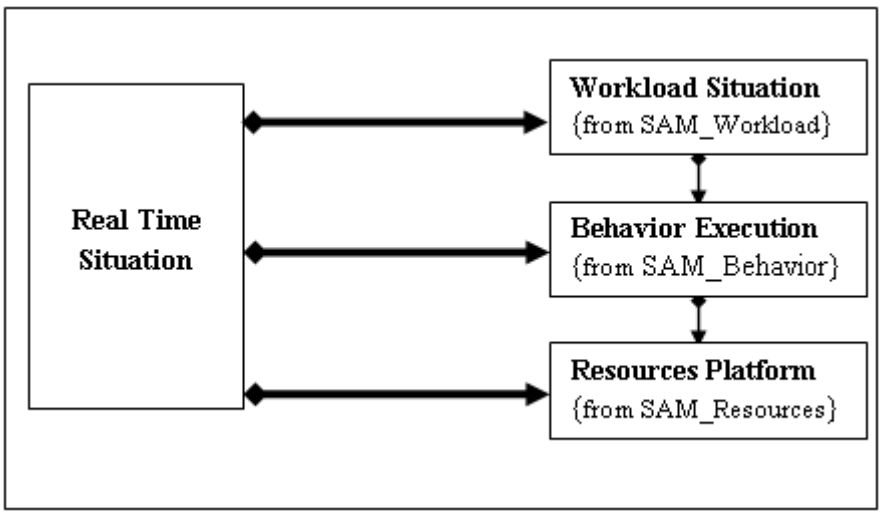

Figure 4.2. Contexte Temps réel.

4.4.1.1. Le modèle de domaine de charge de travail

Ce modèle (Figure 4.3) décrit les constructions requises pour spécifier la charge de calcul sur le système et les informations quantitatives associées : les stimuli de début à fin, les réponses et les exigences temporelles. Un contexte de charge de travail (workload situation) est habituellement défini par l'ensemble de stimuli déclenchant les calculs. Dans le Framework SAM, on appelle l'instance d'un stimulus donné l'occurrence d'un évènement. Lorsque le stimulus peut arriver de manière régulière, on fait allusion à la récurrence des évènements comme étant un élément déclencheur (trigger).

On appelle réponse aux occurrences des événements le calcul engendré par un élément déclencheur. Suivant la nature de l'implémentation de la réponse, on pourrait concrétiser le calcul par une simple tache exécutée sur un seul processeur, ou comme des taches dépendantes sur un ou plusieurs processeurs ; le comportement détaillé impliquant une réponse n'est pas inclus dans ce modèle, ceci est fait dans le modèle de comportement.

On pourra remarquer que les concepts d'élément déclencheur et de réponse spécifient seulement des comportements de début à fin, plus spécifiquement leur aspect « cause » et leur aspect « effet ». A fin de grouper ces deux concepts en une seule entité de modélisation, on adopte le concept de transaction. Une transaction constitue le comportement de cause/effet et de début à fin décrivant un calcul distinct dans le système, par conséquent, l'ensemble des transactions définissant un scénario de charge en vue d'une analyse compose un seul contexte de charge de travail, un contexte de charge de travail peut correspondre à un mode d'opération du système (par exemple, un mode de démarrage, un mode dégradé ou un mode de fonctionnement standard) ou à un niveau d'intensité d'évènements issus de l'environnement

Les concepts de transaction, d'évènement déclencheur et de réponse possèdent un ensemble de propriétés non fonctionnelles, utilisées pour réaliser des analyses quantitatives de performances. Le sous-paquetage SAM prédéfinit un ensemble commun utilisé par la plupart des techniques d'analyse.

Par exemple un élément déclencheur est caractérisé par son schéma d'arrivée (arrival pattern) qui est un type de donnée structurée contenant des attributs concrets tels que sa période, le temps minimum d'arrivée, la fonction de distributions, etc. d'un autre coté les réponses possèdent un ensemble de NFPs portant sur la latence concrétisées par des délais de début à fin ou des exigences temporelles, comme des temps de début à fin ou des échéances.

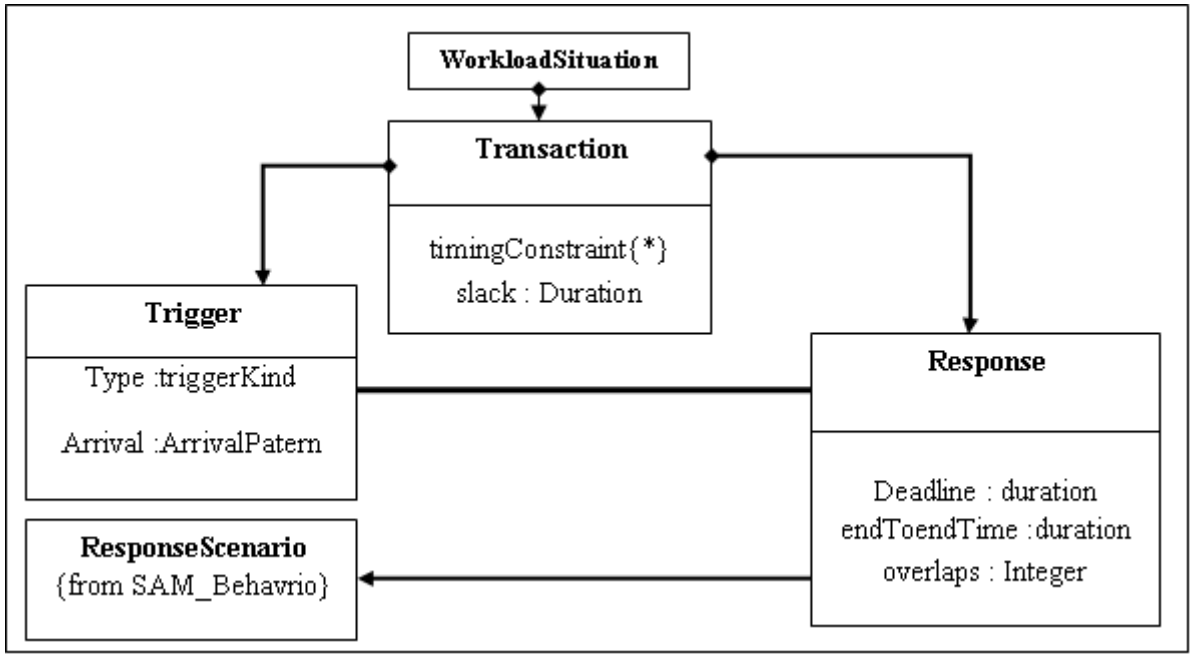

Figure 4.3. Modèle de domaine de la charge de travail. [Omg09]

4.4.1.2. Le modèle de domaine du comportement

Dans ce modèle (Figure 4.4), le concept d'exécution de comportement sert à collecter les descriptions détaillées du comportement des réponses qui sont à leur tour appelés scénario de réponse (reponse scenarios). Le scénario de réponse est la spécification de petits segments d'exécution de code et leurs relations de précédence et de simultanéité. Cet aspect de la modélisation est essentiel aux analyses quantitatives visant à évaluer comment les segments d'exécutions s'affrontent pour l'utilisation des ressources de la plate-forme d'un point de vue temporel.

De cette manière, le comportement détaillé d'une réponse donnée est représenté par une série ordonnée de pas d'exécutions appelée scheduling action (SAction). En considérant une approche holistique pour l'analyse, une action temps réel peut représenter le temps nécessaire à l'exécution d'une portion de code ou bien représenter l'envoi d'un message à travers un réseau .Le classement des SActions suit un schéma de prédécesseur-successeur, avec la possibilité de définir plusieurs successeurs et plusieurs prédécesseurs simultanés. La granularité pour une SAction est souvent un choix de modélisation qui dépend du niveau de détail considéré. Une SAction à un certain niveau d'abstraction peut être décomposée en un ensemble de SActions de grain plus fin. Les scénarios de réponse utilisent les services de ressources pour l'exécution au travers de l'allocation de SActions à des entités ordonnançable (schedulable entities) et aux autres servies de la plateforme à travers des appels aux ressources partagées (shared resources).

4.4.1.3. Le modèle de domaine de la plateforme

Dans le sous-pacquetage SAM, le concept de plateforme de ressources (resources platform) correspond au modèle de ressources d'ingénierie introduit par le profil SPT. Ceci inclus non seulement les ressources matérielles (CPU, ressources de réseau, etc.) mais également les ressources logicielles (processus, tampons, etc.). La figure 4.5 montre un framework pour décrire une plateforme de ressources. On forme une version abstraite à partir d'un modèle de plateforme plus structuré et détaillé (sous-profil MARTE pour l'exécution logicielle et matérielle). Cette version abstraite met l'accent sur l'expression de NFPs orientées vers l'analyse d'ordonnançabilité, sans avoir à distinguer entre différentes niveau d'abstraction (matériel, système d'exploitation temps réel, middleware, etc).

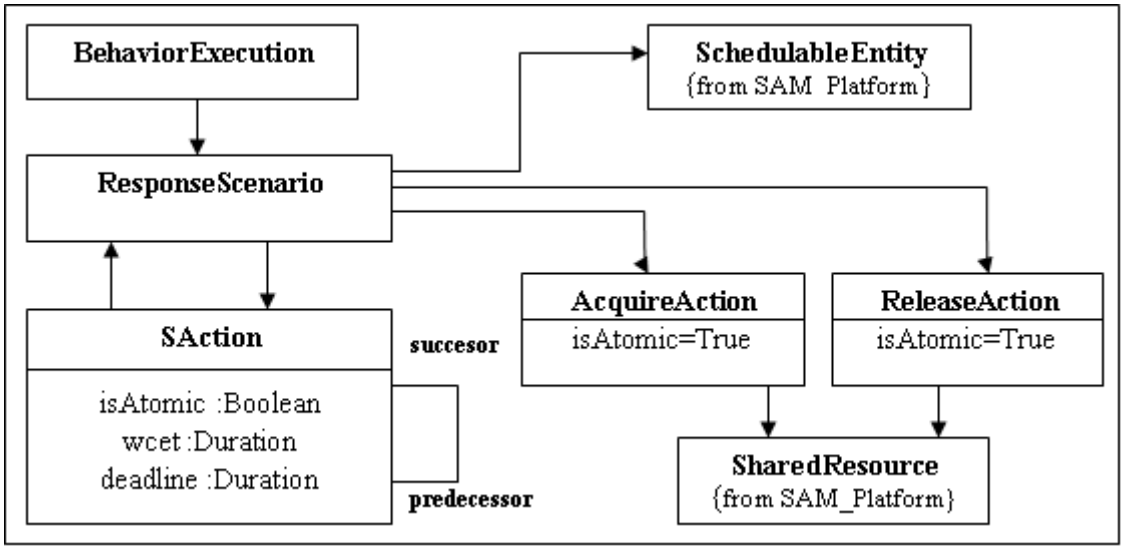

Figure 4.4. Modèle de domaine du comportement. [Omg09]

Le modèle de plateforme est constitué d'un ensemble de ressources avec des NFPs explicites. Ce modèle distingue deux types de moteur de traitement : les moteurs d'exécution (execution engines) comme les processeurs ou les coprocesseurs et le moteur de communication (communication engine) comme le réseau ou les bus. Le Framework SAM attribue des NFPs génériques à chacun de ces éléments, comme par exemple, la vitesse de transmission ou la vitesse de calcul, les temps de surcharge liés aux interruptions, etc.

Une entité ordonnançable (schedulable entity) correspond à une ressource active protégée, utilisée pour exécuter des SActions ou des scénarios de réponse complets. Dans un système d'exploitation temps réel, c'est le mécanisme qui représente une unité d'exécution simultanée, comme une tâche ou un processus. Pour un réseau une entité ordonnançable correspond à un canal ou une unité de traitement des messages qui peuvent être caractérisés par des paramètres d'ordonnancement concrets (comme par exemple la priorité pour un bus).

Les moteurs de traitement possèdent des ressources partagées. Ces ressources partagées peuvent être des systèmes d'entrée/sorties, des canaux DMA, des adaptateurs réseaux, etc. Elles sont allouées dynamiquement aux entités ordonnançables par le biais d'une politique d'accès. Les politiques d'accès communes sont par exemple la politique FIFO (premier arrivé premier servi) ou des politiques basées sur les priorités inversées, comme les priorités plafond.

Les ordonnanceurs peuvent jouer deux rôles dans ce modèle : les ordonnanceurs systèmes (system schedulers) et les ordonnanceurs secondaires (secondary schedulers). Les ordonnanceurs systèmes proposent à ses entités ordonnançables allouées l'ensemble des capacités de calcul fournies par ses processeurs. Ce sont typiquement les ordonnanceurs des systèmes temps réel embarqués, de leur coté, les ordonnanceurs secondaires proposent seulement les capacités fournies par leur entité d'ordonnancement d'accueil. Cette structure hiérarchique est régulièrement utilisée dans les systèmes temps réel, lorsque les utilisateurs veulent appliquer des ordonnancements dynamiques sur des systèmes d'exploitation ne fournissant que de l'ordonnancement statique.

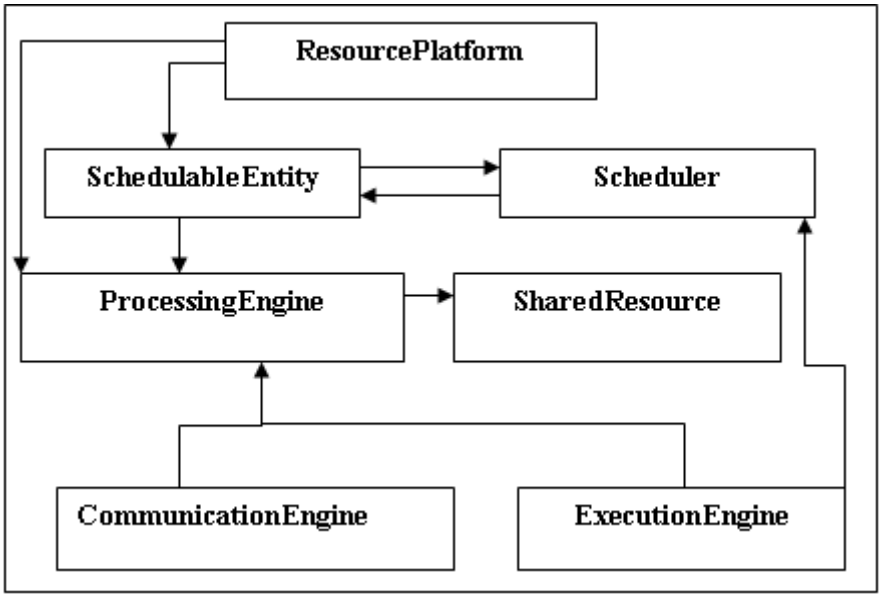

Figure 4.5. Modèle de domaine des ressources. [Omg09]

4.5. Papyrus et son support pour le profil MARTE

4.5.1. Papyrus

Dans le contexte de développement des outils qui supportent MARTE, des différent projets ont été créés (OpenDevFactory -www.usine-logicielle.org-, OpenEmbeDD -openembedd.inria.fr- et CORTESS -www.carroll-research.org-). La société CEA LIST a développé une implémentation open-source du profil UML pour MARTE (incluant un support pour le langage VSL). Il s'agit en fait d'un projet Eclipse qui s'appelle Papyrus (www.papyrusuml.org). Papyrus est un outil dédié à la modélisation en UML2 (Figure 4.6).

Figure 4.6. Logo de Papyrus.

Parmi les caractéristiques essentielles de papyrus, on cite :

- La compatiblité avec Eclipse UML2.
- Le respect total du standard UML2 tel que défini par l'OMG.
- Le respect total du standard DI2 (Diagram Interchange).
- L'architecture extensible de Papyrus qui permet aux utilisateurs d'ajouter de nouveaux diagrammes, de nouvelles procédures de génération de code, etc.
- Le support pour le développement de profils UML2 pour le profil MARTE.

Son interface graphique (Figure 4.7) se compose de plusieurs sous-fenêtres :

1. Editeur graphique des diagrammes UML comportant une palette des éléments de modélisation.
2. Navigateur de l'arborescence du dossier du projet.
3. Cet onglet affiche un aperçu sur l'éditeur.
4. Cet onglet affiche les propriétés des éléments du projet.

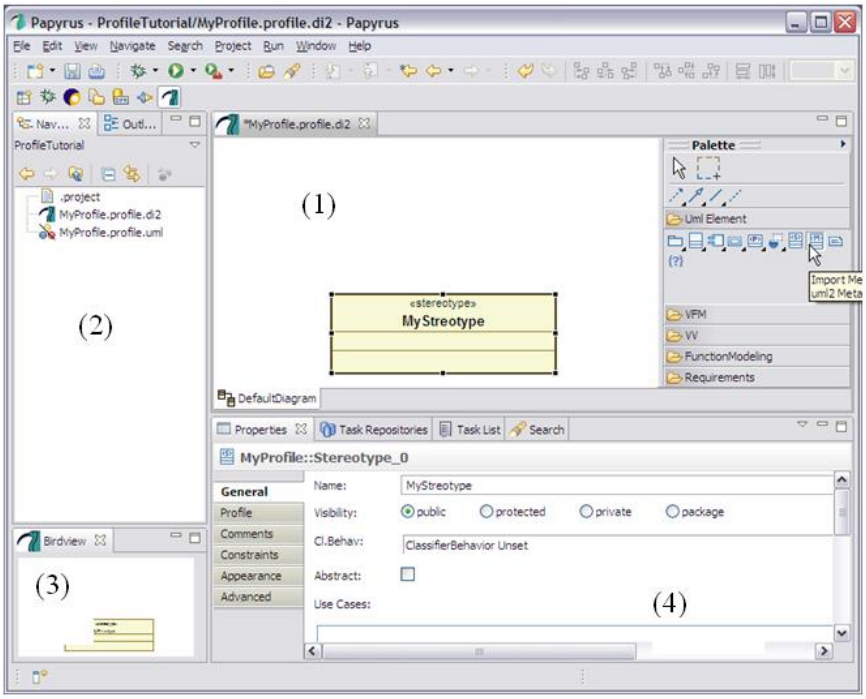

Figure 4.7. Interface graphique de Papyrus.

4.5.2. Utilisation du profil MARTE

Rappelons que le profil MARTE est un ensemble de stéréoptypes, de valeurs marquées et des types de données. L'utilisation du profil consiste en **l'application** des stéréotypes sur les éléments UML (Figure 4.8) permettant de les annoter par des informations spécifiques à un domaine bien précis comme le domaine du temps réel.

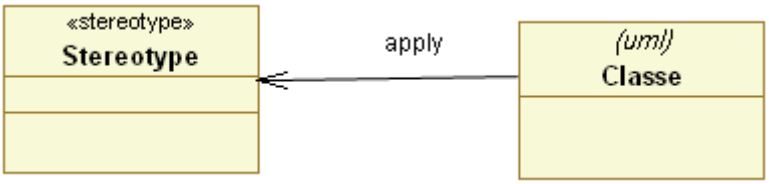

Figure 4.8. Une classe appliquant un stéréotype.

Papyrus est une plateforme Eclipse qui est extensible par des plugins. Le plugin MARTE disponible depuis le site officiel du Papyrus (www.papyrusuml.org) s'intègre correctement avec Papyrus. Pour un projet Papyrus, il suffit de lui appliquer le profil MARTE pour qu'on puisse utiliser ses différents sous-paquetages (Figure 4.9). Dorénavant tous les éléments UML pourront appliquer les stéréotypes du profil MARTE ce qui va leur permettre d'hériter les valeurs marquées de ces stéréotypes qui seront par la suite annotées au-dessus des éléments UML. L'exemple montré dans Figure 4.10. montre une classe ordinaire et une autre appliquant le stéréotype « saCommHost » et ayant parmi les valeurs marquée ; le mode de transmission 'tansmMode' contenant la valeur 'halfDuplex', toutes ces informations sont annotées sur la classe (Figure 4.10).

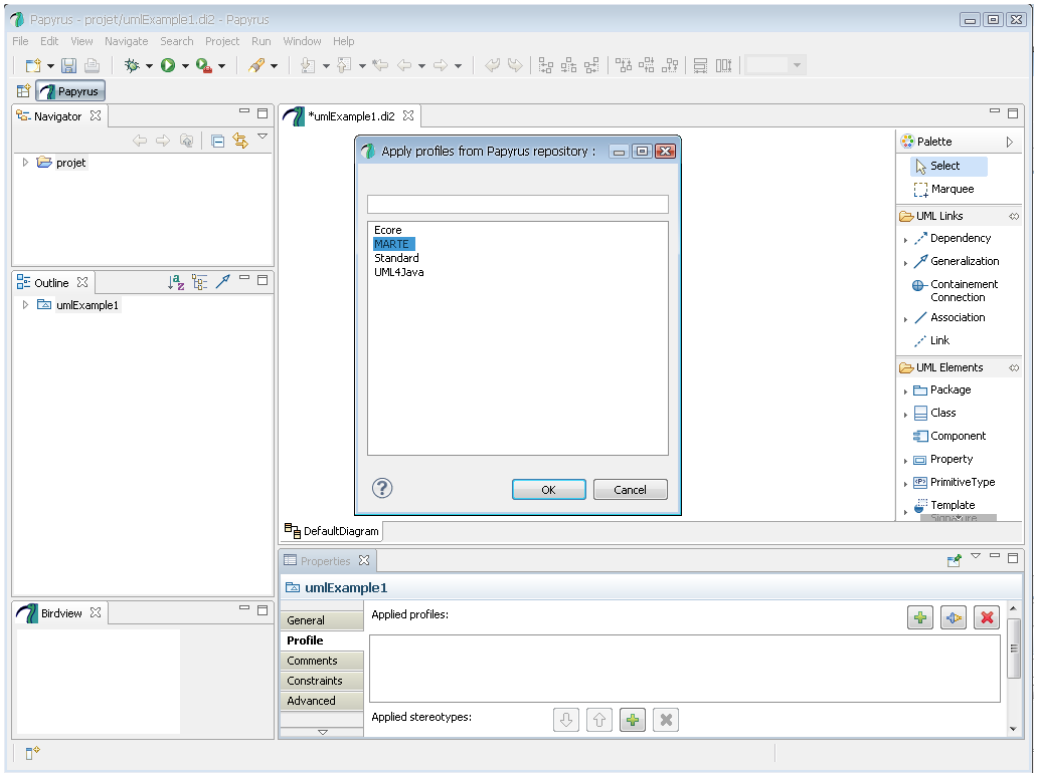

Figure 4.9. Application du profil MARTE pour un projet Papyrus.

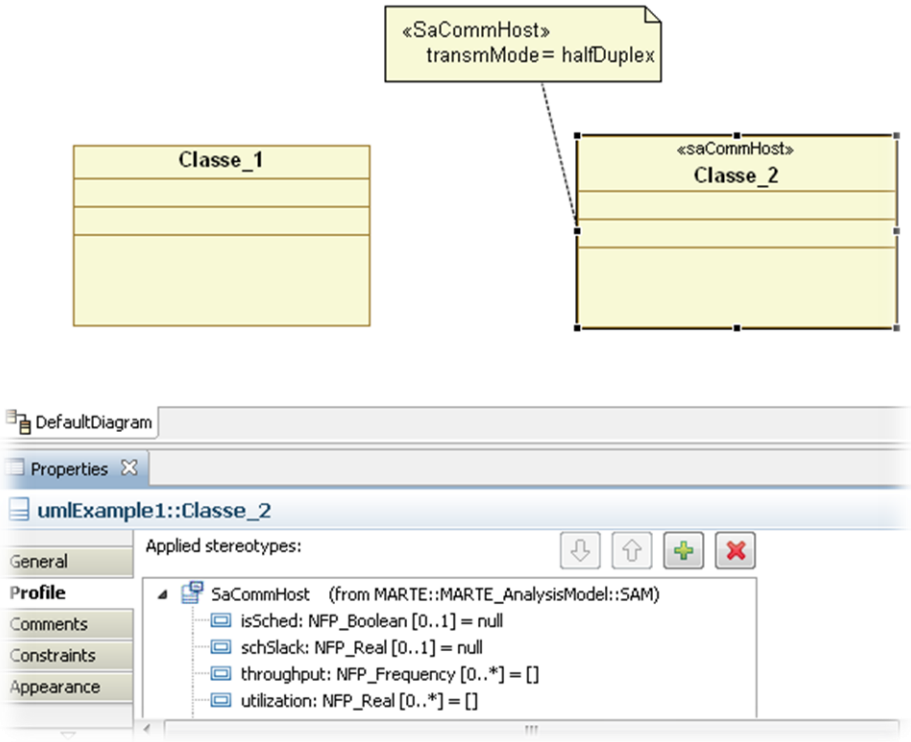

Figure 4.10. Classe ordinaire (Classe_1) et une autre (Classe_2) appliquant le stéréotype SaCommHost.

4.6. Framework basé MARTE pour l'analyse d'ordonnançabilité

MARTE comporte deux cotés, modélisation (des ressource, composant logiciels, et l'architecture matériel), et l'analyse. Le coté analyse du MARTE dispose de plusieurs composantes : ordonnancablité, performance, en plus tout ce qui concerne les propriété non-fonctionnelles (sécurité, l'énergie, la mémoire , la fiabilité.).

Dans notre étude, on se limite à l'analyse d'ordonnançabilité (schedulibility analysis) et son support dans le profil MARTE, donc précisément le sous-profil SAM, on va proposer un **framework basé MARTE pour l'analyse d'ordonnancabilité** (Figure 4.11).

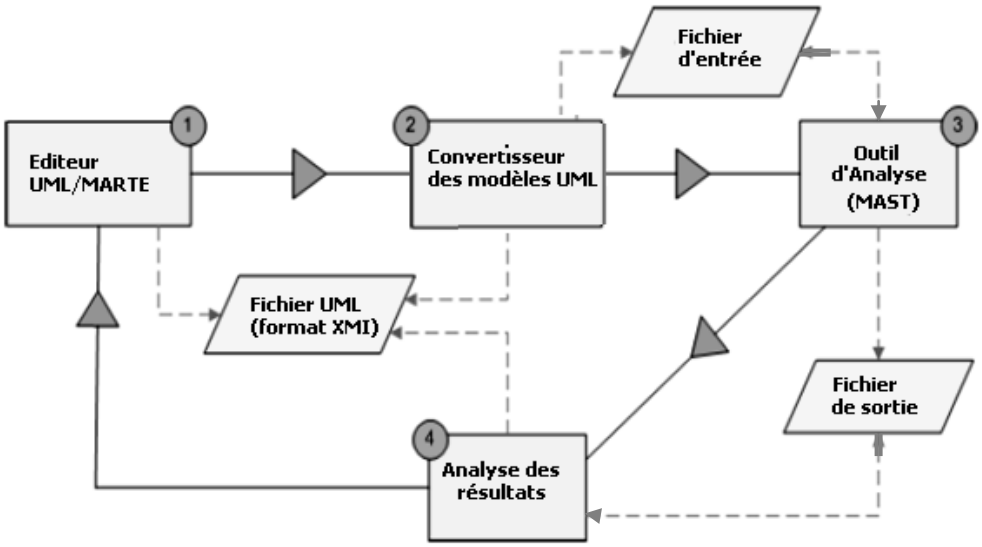

Figure 4.11. Schéma général du Framework proposé basé MARTE pour l'analyse d'ordonnançabilité.

Notre Framework se décompose en quatre parties où chacune se charge d'une tâche bien spécifique, comme suit:

1. **Editeur UML/MARTE :** le rôle de cette partie est la modélisation graphique des éléments UML avec l'application des stéréotypes MARTE spécifique à l'analyse d'ordonnançabilité. On propose un modèle des éléments nécessaires à l'analyse d'ordonnançabilité qui comporte deux vues ; une vue structurelle et une autre comportementale. Et c'est aux concepteurs des systèmes temps de se servir de ce modèle afin qu'ils conçoivent leurs systèmes.
2. **Convertisseur des modèles UML :** Afin qu'on puisse analyser les modèles concus par les outils d'analyse, une phase de conversion des modèles UML de leur format XMI vers le format des fichiers d'entrée des outils d'analyse s'avère obligatoire. Pour cela on développe un convertisseur basé sur l'analyse des fichiers XMI (XMI parsing).
3. **Outil d'analyse :** pour le domaine d'analyse d'ordonnançabilité, plusieurs outils d'analyse et de simulation ont été développées, à titre d'exemple on trouve : MAST, SymTA/S, TIMES. Dans notre Framework on opte pour l'outil MAST (Modeling and Analysis Suite for real-Time applications).
4. **Analyse des résultats :** A partir des résultats fournis par l'outil d'analyse du système (concernant l'ordonnancabilité, le temps de réponse, etc), les concepteurs mettraient en cause probablement la conception de départ.

Dans ce qui suit, on détaille chacune des parties.

4.6.1. Editeur UML/MARTE

on s'intéresse à la modélisation des éléments de base nécessaire pour prendre en charge les aspects de temps et d'ordonnancement liés à l'analyse d'ordonnancabilité, et ceci utilisant les support UML/MARTE fournit par l'outil Papyrus, pour cela on propose un modèle de deux vues : une vue structurelle et une autre comportementale, qui peut être comparé au modèle de 5 vues proposé par Philippe Kruchten [Kru95] et aussi le modèle de 3 vues proposé par Matthias Hagner [HH08]. Ces deux vues sont choisies en fonction du besoin de l'analyse à appliquer qui est l'analyse d'ordonnancabilité, et elles doivent aussi contenir toutes les informations demandées par les outils d'analyse. Pour cette raison, on se trouve des fois obligé de rajouter de nouvelles informations supplémentaires aux éléments du modèle proposé, qui ne sont pas fournit par le support UML/MARTE de l'éditeur utilisé (Papyrus). En parallèle, d'autres détails s'avèrent non judicieux pour l'analyse d'ordonnancabilité, et par conséquent, ils seront tout simplement ignorés.

4.6.1.1. Vue Struturelle

Dans cette vue, on s'intéresse à la modélisation de l'aspect statique du système. Pour l'analyse d'ordonnancabilité, on a deux classes d'élements : les éléments logicielles et les élement matérielle. Les éléments logicielles sont principalement les tâches ordinaires, l'ordonnanceur ainsi qu'un autre type de tâches qui sont les tâches de communication représentant les primitives qui manipulent les ressources critiques par un accès exclusif. Les éléments matériels sont les processeurs et les ressources critiques. Cette vue s'occupe donc de décrire l'aspect distribution (affectation des tâches aux processeurs), ainsi que l'attribution des tâches de communication à leurs ressources.

Eléments logiciels

- **Tâche ordinaire :** en UML, elle correspond à une classe (ayant un nom, à titre d'exemple Tache_01) appliquant le stéréotype « SchedulableResource», et qui a des valeurs marquées nécessaire à l'analyse d'ordonnancablité (le deadline, la période, et la priorité), et qui propose des services (qui sont les méthodes de la classe : Calcul() et Sauvegarde()). Ces services doivent appliquer le stéréotype « saStep » ou « saExecStep » dont une valeur marquée est essentiel à l'analyse d'ordonnancabilité, qui est le temps d'exécution (execTime).
  Une information important doit être rajouter afin que l'outil d'analyse puisse distinguer entre les tâches ordinaires et les tâches de communications. Cette information est matérialisée par un ajout d'une propriété à la classe, nommée 'type' qui a pour valeur 'ORDINAIRE' (Figure 4.12).

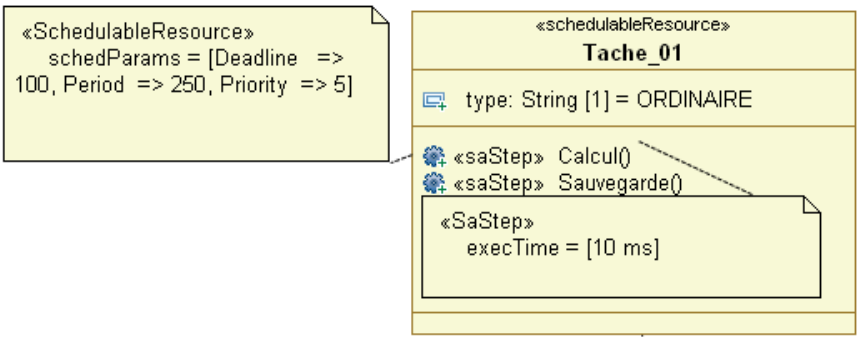

Figure 4.12. Tâche ordinaire utilisant le support UML/MARTE.

- **Tâche de communication :** elle est pratiquement identique à une tâche ordinaire, à une différence près, est que la valeur de la propriété type est bien 'Communication', ainsi que les services proposés par cette tâche applique plutôt le stéréotype « saCommStep », qui sont les primitives permettant l'accès exclusif aux ressources critiques (Figure 4.13).

Figure 4.13. Tâche de communication utilisant le support UML/MARTE.

- **Ordonnanceur :** en UML, il correspond à une classe appliquant le stéréotype « scheduler » permettant de spécifier les paramètres de l'ordonnanceur tels que la politique d'ordonnancement (Figure 4.14).

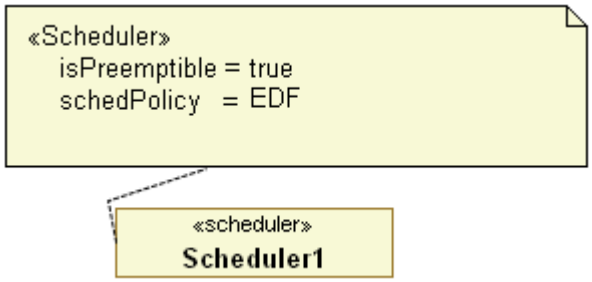

Figure 4.14. Ordonnanceur utilisant le support UML/MARTE.

Eléments Matériels

- **Processeur :** en UML, il correspond à une classe appliquant le stéréotype « saExecHost » permettant de spécifier des valeurs marquée fortement liés à la nature du processeurs tel que le temps de commutation de tâches (ISRswitchT), ainsi que de spécifier l'ordonnanceur qui lui est lié (Figure 4.15).

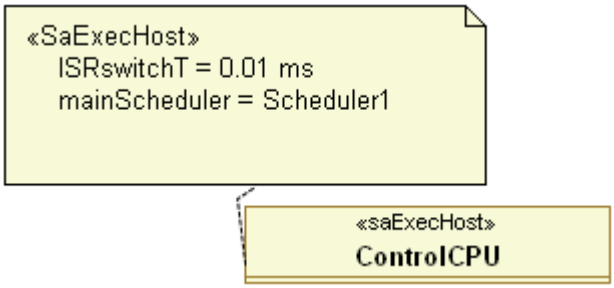

Figure 4.15. Processeur utilisant le support UML/MARTE.

- **Ressource :** en UML, elle correspond à une classe appliquant le stéréotype « saCommHost » permettant de spécifier des propriétés fortement liées à cette ressource comme le débit 'throughput' et le mode de transmission 'transmMode' d'un medium de communication 'Ethernet' (voir Figure 4.16).

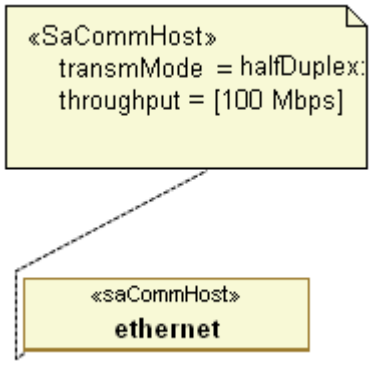

Figure 4.16. Ressource utilisant le support UML/MARTE.

De cette façon, et à partir de cet ensemble réduit d'élément UML/MARTE dédié à l'analyse d'ordonnançablité, un simple système comportant un processeur, une tâche ordinaire et une tâche de communication et une ressource, peut avoir la vue structurelle suivante (Figure 4.17) :

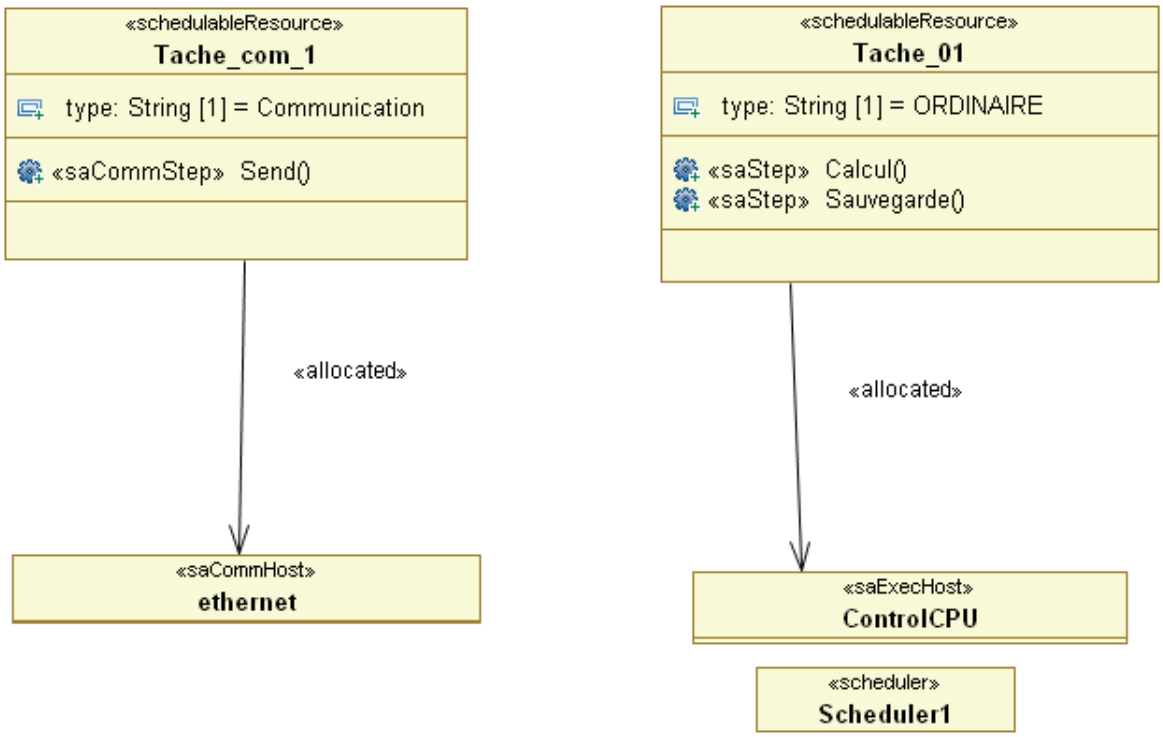

Figure 4.17. Vue structurelle d'un simple système utilisant le support UML/MARTE.

4.6.1.2. Vue Comportementale

Dans cette vue, on s'intéresse par la modélisation de l'aspect dynamique du système et précisément le comportement de tâche. Quand une tâche prend l'exécution sur processeur, elle va exécution des services internes, mais le plus important ce sont les appels qu'elle fasse aux primitives manipulant les ressources critiques gérées par les tâches de communication. Donc pour décrire cette interaction entre les tâche oridinaires et les tâches de communication, on opte pour un diagramme de séquence où le scénario d'exécution d'une tâche sera décrit par une interaction. Pour un ensemble de tâches concurrents, on utilse un fragment combiné pour spécifier un opérateur concurrentiel entre ces tâches (séquentiel, parallèle, conditionné, etc.)

- **Interaction :** en UML, elle correspond à une interaction d'un diagramme de séquence. Elle applique le stéréotype « GaScenario » qui dispose d'une valeur marquée appelée « usedResources » permettant de spécifier la ou les tâches qui ont un comportement décrit par cette interaction (Figure 4.18). Donc le temps d'exécution d'une tâche va dépendre des services internes sollicités ainsi que les primitives de communication appelées.

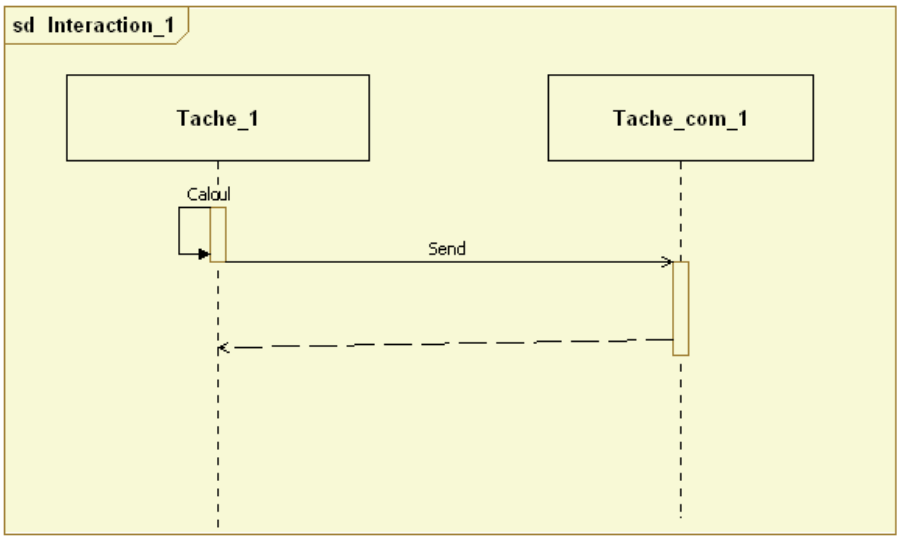

Figure 4.18. Interaction représentant le scénario d'exécution de la tâche 'Tache_1' utilisant le support UML/MARTE.

- **Fragment combiné :** en UML, il correspond à un fragment combiné du diagramme de séquence. Il applique le stéréotype « GaWorkloadBehavior » qui dispose d'une valeur marquée appelé ' behavior' du type « GaScenario » qui regroupe l'ensemble des scénarios (interactions) associés aux tâches respectives sachant que l'exécution de ces tâches sera selon l'opérateur concurrentiel spécifié par la propriété 'Interaction operator' qui pourra avoir comme valeur [seq, alt, opt, break, par, strict, loop, critical, neg, assert, ignore, consider] (Figure 4.19).

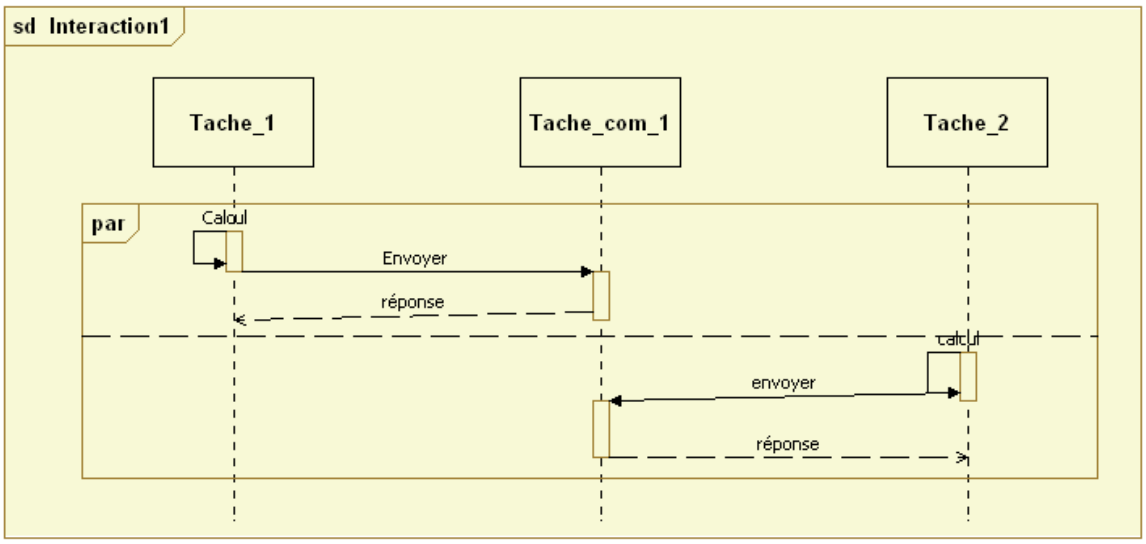

Figure 4.19. Vue comportementale de deux tâches concurrentes qui s'exécute en parallèle utilisant le support UML/MARTE.

4.6.2. Convertisseur des modèles UML

Après avoir terminé la conception des deux vues (structurelle et comportemantale) du système, le modèle UML doit être converti au format des fichier d'entrée de l'outil d'analyse d'ordonnancabilité MAST. Pour un projet Papyrus, n'importe quel modèle est représenté par deux fichiers: un fichier .uml et l'autre .di2. Le fichier .uml contient toutes les informations UML; les noms des classes, des stéréotype appliqués, les valeurs des paramètres, etc. Tandis que le fichier .di2 (Diagram Interchange 2)[Omg03] contient les information nécessaire pour la création d'une représentation graphique du fichier .uml. Tous les deux fichier sont écrit en XML. Le fichier .uml doit être analyser afin d'extraire les informations pertinentes pour la construction d'un fichier d'entrée MAST valide et complet. Ainsi, on développe un analyseur des fichiers XMI (XMI parser) capable de reconnaitre les éléments UML/MARTE à partir de leur codage XMI.

> *XMI (XML Metadata Interchange) est un standard créé par l'OMG pour l'échange d'informations de métadonnées UML basé sur XML. Il s'agit d'un procédé de sérialisation permettant de décrire des objets UML sous forme XML.*

Cette analyse reconnaîtra les éléments UML en se basant sur leurs balises XML, à l'aide du tableau (Tableau 4.1) suivant :

Tableau 4.1. Correspondance élément UML/MARTE – balise XML

Elément UML/MARTE	balise XML
Classe	<packagedElement xmi :type = 'uml:classe'>
Attribut d'une classe	< ownedAttribute >
Méthode d'une classe	< ownedOperation >
Interaction	<ownedBehavior xmi :type=uml' :Interaction>
Fragment combiné	<fragment xmi:type="uml:CombinedFragment">
Stéréotype SchedulableResource	< GRM:SchedulableResource >
Stéréotype saExecHost	< SAM:SaExecHost>
Stéréotype saCommHost	<SAM :SaCommHost>
Stéréotype scheduler	<GRM :Scheduler>
Stéréotype saStep	<SAM :saStep>
Stéréotype saCommStep	<SAM :saCommStep>
Stéréotype allocated	<Alloc :Allocated>
Stéréotype GaScenario	<GQAM : GaScenario>
Stéréotype GaWorkloadBehavior	<GQAM : GaWorkloadBehavior>

Dans le format XML du fichier .uml, chaque balise a un identificateur unique représenté par l'attribut « xmi :id » et ceci afin d'éviter l'ambiguité et assure l'unicité des balises. Pour récupérer un attribut d'une balise, par exemple le nom d'une classe, le morceau du code Java suivant permettant facilement de le faire :

```
public void startElement (String uri, String localName, String qName,
Attributes attributes) throws SAXException
{
if(qName.equalsIgnoreCase("packagedElement") &&
attributes.getValue("xmi:type").equals("uml:classe"))
{
String Nom_Classe= attributes.getValue("name") ;
System.out.println(Nom_Classe) ;
}
}
```

De l'autre coté, un fichier d'entrée MAST est un fichier textuel, qui se compose de plusieurs sections, comme montre Tableau 4.2. Chacune de ces sections seront construit en se basant sur les informations précédemment extraites du fichier .uml.

Tableau 4.2. Sections d'un fichier d'entrée MAST avec leur format.

Section	Format	Description & éléments MARTE impliqués
Entête	**Model (** **Model_Name => Controller,** **Model_Date => 2000-01-01);**	Nom et date du modèle d'ordonnancement.
Processing Resources	**Processing_Resource (** **Type=> Regular_Processor,** **Name=> CPU_1);**	Nom et type du processeur. [**saExecHost**]
Scheduler	**Scheduler (** **Type => Primary_Scheduler,** **Name => Scheduler_1,** **Host => Processor_1,** **Policy =>** **(Type => EDF,** **Worst_Context_Switch => 102.5));**	Nom et type d'ordonnanceur, politique d'ordonnancement et le processeur hôte avec son temps de commutation de tâches (en ms). [**scheduler**]
Scheduling servers	**Scheduling_Server (** **Type=> Fixed_Priority,** **Name=> Servo_Control,** **Server_Sched_Parameters=> (** **Type=> EDF_policy,** **Deadline=> 5000,** **Preassigned => No),** **Synchronization_Parameters => (** **Type => SRP_Parameters,** **Preemption_Level => 2000,** **Preassigned => No),** **Server_Processing_Resource=> Processor_1);**	Paramètres d'ordonnancement d'une tâche (nom de la tâche, deadline, politique appliquée, processeur hôte, etc.). [**scheduler**, **SchedulableResource**]
Resources	**Shared_Resource (** **Type => Immediate_Ceiling_Resource,** **Name => Data_Server);**	Nom et type d'une ressource. [**saCommHost**]
Operation / Critical Sections	**Operation (** **Type => Simple,** **Name => Read_New_Point,** **Worst_Case_Execution_Time => 87,** **Shared_Resources_List=> (Servo_Data));**	Méthode d'une tâche de communication (nom, temps d'exécution et la ressource critique manipulée). [**saCommStep**]
Operation/ Task operations	**Operation (** **Type => Enclosing,** **Name => Control,** **Worst_Case_Execution_Time => 20,** **Composite_Operation_List => (Read,Send));**	Méthode d'une tâche ordinaire (nom, temps d'exécution et la liste des primitives des tâches de communication). [**saStep**, **GaScenario**, **GaWorkloadBehavior**]
Transactions	**Transaction (** **Type => Regular,** **Name => Control_Task,** **External_Events => (** **(Type => Periodic,** **Name => E1,** **Period => 100)),** **Internal_Events => (**	Description des flux d'évènements générés pour le déclenchement d'une tâches périodique (à chaque période et pour un deadline donné) ou sporadique, ces flux décrivent les contraintes de précédence entre les tâches dépendantes.

(Type => regular, [**SchedulableResource**]
name => O1,
Timing_Requirements => (
Type => Hard_Global_Deadline,
Deadline => 100,
Referenced_Event => E1))),
Event_Handlers => (
(Type => Activity,
Input_Event => E1,
Output_Event => O1,
Activity_Operation => Control,
Activity_Server => Control_Task)));

4.6.3. Outil d'analyse MAST

MAST (Modeling and Analysis Suite for Real-Time Applications) est un ensemble d'outils open source qui permet d'effectuer la modélisation d'applications temps réel ainsi que l'analyse temporelle de ces applications. MAST est encore en cours de développement. Ses outils d'analyse comprennent le pire temps de réponse, le calcul du temps de blocage, le calcul des marge de manœuvre, l'affectation optimisée des priorités ainsi que des outils de simulation (Figure 4.20).

Son utilisation est très facile et conviviale. Il prend comme entrée : le nom du fichier d'entrée, nom du fichier de sortie, la politique d'ordonnancement. En résultat, il fournit pour chaque tâche le pire temps de réponse, et en comparant ces temps de réponse avec les contraintes temporelles, on peut déterminer si le système est ordonnançable ou non. Il peut aussi fournir la marge de manœuvre pour chaque tâche. C'est en fait la quantité de temps par laquelle on peut augmenter ou diminuer le temps d'exécution d'une tâche tout gardant le système ordonnançable. Ce dernier donne une information sur combien on est proche ou loin de l'ordonnançabilité.

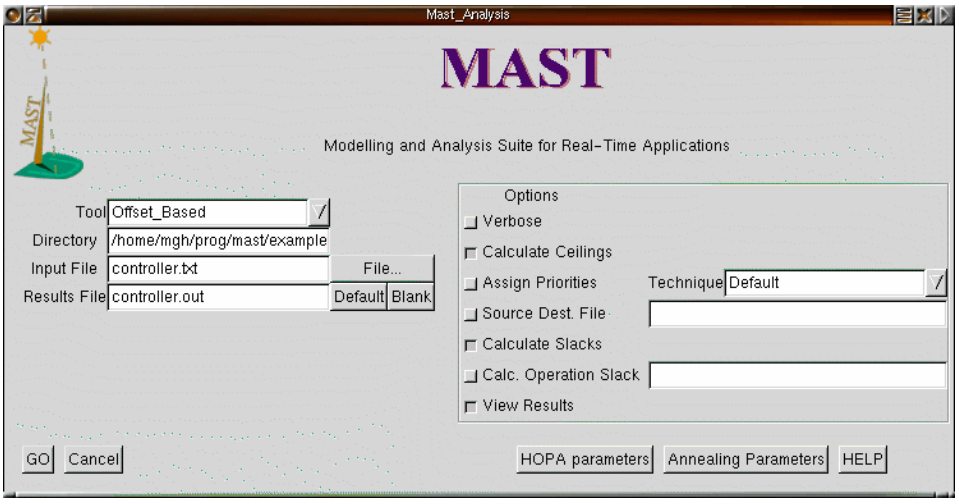

Figure 4.20. Interface graphique de l'outil MAST.

Les éléments du fichier d'entrée MAST sont les suivants:

- **Processing Resource**

  Cet élément décrit les unités matérielles de traitement ou tout simplement les processeurs. La syntaxe est comme suit :

```
Processing_Resource (
      Type                => Fixed_Priority_Processor,
```

```
      Name              => CPU,
      Worst_Context_Switch    => 0.25);
```

Où les attributs type, name et worst_context_switch correspondent respectivement le type, le nom et le pire temps de commutation de tâches du processeur.

- **Scheduler**

Cet élément décrit l'ordonnanceur. La syntaxe est comme suit :

```
Scheduler (
        Type  => Primary_Scheduler,
        Name  => CPU,
        Host  => CPU,
        Policy =>
           (Type                  => EDF);
```

Où les attributs type, name, host et policy correspondent au type, le nom de l'ordonnanceur, l'unité de traitement associée ainsi que le type de la politique d'ordonnancement.

- **Scheduling Servers**

Cet élément décrit pour chaque tâche, son type, ses paramètres d'ordonnancement, ses paramètres de synchronisation ainsi que l'unité de traitement à laquelle la tâche est attachée.

```
Scheduling_Server (
      Type                    => Regular,
      Name                    => Planning_Task,
      Server_Sched_Parameters       => (
            Type        => EDF_policy,
            Deadline    => 150,
              Preassigned             => No),
        Synchronization_Parameters => (
                Type             => SRP_Parameters,
                Preemption_Level => 2000,
                Preassigned      => No),
      Server_Processing_Resource    => CPU);
```

- **Shared Resource**

Cet élément décrit les ressources partagé, selon la syntaxe suivante :

```
Shared_Resource (
      Type  => Immediate_Ceiling_Resource,
      Name  => Data_Server);
```

Où type et name sont respectivement le type et le name de la ressource partagée.

- **Operation**

Cet élément décrit les opérations des tâches ou les primitives d'accès aux ressources critiques.
La syntaxe de déclaration d'une opération d'une tâche est comme suit :

```
Operation (
      Type                    => Enclosing,
      Name                    => Control,
      Worst_Case_Execution_Time => 20,
      Composite_Operation_List  => (Send,Read));
```

Où les attributs type, name, worst_case_execution_time et composite_operation_list correspondent respectivement au type, le nom et le pire temps d'exécution d'une tâche ainsi que la liste des primitives d'accès aux ressources critiques que la tâche va les solliciter.
La syntaxe de déclaration d'une primitive d'accès aux ressources critiques est comme suit :

```
Operation (
      Type                => Simple,
      Name                => Read,
      Worst_Case_Execution_Time => 2,
```

```
      Shared_Resources_List     => (Data_Server));
```

Où les attributs type, name, worst_case_execution_time et shared_resources_list correspondent respectivement au type, le nom, le pire temps d'exécution d'une primitive d'accès aux ressources critiques ainsi que la liste des ressources critiques sur lesquelles cette primitive va opérer.

- **Transaction**

Cet élément décrit pour une tâche les flux des évènements : interne (qui signifie l'évènement d'arrivée d'une tâche avec sa période) et externe (échéance), comme montré dans l'exemple suivant :

```
Transaction (
      Type => Regular,
      Name => Control_Task,
      External_Events => (
            (Type       => Periodic,
             Name       => E1,
             Period => 200)),
      Internal_Events => (
            (Type       => regular,
             name       => O1,
                  Timing_Requirements => (
                    Type               => Hard_Global_Deadline,
                    Deadline     => 100,
                    Referenced_Event => E1))),
      Event_Handlers => (
            (Type                 => Activity,
             Input_Event          => E1,
             Output_Event         => O1,
             Activity_Operation => Control,
             Activity_Server    => Control_Task)));
```

4.6.4. Analyse des résultats

On se basant sur les résultats fournis par l'outil d'analyse MAST concernant l'ordonnançabilité du système (et principalement le pire temps de réponse ainsi que les périodes creuses), les concepteurs du système peuvent apporter des changements au modèle afin d'obtenir des résultats plus mieux. On propose à travers ce Framework d'apporter les changements à deux niveaux :

4.6.4.1. Au niveau de la vue structurelle

Et ceci soit par la décomposition d'une tâche en sous-tâches primitives (Figure 4.21), ceci en répartissant les services indépendants d'une tâche sur les sous-tâches tout en gardant les mêmes paramètres d'ordonnançabilité. Soit par la fusion de plusieurs tâches en une seule (Figure 4.22), bien sûr ceci n'est possible que si ces tâches ont les mêmes paramètres d'ordonnancabilité.

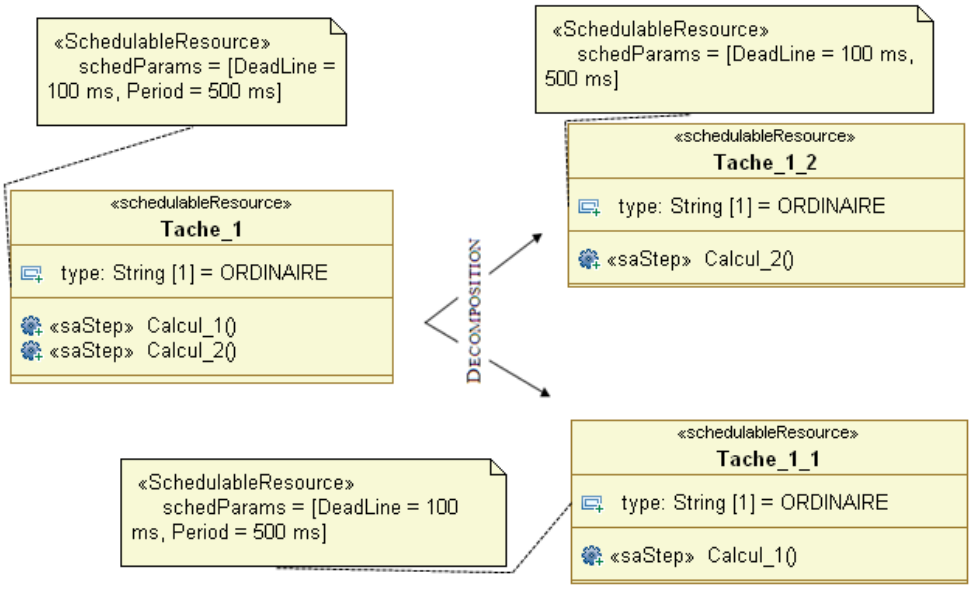

Figure 4.21. Décomposition d'une tâche en deux sous-tâches.

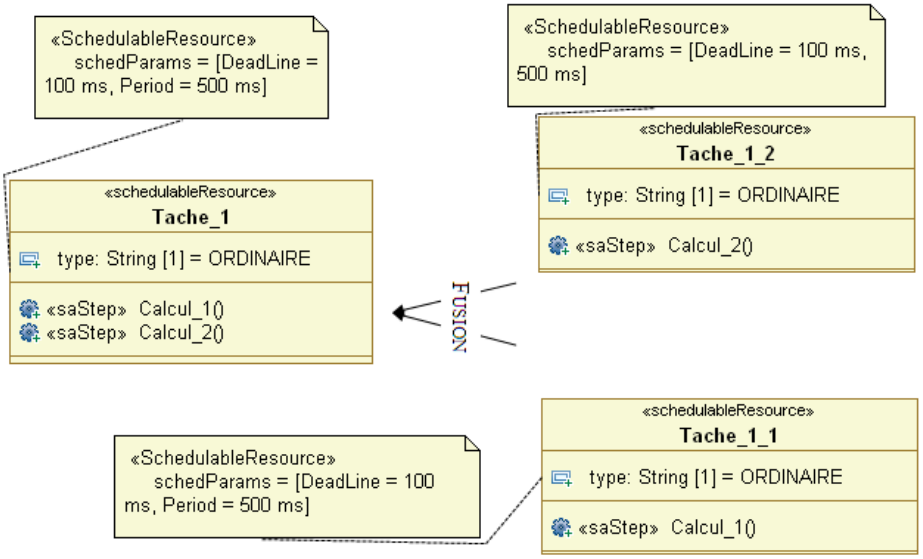

Figure 4.22. Fusion de deux tâches en une seule.

4.6.4.2. Au niveau de la vue comportementale

Et ceci par le changement du comportement (l'interaction) d'une tâche en jouant sur l'ordre des appels aux primitives des tâches de communication (Figure 4.23). Car ce qui peut suspendre l'exécution d'une tâche et l'emmener à la file d'attente est bien la non disponibilité de la ressource critique demandée. On note ici que la bascule de l'ordre des appels aux primitives de communications ne doit pas changer la sémantique fonctionnelle de la tâche.

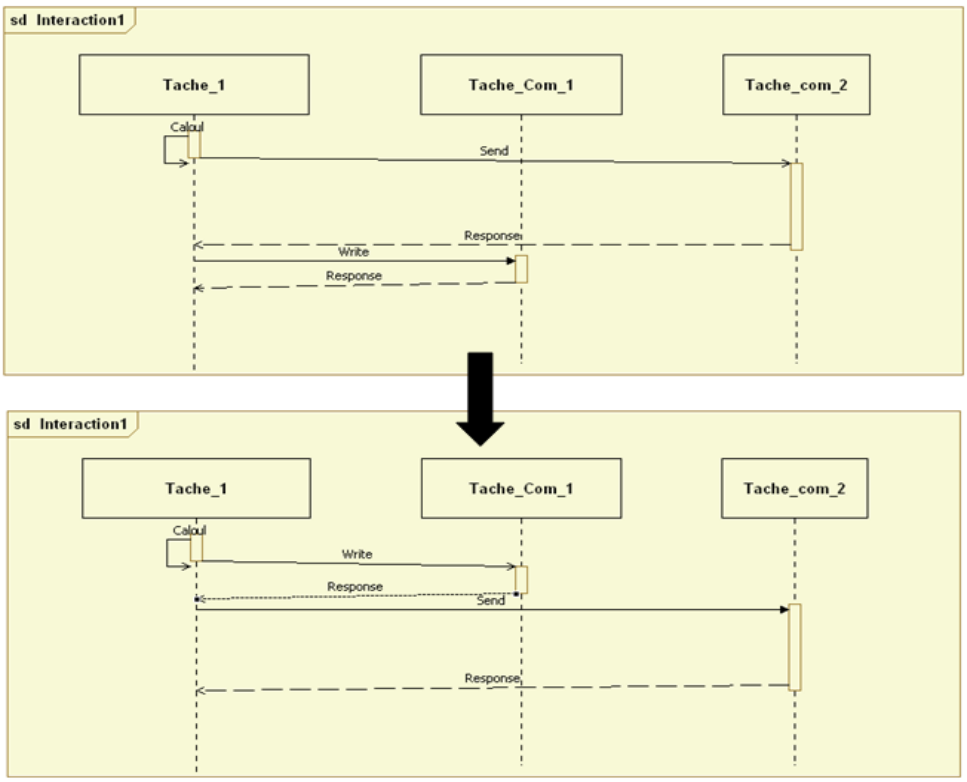

Figure 4.23. Changement d'ordre des appels aux primitives de communication.

4.7. Évaluation du Framework proposé sur un système temps réel utilisant deux techniques d'ordonnancement

4.7.1. Description du système

Ce système comporte :

- un seul processeur
- trois tâches périodiques
- deux ressources critiques

Les paramètres d'ordonnancabilité pour chaque tâche sont comme montre Tableau 4.3.

Tableau 4.3. Paramètres d'ordonnançabilité des tâches.

Tâche	Priorité	Echéance	période
Control_task	3	100	200
Planning_task	2	150	300
Status_task	1	350	700

4.7.2. Vue structurelle

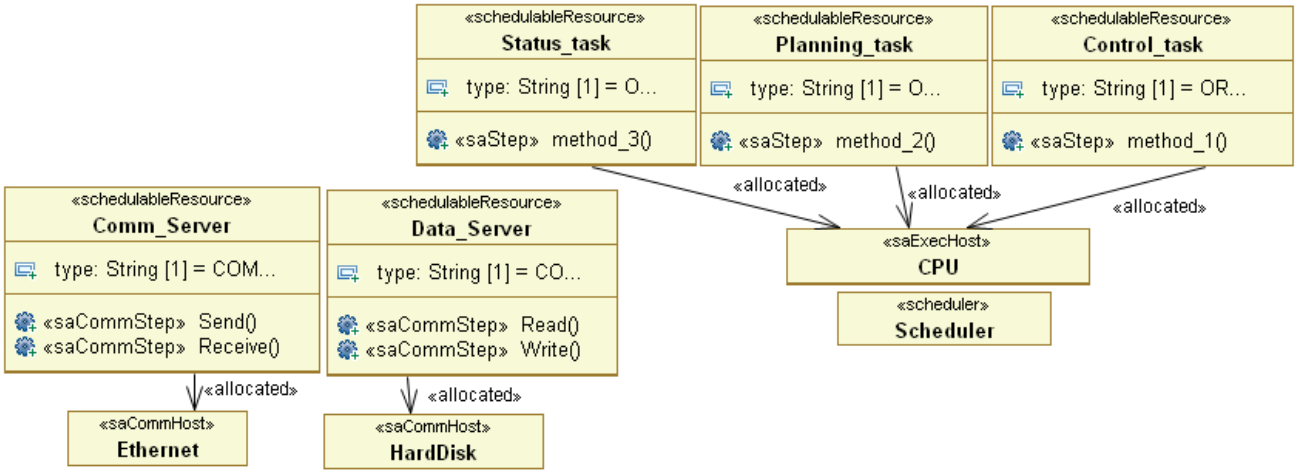

Figure 4.24. Vue structurelle du système utilisant le Framework proposé.

4.7.3. Vue comportementale

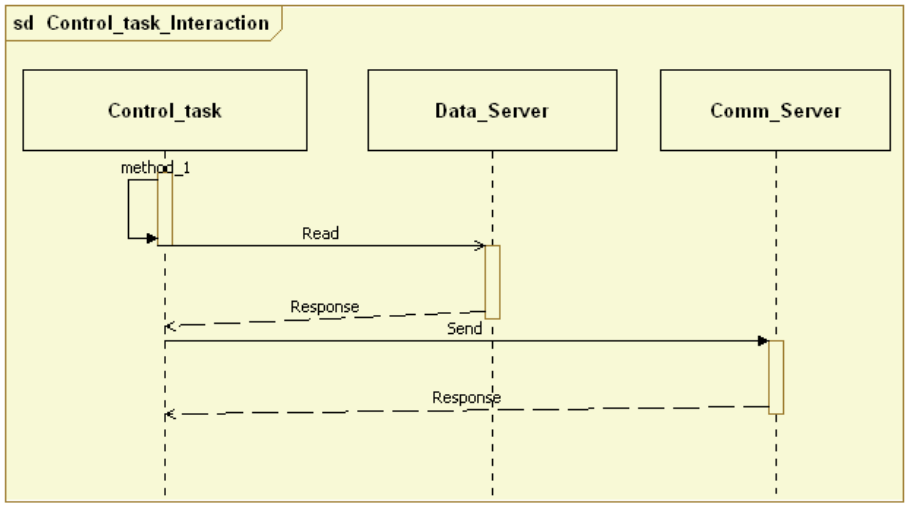

Figure 4.25. Vue comportementale de la tâche 'Control_task' utilisant le Framework proposé.

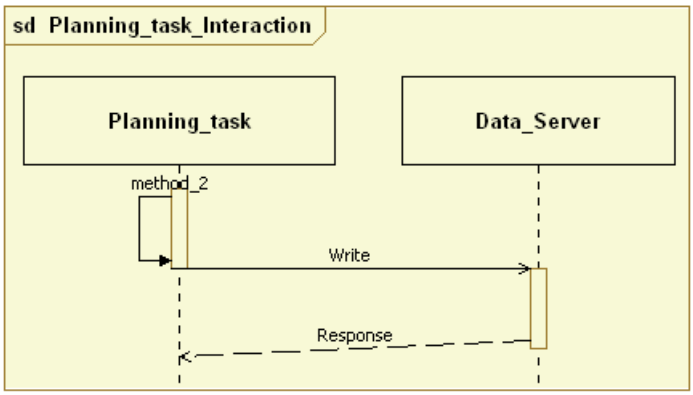

Figure 4.26. Vue comportementale de la tâche 'Planning_task' utilisant le Framework proposé.

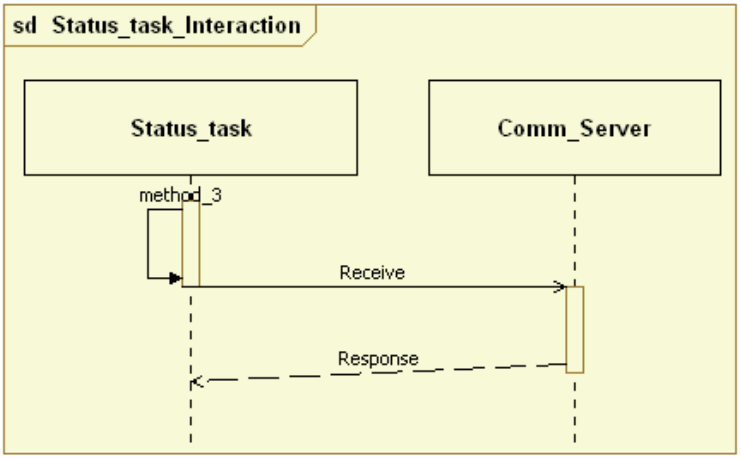

Figure 4.27. Vue comportementale de la tâche 'Status_task' utilisant le Framework proposé.

4.7.4. Conversion des modèles

4.7.4.1. Selon les paramètres d'ordonnancement de l'algorithme DM

```
Model (Model_Name  => DM_SCHEDULING_MODEL, Model_Date  => 2011-08-10);

-- Processing Resources
Processing_Resource (Type=> Fixed_Priority_Processor,Name=> CPU,
Worst_Context_Switch    => 0.25);

-- Scheduling Servers
Scheduling_Server (Type => Fixed_Priority, Name => Control_Task,
Server_Sched_Parameters => (Type => Fixed_Priority_policy,The_Priority => 3,
Preassigned => No), Server_Processing_Resource => CPU);

Scheduling_Server (Type => Fixed_Priority, Name => Planning_Task,
Server_Sched_Parameters => (Type => Fixed_Priority_policy,The_Priority => 2,
Preassigned => No), Server_Processing_Resource => CPU);

Scheduling_Server (Type => Fixed_Priority, Name => Status_Task,
Server_Sched_Parameters => (Type => Fixed_Priority_policy,The_Priority => 1,
Preassigned => No), Server_Processing_Resource => CPU);

-- Resources
Shared_Resource (Type   => Immediate_Ceiling_Resource, Name => Data_Server);

Shared_Resource (Type   => Immediate_Ceiling_Resource, Name => Comm_Server);

-- Operations
-- Critical Sections
Operation (Type => Simple,Name => Read, Worst_Case_Execution_Time => 2,
Shared_Resources_List => (Data_Server));

Operation (Type => Simple,Name => Write,Worst_Case_Execution_Time => 20,
Shared_Resources_List => (Data_Server));

Operation (Type => Simple,Name => Send, Worst_Case_Execution_Time => 10,
Shared_Resources_List => (Comm_Server));

Operation (Type => Simple,Name => Receive,Worst_Case_Execution_Time => 10,
Shared_Resources_List => (Comm_Server));

-- Task operations
```

```
Operation (Type => Enclosing,Name => Control,Worst_Case_Execution_Time => 20,
Composite_Operation_List  => (Read,Send));

Operation (Type => Enclosing,Name => Planning,Worst_Case_Execution_Time => 40,
Composite_Operation_List  => (Write));

Operation (Type => Enclosing, Name => Status,Worst_Case_Execution_Time => 100,
Composite_Operation_List  => (Receive));

-- Transactions
Transaction (Type => Regular, Name => Control_Task,External_Events => (
(Type => Periodic,Name  => E1, Period => 200)), Internal_Events => (
(Type => regular,name   => O1,Timing_Requirements =>(Type=> Hard_Global_Deadline,
Deadline => 100,Referenced_Event => E1))),Event_Handlers => (
(Type =>Activity , Input_Event => E1 , Output_Event => O1,
Activity_Operation => Control, Activity_Server    => Control_Task)));

Transaction (Type => Regular,Name   => Planning_Task,External_Events => (
(Type => Periodic, Name   => E2,Period => 300)),Internal_Events => (Type=>
regular,name=> O2,Timing_Requirements => (Type => Hard_Global_Deadline,
Deadline => 150,Referenced_Event => E2))),Event_Handlers =>((Type => Activity,
Input_Event => E2,Output_Event => O2, Activity_Operation => Planning,
Activity_Server    => Planning_Task)));

Transaction (Type => Regular, Name => Status_Task,External_Events =>(
(Type=> Periodic,Name=> E3,Period => 700)),Internal_Events => (
(Type=> regular,name=> O3,Timing_Requirements => (Type => Hard_Global_Deadline,
Deadline => 350,Referenced_Event => E3))),Event_Handlers => (
(Type => Activity, Input_Event => E3,Output_Event => O3,
Activity_Operation => Status, Activity_Server    => Status_Task)));
```

4.7.4.2. Selon les paramètres d'ordonnancement de l'algorithme EDF

```
Model (Model_Name  => EDF_SCHEDULING_Model, Model_Date  => 2011-08-10);

-- Processing Resources
Processing_Resource (Type=> Regular_Processor,Name=> CPU);

Scheduler (Type => Primary_Scheduler, Name  => CPU,Host  => CPU,
Policy =>(Type => EDF, Worst_Context_Switch => 0.25));

-- Scheduling Servers
Scheduling_Server (Type => Fixed_Priority, Name => Control_Task,
Server_Sched_Parameters => (Type => EDF_policy, Deadline    => 100,
Preassigned => No), Synchronization_Parameters => (
Type => SRP_Parameters,Preemption_Level => 2000,Preassigned => No),
Server_Processing_Resource    => CPU);

Scheduling_Server (Type=> Fixed_Priority, Name => Planning_Task,
Server_Sched_Parameters => (Type => EDF_policy, Deadline    => 150,
Preassigned => No),Synchronization_Parameters =>(Type => SRP_Parameters,
Preemption_Level => 2000,Preassigned => No),Server_Processing_Resource => CPU);
Scheduling_Server(Type  => Fixed_Priority, Name => Status_Task,
Server_Sched_Parameters => (Type => EDF_policy, Deadline=> 350,Preassigned => No),
Synchronization_Parameters =>(Type=> SRP_Parameters, Preemption_Level => 2000,
Preassigned => No),Server_Processing_Resource  => CPU);

-- Resources
Shared_Resource (Type   => SRP_Resource, Name => Data_Server);

Shared_Resource (Type   => SRP_Resource, Name   => Comm_Server);
```

```
-- Operations
-- Critical Sections
Operation (Type => Simple,Name => Read, Worst_Case_Execution_Time => 2,
Shared_Resources_List => (Data_Server));

Operation (Type => Simple,Name => Write, Worst_Case_Execution_Time => 20,
Shared_Resources_List     => (Data_Server));

Operation (Type => Simple,Name => Send, Worst_Case_Execution_Time => 10,
Shared_Resources_List => (Comm_Server));

Operation (Type => Simple,Name => Receive, Worst_Case_Execution_Time => 10,
Shared_Resources_List => (Comm_Server));

-- Task operations
Operation (Type => Enclosing, Name => Control, Worst_Case_Execution_Time => 20,
Composite_Operation_List  => (Read,Send));

Operation (Type => Enclosing, Name => Planning, Worst_Case_Execution_Time => 40,
Composite_Operation_List  => (Write));

Operation (Type => Enclosing, Name  => Status, Worst_Case_Execution_Time => 100,
Composite_Operation_List  => (Receive));

-- Transactions
Transaction (Type => Regular, Name => Control_Task, External_Events => (
(Type => Periodic, Name => E1, Period => 200)),Internal_Events => (
(Type => regular, name  => O1, Timing_Requirements =>(Type=>Hard_Global_Deadline,
Deadline => 100, Referenced_Event => E1))),Event_Handlers =>((Type => Activity,
Input_Event => E1,Output_Event => O1, Activity_Operation => Control,
Activity_Server => Control_Task)));

Transaction (Type => Regular,Name => Planning_Task,External_Events => (
(Type => Periodic,Name    => E2,Period => 300)),Internal_Events => (
(Type => regular, name  => O2,Timing_Requirements =>(Type=> Hard_Global_Deadline,
Deadline => 150,Referenced_Event => E2))),Event_Handlers =>((Type =>Activity,
Input_Event => E2,Output_Event => O2,Activity_Operation => Planning,
Activity_Server     => Planning_Task)));

Transaction (Type => Regular, Name => Status_Task, External_Events => (
(Type => Periodic, Name => E3, Period => 700)), Internal_Events => (
(Type => regular, name  => O3, Timing_Requirements=>(Type =>Hard_Global_Deadline,
Deadline => 350, Referenced_Event => E3))),Event_Handlers => (
(Type => Activity, Input_Event => E3, Output_Event => O3,
Activity_Operation => Status, Activity_Server => Status_Task)));
```

4.7.5. Résultats obtenus de l'analyse par les deux techniques d'ordonnancement

4.7.5.1. Première technique d'ordonnancement à priorité fixe: DM

Le système est ordonnancable selon la technique d'ordonnancement à priorité fixe DM où les trois tâches n'ont pas violé les contraintes temporelles (le pire temps de réponse est bien moins de l'échéance) où il y avait assez de marges de manœuvre pour les trois tâches, ceci avec un pourcentage d'utilisation du CPU de 38.11% comme montré dans Tableau 4.4.

Tableau 4.4. Résultat de l'analyse du système avec la technique DM

Tâche	Pire temps de réponse	Marge	Utilisation du CPU
Control_task	40.5	>=10000%	
Planning_task	71.0	296.88%	38.11%
Status_task	161.0	694.53%	

4.7.5.2. Deuxième technique d'ordonnancement à priorité dynamique: EDF

Le système est aussi ordonnancable selon la deuxième technique d'ordonnancement à priorité dynamique EDF où les trois tâches ont respecté les contraintes temporelles (le pire temps de réponse est bien moins de l'échéance) comme montré dans Tableau 4.5. Comparant les résultats obtenus de cette technique avec la première technique, on trouve une petite variation où on a enregistré pour la troisième tâche'Status_task' un pire temps de réponse de 181.5 qui est plus grand par rapport au premier pire temps de réponse qui était de 161.0.

Tableau 4.5. Résultat de l'analyse du système avec la technique EDF.

Tâche	Pire temps de réponse	Marge	Utilisation du CPU
Control_task	40.5	789.84%	
Planning_task	71.0	296.88%	38.11%
Status_task	181.5	694.53%	

4.8. Conclusion

En conclusion, l'utilisation du profil UML pour MARTE présente une spécialisation d'UML pour les systèmes temps réel. D'après ce chapitre on peut dire qu'on fait une petite exploration de ce profil de point de vue analyse d'ordonnancabilité, ceci en proposant un Framework basé MARTE pour l'analyse d'ordonnancabilité via l'outil d'analyse MAST.

Conclusion générale

La nature critique des systèmes temps-réel et embarqués nécessite qu'ils soient prédictibles et que leurs architectures logicielles et matérielles soient spécifiés sans ambiguïté, afin qu'il soit possible de s'assurer, en plus de sa correction fonctionnelle, de la correction temporelle du système. La criticité des systèmes temps-réel et embarqués fait du processus de leurs développement, une tâche délicate.
L'analyse de l'ordonnançabilité joue un rôle principal dans la conception de ce genre de systèmes, une fois le concepteur définit les valeurs des paramètres du système, l'analyse de l'ordonnançabilité est effectuée afin de déterminer s'il est possible de satisfaire les différentes contraintes imposée par le système.
La conception des systèmes temps-réel et embarqués repose sur des modèles représentant le comportement logiciel ainsi la plateforme matérielle. Ces modèles doivent permettre aux différents outils d'analyse de l'ordonnançabilité de déterminer si le système conçu respecte ou pas les contraintes temporelles spécifiées, ainsi de permettre la possibilité de faire les modifications nécessaires, en cas d'échec de l'analyse en première itération, afin que le système puisse satisfaire toutes ses contraintes.
Grâce aux mécanismes du profil MARTE, la modélisation des systèmes temps-réel, et la spécification des besoins temporels et de synchronisation sont améliorés; se basant sur ce profil, on peut définir des modèles sans ambiguïté sur lesquels il est possible d'effectuer l'analyse de l'ordonnançabilité temps-réel, ainsi, la modification de la modélisation est de départ faisable suite aux résultats enregistrés par les outils de vérification.

On a élaboré un modèle d'analyse d'ordonnançabilité, en redéfinissant des éléments du framework MARTE, modélisant toutes les informations nécessaires pour effectuer une analyse de l'ordonnançabilité, en les adaptant aux besoins de l'outil MAST, ces informations sont fournies comme entrée à l'outil de l'analyse de l'ordonnançabilité, ensuite, les résultats de l'analyse peuvent être utilisés afin d'améliorer le modèle original. La contribution est axée autour :

- La reprise du Framework général de traitement de MARTE pour le cas de l'analyse d'ordonnançabilité en proposant un schéma qui comprend :
  - **Editeur UML/MARTE :** qui se matérialise par la proposition d'un ensemble des éléments de modèles (exploitant un éditeur open source : Papyrus) nécessaires à l'analyse d'ordonnançabilité qui comporte deux vues ; une vue structurelle (éléments logiciels et matériels) et une autre comportementale.
  - **Convertisseur des modèles UML :** de leur format XMI vers le format des fichiers d'entrée de l'outil d'analyse de l'ordonnançabilité MAST(Modeling and Analysis Suite for real-Time applications). Il est développé en Java.
  - **Analyse des résultats :** elle regroupe quelques réflexions au niveau conceptuel, aidant à réagir suite aux résultats obtenus de la part de l'outil d'analyse.
- L'évaluation du Framework proposé sur des modèles de tâches des systèmes temps en variant les politiques d'ordonnancement avec l'interprétation des résultats.

Le travail présenté dans ce mémoire peut être complété par un enrichissement de modèles des domaines proposés afin d'améliorer, particulièrement, la représentation des aspects temps et concurrence; D'autres propositions de modifications qui peuvent être effectuées sur les modèles qui ne réussissent pas à l'analyse de l'ordonnançabilité, peuvent enrichir le travail. Un travail pareil peut être effectué, en adaptant les modèles de domaines aux besoins d'un autre outil d'analyse qui supporte une représentation de l'architecture matérielle plus détaillée que celle proposée par MAST afin de prendre en considération autre type de contraintes liées à l'architecture matérielle des systèmes temps-réel. Le travail reste ouvert à d'autres perspectives.

Bibliographie

[ACLS04] Ludivic Apvrille, Jean-Pierre Courtiat, Christophe Lohr, and Pierre de Saqui-Sannes. TURTLE: A Real-Time UML Profile Supported by Formal ValidationToolkit. IEEE Trans. Software Eng., 30(7):473–487, 2004.

[AMS07] C. André, F. Mallet and R. de Simone: "Time Modeling in MARTE", in proceeding of Forum on specification & Design Languages (FDL '07), Barcelona, Spain, September 2007.

[ASK05] Ludivic Apvrille, Pierre de Saqui-Sannes, and Ferhat Khendek. TURTLE-P:A UML Profile for the Formal Validation of Critical and Distributed Systems.To appear in Software and Systems Modeling, Springer 2005.

[Ber03] Kirsten Berkenkotter. Using UML 2.0 in Real-Time Development: A Critical Review. In SVERTS, pages 41–54, San Francisco, USA, October 2003.

[BK03] M. Björkander, C. Kobryn: "Architecting Systems with UML 2.0". IEEE Computer Society, IEEE Software. 2003.

[BM05] Simonetta Balsamo and Moreno Marzolla. Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models. In Fifth International Workshop on Software and Performance, Palma, Illes Balears, Spain, July 2005.

[BMIS04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni.Model-Based Performance Prediction in Software Development: A Survey.IEEE Trans. Software Eng., 30(5):295–310, 2004.

[BP04] Simona Bernardi and Dorina C. Petriu. Comparing two UML Profiles for Nonfunctional Requirement Annotations: the SPT and QoS Profiles. In SVERTS, Lisbone, Portugal, October 2004.

[BRS02] L. Bichler, A. Radermacher, A. Schür John Smith: "Evaluating UML Extensions for Modeling Real-Time Systems", In Proceedings of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002). Los Alamitos, CA, USA, IEEE Computer Society 2002, pp. 271-8 Conference Paper.

[CDKM02] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri: "Scheduling in real-time systems". John Wiley & Sons. 2002.

[CDKM99] Francis COTTET & Joëlle DELACROIX & Claude KAISER & Zubir MAMMERI. Ordonnancement temps réel (ordonnancement centralisé). *Techniques de l'Ingénieur, traité informatique industrielle S 8 055*, 28 :1–22, 1999.

[Che02] Albert M. K. CHENG. Real-time systems Scheduling, Analysis, and Verification, chapter Analysis and verification of non-real-time systems. 2002.

[Dec02] David Decotigny. Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps réel,thèse, 2002.

[dM03] Miguel A. de Miguel. General Framework for the Description of QoS in UML. In IEEE Computer Society, editor, Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03), pages 61–70, Hakodate, Hokkaido, Japan, May 2003.

[DMS99] C. Duvallet, Z. Mammeri et B. Sadeg, « Analyse des protocoles de contrôle de concurrence et des propriétés ACID dans les SGBD temps réel », Revue TSI, 1999.

[Dou98a] B. P. Douglass: "Real-Time UML.Developing Efficient Objects for Embedded Systems". Addison-Wesley. 1998.

[Dou98b] B. P. Douglass: "Designing Real-Time Systems with UML". Embedded Systems Programming. 1998.

[DP91] Alain DORSEUIL, Pascal Pillot. Le temps réel en milieu industriel, série Informatique Industrielle, Paris, 1991.

[Fau03] Sébastien Faucou. Description et construction d'architectures opérationnelles validées temporellement. Mars 2003.

[FM02] Stephan Flake and Wolfgang Mueller. A UML profile for real-time constraints with the OCL. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings, volume 2460 of LNCS, pages 179–195. Springer, 2002.

[GH05] Zonghua Gu and Zhimin He. Real-Time Scheduling Techniques for Implementation Synthesis from Component-Based Software Models. In Accepted to ACM SIGSOFT International Symposium on Component-Based Software Engineering (CBSE 2005), St. Louis, MO, 2005.

[GMSB96] Marie-Claude Gaudel, Bruno Marre, Françoise Schlienger et Gilles Bernot . – Précis de génie logiciel. – Masson, 1996.

[GO04] Susanne Graf and Ileana Ober. How useful is the UML profile SPT without semantics. In SVERTS, Lisbone, Portugal, October 2004.

[Gom00] H. Gomaa: "Designing Concurrent, Distributed, and Real-Time Applications with UML". Addison-Wesley. 2000.

[GOO03] Susanne Graf, Ileana Ober, and Iulian Ober. Timed annotations with UML. In SVERTS, San Francisco, USA, October 2003.

[GS04] Z. Gu and K. G. Shin. Synthesis of Real-Time Implementation from UML-RT Models. In 2nd RTAS Workshop on Model-Driven Embedded Systems (MoDES'04), Toronto, Ontario, Canada, May 2004.

[GT03] S. Gérard and F. Terrier. UML for Real: Which Native Concepts to Use, chapter UML for Real-Time, pages 17–51. Kluwer Academic Publishers, 2003.

[HH08] M. Hagner and M. Huhn: "Tool Support for a Scheduling Analysis View", in proceeding of Design, Automation and Test in Europe (DATE'08), Munich, Germany, 10-14 March 2008.

[Hla08] Pierre-Emmanuel Hladik. Partie II : Analyse d'ordonnançabilité. INSA de Toulouse / LAAS-CNRS, 2008.

[Itu00] ITU-T. ITU-T recommandation Z.109, SDL combined with UML. 2000.

[JMEP00] R. Jigorea, S. Manolache, P. Eles, Z. Peng: "Modelling of Real-Time Embedded Systems in an Object-Oriented Design Environment with UML". 2nd ARTES Graduate Student Conference. Chalmers University of Technology, Göteborg (Sweden). March 2000.

[KCH01] Saehwa Kim, Sukjae Cho, and Seongsoo Hong. Automatic Implementation of Real-Time Object-Oriented Models and Schedulability Issues. In Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), pages 137–141,Rome, Italy,2001.

[KCH00] Saehwa Kim, Sukjae Cho, and Seongsoo Hong. Schedulability-aware Mapping of Real-Time Object-Oriented Models to Multi-Threaded Implementations. In 7th International Workshop on Real-Time Computing and Applications Symposium(RTCSA 2000), 12-14 December 2000, Cheju Island, South Korea, pages 7–14. IEEE Computer Society, 2000.

[Kru95] P. Kruchten. The 4+1 view model of architecture. IEEE Softw., 12(6):42–50, 1995.

[KS01] J. M. Küster, J. Stroop: "Consistent Design of Embedded Real-Time Systems with UML-RT". 4th IEEE International Symposium on Object Oriented Real-Time Distributing Computing ISORC 2001. Los Alamitos, CA, USA, IEEE Computer Society 2001, pp. 31-40 Conference Paper.

[KWS03] Sharath Kodase, Shige Wang, and Kang G. Shin. Transforming Structural Model to Runtime Model of Embedded Software with Real-Time Constraints. In Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany, pages 20170–20175. IEEE Computer Society, 2003.

[Len99] Jacques Lenoir – Les outils de conception système du logiciel enfoui. Electronique, no89, Février 1999.

[LL73] C. L. Liu, J. W. Layland: "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". Journal of the ACM, vol. 20, no. 1, pp. 46-61, January 1973.

[LQV01] L. Lavazza, G. Quaroni, M. Venturelli: "Combining UML and Formal Notations for Modelling Real-Time Systems". Software Engineering Notes, vol. 26, no. 5, pp. 196-206, September 2001.

[Mar03] P. Martins: “Integrating Real-Time UML Models with Schedulability Analysis”. White Paper in Tri Pacific Software Products 2003.

[MWS+05] Jeffrey R. Merrick, Shige Wang, Kang G. Shin, Jing Song, and William Milam. Priority Refinement for Dependent Tasks in Large Embedded Real-Time Software. In 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’05). IEEE Computer Society, 2005.

[Ome11] OMEGA ST. Project. http://www-omega.imag.fr/, consulté en juin 2011.

[Omg03] OMG, “Unified Modeling Language Specification (version 2.0): Diagram Interchange”, 2003.

[Omg04] OMG. UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms. OMG Adopted Specification, ptc/2004-06-01, June 2004.

[Omg05] OMG: “UML Profile for Schedulability, Performance, and Time Specification,” Version 1.1, formal/05-01-02; January 2005.

[Omg07] OMG: “A UML Profile for MARTE”, OMG Adopted Specification, Version Beta 1, OMG Document Number: ptc/07-08-04, August 2007.

[Omg09] OMG: “A UML Profile for MARTE”, OMG Adopted Specification, Version 1.0, OMG Document Number: ptc/2009-05-15, formal/2009-11-02.

[Pai06] Stéphane PAILLER. Analyse Hors Ligne d’Ordonnançabilité d’Applications Temps Réel comportant des Tâches Conditionnelles et Sporadiques. *Équipe Temps Réel du LISI*. Octobre 2006.

[Per90] J.P. Perez – Systèmes Temps Réel: Méthodes de Spécification et de Conception. – DUNOD, 1990.

[Rtl98] RTL, Real-time LOTOS, Software and Tools for Communicating Systems. http://www.laas.fr/RTLOTOS/, 1998.

[SGW94] B. Selic, G. Gullekson, and P.T. Ward. Real-Time Object-Oriented Modeling. John Wiley and Sons, 1994.

[SK00] Manas Saksena and Panagiota Kervelas. Designing for Schedulability: Integrating Schedulability Analysis with Object-Oriendted Design. In The 12th Euromicro Conference on Real-Time Systems, June 2000.

[SKW00] M. Saksena, P. Karvelas, and Y.Wang. Automated Synthesis of Multi-Tasking Implementations from Real-Time Object-Oriented Models. In 3rd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000), pages 360–367, Newport Beach, CA, USA, March 2000. IEEE Computer Society.

[SR04] H. Saiedian, S. Raguraman: “Using UML-Based Rate Monotonic Analysis to Predict Schedulability”. IEEE Computer Society, vol. 37, no. 10, pp. 56-63, October 2004.

[SR98] B. Selic, J. Rumbaugh: “Using UML for Modeling Complex Real-Time Systems”. White Paper, published by ObjecTime Limited, 340 March Rd., Kamata, Astario, Canada. 1998.

[TE99] Yvon TRINQUET & Jean-Pierre ELLOY. Systèmes d’exploiataion temps réel(principes). *Téchniques de l’Ingénieur, traité mesure et contrôle R 8 050*, 24 :1–11, 1999.

[Tel11] Telelogic. Telelogic Products. http://www.telelogic.com/products/, consulté en juin 2011.

[Tto11] TTool. A Toolkit for Editing and Validating TURTLE Diagrams. http://www.eurecom.fr/apvrille/TURTLE/index.html, consulté en juin 2011.

[WP04] C.M.Woodside and D.C. Petriu. Capabilities of the UML Profile for Schedulability Performance and Time (SPT). In Workshop SIVOES-SPT held in conjunction with the 10th IEEE RTAS’2004, Toronto, Canada, May 2004.

[XWP03] Jing Xu, C.M.Woodside, and D.C.Petriu. Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time. In P.Kemper and W.Sanders, editors, Proc. of 13th International Conference on Modelling Techniques and Tools for Computer Performace Evaluation – Performance TOOLS’2003, volume 2794 of LNCS, pages 291–307, Sept 2003.

Mémoire : Analyse De L'ordonnançabilite Sur Des Modèles UML Temps-Reel

Nom: ACHI Prénom: LEMYA Encadreur: Mr. Mohammed Mostefai

Résumé :

L'utilisation des outils actuels de vérification et de simulation de l'ordonnançabilité des systèmes permet la validation de l'ordonnancement des systèmes. L'étude de l'analyse d'ordonnançabilité est judicieuse surtout quand les systèmes ne tolèrent pas de fautes. La non prise en compte des aspects de l'analyse d'ordonnançabilité dans les étapes précoces du cycle de développement (la conception) de ces systèmes, rend la mise en cause du modèle plus difficile et onéreuse surtout quand l'ordonnancement est irréalisable. Grâce aux mécanismes d'extension d'UML, le profil MARTE permet de prendre en compte les aspects particuliers des systèmes temps réel et embarqué, ainsi les modèles UML peuvent être annotés par des informations supplémentaires ce qui rend la modification de la modélisation de départ faisable suite aux résultats aboutis par les outils de vérification. Le but de ce travail consiste à fournir un framework basé MARTE permettant aux concepteurs des systèmes Temps-réel de modéliser leurs systèmes afin d'analyser leurs ordonnançabilités grâce à l'outil d'analyse MAST.

Mots clés : Systèmes temps-réel et embarqués, MARTE, Analyse de l'ordonnançabilité temps-réel

---

المذكرة : تحليل قابلية الجدولة في نماذج UML الوقت الحقيقي

اللقب: عشي الإسم: لمياء المؤطر: الأستاذ محمد مصطفاي

ملخص:

استخدام الأدوات الحالية للتحقق ولمحاكاة قابلية جدولة النظم يسمح بإثبات صحة تقنيات جدولة الأنظمة. تعتبر دراسة تحليل قابلية جدولة مهمة خصوصا إذا تعلق الأمر بصنف الأنظمة التي لا تتسامح مع الأخطاء.عدم الأخذ بعين الاعتبار جوانب تحليل قابلية جدولة النظم في المراحل المبكرة من دورة الإنشاء (التصميم) لهذه النظم يجعل تصحيح النموذج أكثر صعوبة وتكلفة وخصوصا عندما تكون تقنية الجدولة غير ممكن عمليا. مع التوسع في جانب UML الذي يسمى MARTE أصبح بالإمكان الأخذ بعين الاعتبار الجوانب الفريدة الخاصة بأنظمة الوقت الحقيقي والمدمجة، ونماذج UML يمكن إثراؤها بمعلومات إضافية مما يجعل تغيير النموذج الأولي ممكنا وذلك بحسب النتائج التي تستخلصها وسائل التحقق. الهدف من هذا العمل هو تزويد هيكل مبني على MARTE، الهيكل يسمح لمصممي أنظمة الوقت الحقيقي بتحقيق نماذج لأنظمتهم من أجل تحليل قابليتها للجدولة باستخدام وسيلة التحليل MAST.

الكلمات مفتاحية : أنظمة الوقت الحقيقي والمدمجة ، MARTE، تحليل قابلية جدولة نظم الوقت الحقيقي

Statement :schedulability analysis on Real-time UML models

Name : ACHI First name : Lemya Directed by : Mr. Mohammed MOSTEFAI

Abstract:

The use of current tools for schedulability verification and simulation allows the validating of systems scheduling. The study of the schedulability analysis is appropriate especially when the systems do not tolerate mistakes. Missing the aspects of schedulability analysis in the early stages of the development cycle (design) of these systems makes the implication of the model more difficult especially when the scheduling politic was unfeasible. With UML profile MARTE, it is possible to specify the especial aspects of real-time and embedded systems, thus UML models can be annotated with additional information which permit to change the starting model according to the results have been achieved by the verification tools. The aim of this work is to provide designers of Real-Time systems with a framework based on MARTE which allows them to design their systems in order to analyze their schedulability using the analysis tool MAST.

Key words: Real-time and embeded systems, MARTE, Real time schedulability analysis