Un processus d'ingénierie de Cheddar pour la simulation de systèmes temps réel à grande échelle

Alain Plantec et Frank Singhoff

LISyC/EA 3883, Université of Brest, 20 av Le Gorgeu
CS 93837, 29238 Brest Cedex 3, France
{plantec,singhoff}@univ-brest.fr

Résumé : La théorie de l'ordonnancement temps réel fournit des méthodes algébriques et algorithmiques permettant de vérifier le comportement temporel d'une application temps réel. Bien que mature pour certains contextes d'utilisation, cette théorie demeure relativement peu appliquée par l'industrie. Le projet Cheddar se propose d'évaluer l'applicabilité de la théorie de l'ordonnancement temps réel. Cet article présente les contributions de ce projet à l'analyse de performances par simulation d'architectures à grande échelle. Dans le cadre de leurs travaux, les auteurs développent l'environnement Cheddar qui permet de modéliser un système et d'y appliquer les résultats de la théorie de l'ordonnancement. Pour les applications spécifiques pour lesquels il n'existent pas de résultats analytiques, Cheddar propose un langage spécifique qui, basé sur les automates temporisés, permet à l'utilisateur d'étudier son application par le biais de simulations. Des expérimentations ont montré une adéquation satisfaisante du langage pour la modélisation d'une large variété d'ordonnanceurs temps réel. Toutefois, les expérimentations ont aussi démontré que l'efficacité de ce simulateur devait être améliorée afin de pouvoir traiter d'applications industrielles de grande taille. Dans cet article, les auteurs présentent le procédé d'ingénierie dirigée par les modèles exploité pour la mise en œuvre de Cheddar afin d'en améliorer l'efficacité. Pour optimiser les performances, la solution mise en œuvre consiste à produire une version spécialisée de Cheddar incluant un nouvel ordonnanceur à analyser : à partir de son modèle d'ordonnanceur, un composant spécifique est généré. Ce composant est alors intégré et une nouvelle version de Cheddar est recompilée. L'outillage mis en œuvre pour la génération de code s'appuie sur la modélisation formelle des composants internes de Cheddar ainsi que sur la syntaxe abstraite du langage de Cheddar. Cet outillage est mis en œuvre avec Platypus, un méta-environnement basé sur l'utilisation de la technologie STEP, notamment le langage de modélisation EXPRESS.

Mots clés : Ingénierie dirigée par les modèles, théorie de l'ordonnancement temps réel, simulateur, langage spécifique, AADL.

1. INTRODUCTION

La théorie de l'ordonnancement temps réel fournit des méthodes algébriques (ou tests de faisabilité) et algorithmiques permettant de vérifier le comportement temporel d'une application temps réel. Les bases de cette théorie ont été proposées par Liu et Layland dans les années 1970 [18] et ont, depuis, fait l'objet de très nombreuses recherches et expérimentations [6,7,17]. Toutefois, dans de très nombreux cas, la théorie de l'ordonnancement temps réel reste inappliquée par le monde industriel malgré un intérêt certain pour ce type de méthodes. Plusieurs hypothèses peuvent être avancées pour expliquer cet état de fait : l'inadéquation de cette approche dans certains contextes tels que les systèmes répartis pour lesquels peu de résultats théoriques ont été proposés, l'absence d'outil flexible et ouvert, le faible couplage avec les méthodes de conception ou les processus d'ingénierie, ...

Le projet Cheddar se propose d'évaluer l'applicabilité de la théorie de l'ordonnancement temps réel en étudiant certaines de ces hypothèses. Cet article présente les contributions de ce projet à l'analyse de performances par simulation d'architectures à grande échelle. En effet, de nombreuses applications industrielles ne peuvent pas être analysées par les méthodes analytiques de la théorie de l'ordonnancement. D'une part, la théorie de l'ordonnancement temps réel ne fournit pas nécessairement de tests de faisabilité pour les ordonnanceurs et les modèles de tâches employés par le monde industriel. D'autre part, élaborer ces tests de faisabilité est une activité coûteuse et généralement difficile. Dans ce contexte, les systèmes à analyser sont généralement de grande taille, de sorte qu'il est difficile d'employer des méthodes de *model-checking* [33].

Le monde industriel est donc souvent réduit à vérifier certains de ses systèmes par la simulation. Il existe de nombreux modèles à événements discrets qui offrent des outils de simulation permettant de réaliser ces études de performances. C'est notamment le cas des réseaux de Petri [22] et d'outils tels que CPN Tools [32]. Une autre

solution consiste à développer des outils de simulations *ad hoc*. Cette solution est coûteuse et n'offre, en général, qu'un faible niveau de réutilisation logicielle. L'environnement Cheddar propose une alternative par la mise à disposition d'un langage spécifique ainsi que des outils associés (interpréteur, compilateur,...). Ce langage permet de modéliser un ordonnanceur temps réel et d'en étudier le comportement par simulation. Nous proposons un processus d'utilisation associé à ce langage afin que l'utilisateur puisse concevoir et tester son ordonnanceur. Par la suite, grâce à l'utilisation d'un outil meta-CASE, l'utilisateur peut générer son outil de simulation, qui, après compilation, est capable de conduire des simulations sur des modèles de taille importante. Ce processus d'ingénierie dirigée par les modèles est mis en œuvre grâce à Platypus, un méta-environnement basé sur l'utilisation de la technologie STEP, notamment le langage de modélisation EXPRESS [10,13].

Dans la suite de cet article, nous présentons ce processus d'ingénierie dirigée par les modèles. Dans le chapitre 2, nous décrivons l'environnement Cheddar ainsi que son langage de modélisation. Cette présentation est illustrée par un modèle d'ordonnanceur hiérarchique. Le chapitre 3 décrit comment il est possible de mettre en œuvre les différents outils associés au langage de modélisation de Cheddar à partir d'un modèle d'ordonnanceur. Enfin, nous concluons et dressons quelques perspectives dans le chapitre 4.

2. L'ENVIRONNEMENT DE SIMULATION CHEDDAR

Cheddar est un canevas constitué d'un ensemble de paquetages Ada et dont l'objectif est d'offrir à l'utilisateur des outils pour l'analyse de performances d'applications temps réel [29]. Ce canevas implante les outils analytiques et les algorithmes d'ordonnancement temps réel classiques. Les systèmes à analyser peuvent être décrits, soit dans un formalisme propriétaire, soit par une spécification AADL (AADL pour Architecture Analysis and Design Language). AADL est un langage graphique et textuel normalisé par la SAE sous le standard AS-5506 [9]. Il permet la description de l'architecture matérielle et logicielle d'un système temps réel embarqué.

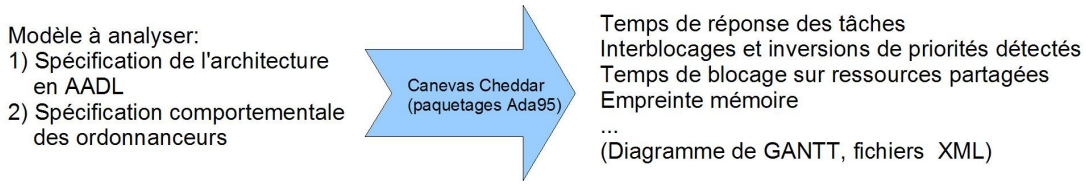

Figure 1 : Fonctionnement général du canevas Cheddar

La figure 1 décrit succinctement l'utilisation de Cheddar lorsque l'on souhaite effectuer une analyse de performances :

1- Il est d'abord nécessaire de modéliser l'architecture que l'on souhaite analyser. Cette spécification peut être exprimée en AADL. Il s'agit alors de décrire le système en termes de composants tels que des processeurs, des tâches ou des ressources partagées. Chaque composant est décoré de propriétés décrivant son comportement logique et temporel. Ainsi, il est possible de spécifier la période d'une tâche, sa capacité, le type d'ordonnanceur associé à un processeur donné. Bien qu'il existe des propriétés standardisées par AADL, Cheddar propose un certain nombre d'extensions autorisant l'application des méthodes d'analyse proposées par la théorie de l'ordonnancement temps réel [37]. Cheddar offre un éditeur simplifié permettant de décrire l'architecture à analyser. Néanmoins, la démarche naturelle consiste à utiliser un outil de modélisation pour l'édition d'un tel modèle. À ce jour, des possibilités d'interopérabilité entre Cheddar et des outils de modélisation tel que Stood [35] existent. Stood est un outil de conception d'applications temps réel qui se distingue par le support d'une approche méthodologique rigoureuse et une compatibilité avec le standard AADL, lequel permet d'élaborer des chaînes d'outils très efficaces.
2- L'utilisateur peut alors utiliser deux types de services offerts par le canevas Cheddar : la simulation ou le calcul de tests de faisabilité. Le choix entre l'un, l'autre ou les deux types d'outils dépend des caractéristiques du modèle à analyser. Après analyse, les résultats peuvent être soit affichés dans l'éditeur de Cheddar, soit exportés vers d'autres outils via des fichiers XML.

Le canevas Cheddar offre un langage spécifique ainsi qu'un ensemble d'outils associés pour la modélisation et la mise en œuvre d'algorithmes d'ordonnancement temps réel. L'utilisation de ce langage permet à l'utilisateur de définir un algorithme et de le tester.

Il existe donc plusieurs niveaux d'utilisation du canevas Cheddar :

- Lorsque le modèle AADL à analyser comporte des algorithmes d'ordonnancement classiques et déjà implantés dans le canevas, l'utilisateur exploite l'interface homme-machine pour éditer son modèle et en effectuer une analyse de performances. Cette alternative est celle qui demande le moins d'expertise de la part de l'utilisateur quant à l'utilisation de Cheddar.
- Si l'algorithme n'existe pas dans le canevas Cheddar, mais qu'il peut être modélisé par le langage spécifique de Cheddar, l'utilisateur doit décrire son algorithme d'ordonnancement en plus du modèle AADL à analyser. L'utilisateur doit alors maîtriser le langage de modélisation d'ordonnanceurs temps réel.
- Enfin, si l'algorithme et/ou le système à analyser sont trop spécifiques, l'utilisateur doit modifier le canevas Cheddar afin d'implanter ses algorithmes d'ordonnancement temps réel. Cette dernière alternative est la plus coûteuse pour l'utilisateur, car il lui est alors nécessaire de comprendre et de modifier l'architecture ainsi que la mise en œuvre du canevas. L'existence du langage spécifique a pour objet d'éviter, autant que possible, la modification directe du canevas par l'utilisateur afin que l'utilisation du canevas reste la plus simple possible.

2.1. Un langage pour la modélisation d'ordonnanceurs temps réel

Le langage spécifique de Cheddar doit permettre de modéliser le comportement d'un ordonnanceur temps réel. Ceci consiste à modéliser :

1- Les opérations arithmétiques et logiques de l'ordonnanceur, qui décrivent, par exemple, comment calculer la priorité des tâches. Il s'agit essentiellement de traitements à effectuer sur les attributs de certaines entités telles que les tâches, les processeurs ou les ressources partagées du système à analyser.
2- Les contraintes temporelles et les synchronisations entre des entités telles que les tâches et les ordonnanceurs de l'architecture à analyser. Il s'agit ici d'exprimer comment et quand ces entités doivent collaborer afin de partager les différentes ressources.

Le langage Cheddar est donc défini par l'association de deux langages spécifiques. La partie algorithmique est exprimée par un sous-ensemble d'Ada. Les contraintes temporelles et de synchronisation sont exprimées par un modèle d'automates temporisés. La syntaxe complète de ce langage est décrite dans le guide d'utilisation de Cheddar [28].

2.1.1. Un sous-ensemble d'Ada

Ce langage permet l'expression des opérations arithmétiques et logiques sur les données de simulation. Ces dernières sont associées aux entités du modèle à analyser (ex : attribut d'une tâche, d'un processeur). Ainsi, il permet d'exprimer des règles de tri [21] telles que Earliest Deadline par exemple. Les instructions arithmétiques et logiques sont organisées en sous-programmes appelés *section*. Ces sous-programmes sont typés. Il en existe trois types :

- Des sous-programmes qui permettent la déclaration et l'initialisation de nouvelles données de simulation. Ils sont appelés *start_section*.
- Des sous-programmes qui permettent l'exécution d'opérations arithmétiques et logiques sur des données pendant la simulation. Ils sont appelés *priority_section*.
- Enfin, des sous-programmes qui permettent de sélectionner une tâche parmi un ensemble de tâches prêtes en fonction d'une donnée de simulation. Ils sont appelés *election_section*.

Le langage définit un certain nombre de types, d'opérateurs et d'instructions. On y retrouve des instructions et opérateurs classiques tels que l'itération, la conditionnelle ou l'affectation. D'autres instructions ou opérateurs sont plus spécifiques au domaine de l'ordonnancement temps réel tels que *return, lcm*, *max_to_index* ou *uniform/exponential*. Ainsi, l'instruction *return* permet d'élire une tâche conformément à une donnée de simulation particulière. Les instructions *uniform/exponential* permettent de paramétrer comment le simulateur doit évaluer les variables aléatoires nécessaires à la simulation. L'opérateur *lcm* permet de calculer le plus petit commun multiple (opérateur usuel dans les tests de faisabilité). Enfin, l'opérateur *max_to_index* recherche la plus grande valeur dans un tableau stockant une donnée de simulation par tâche. Il permet d'implanter la recherche de la plus grande priorité des tâches de l'architecture à analyser par exemple. Le langage de Cheddar est typé. Les types usuels tels que *integer*, *boolean*, *double* ou *string* sont disponibles. Des types relatifs au domaine de l'ordonnancement temps réel sont aussi disponibles.

2.1.2. Un langage basé sur les automates temporisés

La deuxième partie d'un modèle d'ordonnanceur temps réel dans l'environnement Cheddar est un ensemble d'automates temporisés modélisant les contraintes temporelles et les synchronisations entre les entités de l'architecture modélisée.

Le langage décrit dans le paragraphe 2.1.1 est suffisant pour exprimer les ordonnanceurs pour lesquels les contraintes de synchronisation entre les entités sont fixes. Par le passé, nous avons montré qu'il était possible de modéliser des algorithmes simples tels que Earliest Deadline First, Rate Monotonic ou Maximum Urgency First [6].

Il existe toutefois des ordonnanceurs qui nécessitent l'expression de synchronisations plus complexes. C'est notamment le cas des ordonnanceurs hiérarchiques. Une architecture utilise un ordonnancement hiérarchique lorsque plusieurs ordonnanceurs se coordonnent pour le partage d'un processeur. Les ordonnanceurs hiérarchiques ont été initialement proposés dans le contexte des systèmes temps partagé afin de permettre à l'utilisateur de définir des politiques d'ordonnancement adaptées aux besoins de ses applications [16,31]. L'ordonnancement hiérarchique offre une solution séduisante pour l'assemblage d'applications ayant des besoins différents en ressources. Plusieurs normes et standards offrent des services d'ordonnancement hiérarchique. Les modèles d'ordonnancement proposés par ARINC 653, POSIX 1003 ou Ada 2005 en sont des exemples [2,5,15,25,26,27].

Le modèle d'automate utilisé par Cheddar est un modèle simplifié mais proche de celui utilisé dans la plate-forme UPPAAL [1,3,8]. UPPAAL est un ensemble d'outils de *model-checking* et de simulation pour des automates temporisés étendus par des variables. Le modèle d'automate d'UPPAAL permet de définir des horloges, des gardes et des opérations de synchronisation entre les automates [3].

Ainsi, dans Cheddar, un automate temporisé est une machine à états finis étendue par des variables permettant de modéliser le temps et les contraintes de synchronisation entre les différentes entités de l'architecture modélisée. Un système est alors modélisé par un réseau d'automates temporisés. Un sous-programme Cheddar peut être exécuté lors du franchissement d'une transition afin de modifier les données de simulation associées à l'algorithme d'ordonnancement ainsi modélisé (exemple : élection d'une tâche ou calcul d'une priorité).

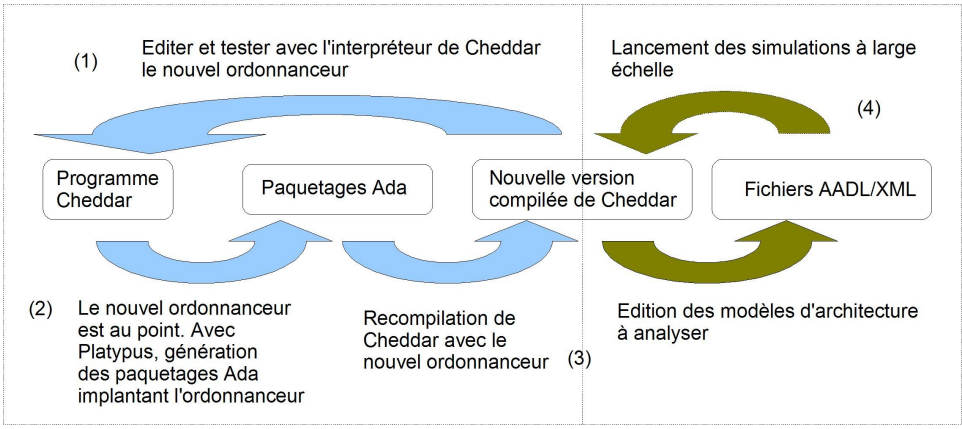

Figure 2 : Processus de simulation d'un ordonnanceur avec Cheddar

2.2. Processus d'utilisation d'un programme Cheddar : de la modélisation à la simulation

Si l'architecture à analyser comporte des ordonnanceurs non encore implantés dans le canevas Cheddar, l'utilisateur peut utiliser le processus décrit par la figure 2 pour la mise en œuvre de ses simulations :

1- L'utilisateur doit d'abord décrire avec le langage de Cheddar les ordonnanceurs à implanter. Il lui est alors possible de mettre au point ses modèles d'ordonnanceur grâce à un interpréteur. L'interpréteur lui permet de tester rapidement le comportement de son programme par des simulations.

2- Une fois son ordonnanceur au point, grâce à Platypus, il peut transformer son programme Cheddar en un ensemble de paquetages Ada pouvant être automatiquement intégré au canevas. Dans la phase de transformation du modèle vers le code Ada, il est possible de spécifier des contraintes afin d'adapter le code généré aux futures simulations (ex : empreinte mémoire et temps de réponse des simulations selon les caractéristiques de l'architecture à analyser).

3- Les paquetages Ada générés peuvent enfin être compilés et intégrés dans Cheddar afin de produire une nouvelle version du canevas.

4- Par la suite, l'utilisateur peut exécuter des simulations de grande taille comme si son ordonnanceur avait été implanté manuellement dans Cheddar.

2.3. Exemple d'un modèle : ordonnancement hiérarchique du standard ARINC 653

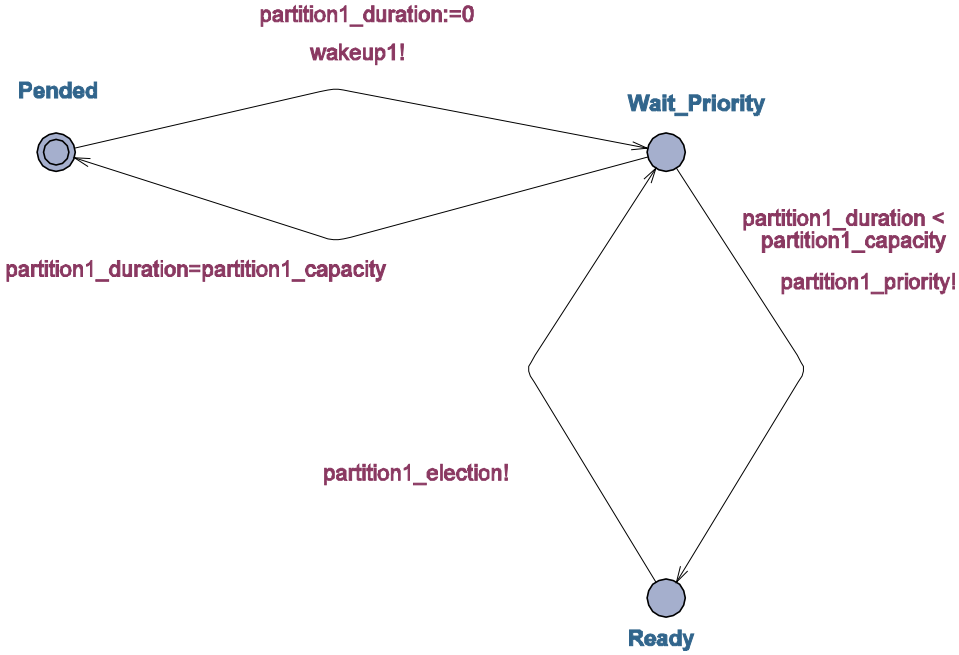

Figure 3 : Automate modélisant l'ordonnanceur de la partition 1

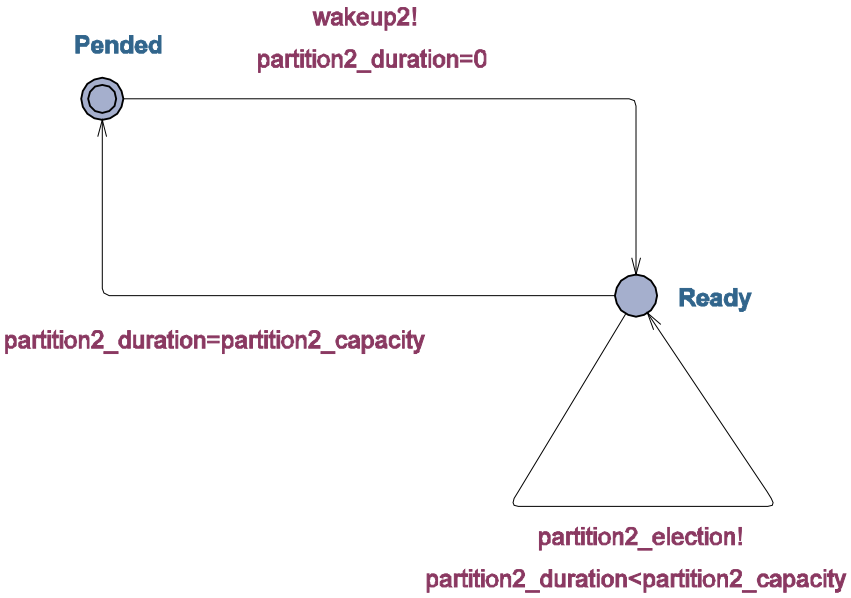

Figure 4 : Automate modélisant l'ordonnanceur de la partition 2

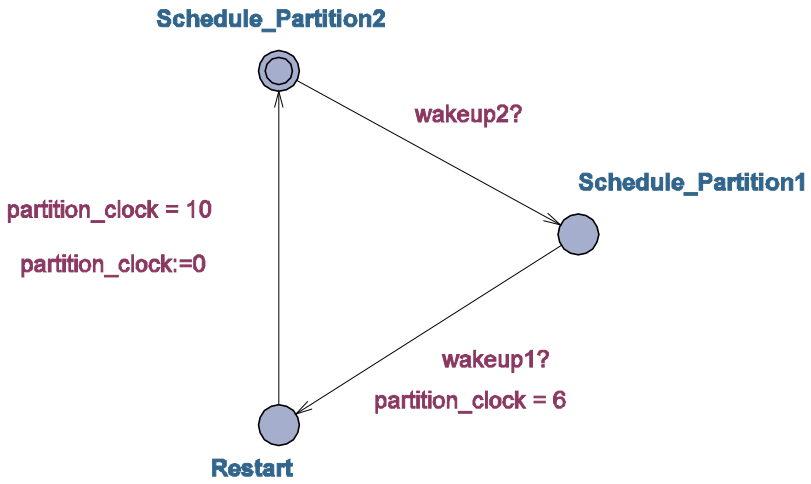

Figure 5 : Automate modélisant l'ordonnanceur de partitions

À titre d'illustration, ce paragraphe décrit un exemple de programme Cheddar qui modélise l'ordonnancement hiérarchique d'ARINC 653 [2]. Une architecture ARINC 653 est un ensemble d'applications éventuellement réparties sur plusieurs processeurs. Une application est appelée une partition. Dans la norme ARINC 653, la partition est l'abstraction qui permet la mise en œuvre des mécanismes d'isolation mémoire et temporelle. Chaque partition est composée d'un ensemble de tâches. Le partage du processeur est alors effectué grâce à un ordonnancement hiérarchique à deux niveaux :

1- Les partitions sont ordonnnancées cycliquement. Pour chaque cycle, l'ordonnancement de partitions stipule quand et pour combien de temps la partition est active, c'est-à-dire quand elle peut exécuter une de ces tâches. Ce premier niveau d'ordonnancement est statique. L'ordonnancement est calculé lors de la phase de conception du système. C'est donc un ordonnancement hors ligne.

2- Le deuxième niveau d'ordonnancement est celui des tâches. Chaque partition ordonnance ses tâches avec un ordonnanceur préemptif à priorités fixes. Les tâches d'une partition donnée peuvent être exécutées uniquement pendant les instants où leur partition est active. Ce deuxième ordonnancement est donc en ligne.

Le modèle d'ordonnancement d'ARINC 653 peut être représenté par un ensemble de programmes Cheddar. Une représentation graphique partielle de la partie synchronisation de cet ordonnanceur hiérarchique est donnée par les figures 3, 4 et 5. Le lecteur pourra se référer à [30] pour consulter l'intégralité des programmes Cheddar de cet exemple.

L'automate de la figure 5 modélise quand les partitions sont actives ou non : c'est le premier niveau d'ordonnancement ARINC 653. Dans cet exemple, nous avons deux partitions exécutées sur le même processeur. Les automates des figures 3 et 4 modélisent l'ordonnancement des tâches de ces partitions : nous modélisons ici le deuxième niveau d'ordonnancement ARINC 653.

Les automates modélisant les ordonnanceurs de tâches des partitions 1 et 2 sont constitués des états suivants :

1- L'état *Pended* indique que la partition est inactive : elle ne peut pas utiliser le processeur afin d'exécuter une de ses tâches.

2- Les états *Wait_Priority* et *Ready* sont les états désignant une partition comme active. Une partition dans l'un de ces états peut exécuter une de ses tâches conformément à son ordonnanceur. L'état *Wait_Priority* est un état intermédiaire pendant lequel l'ordonnanceur de tâches calcule une priorité pour chaque tâche. Ainsi, le sous-programme *partition1_priority* exécuté lors du franchissement de la transition sortante de *Wait_Priority* permet de calculer les priorités des tâches de la partition 1. L'état *Ready* permet de sélectionner la tâche à exécuter pendant la prochaine unité de temps. Conformément au programme de la figure 3, le simulateur invoque un autre sous-programme modélisé par la synchronisation *partition1_election*. Ce sous-programme sélectionne la prochaine tâche à exécuter sur la partition 1 lorsque la transition sortante de l'état *Ready* est franchie.

L'automate de la figure 5 modélise l'ordonnancement cyclique des partitions : le cycle d'ordonnancement des partitions est constitué de 4 unités de temps consacrées à la partition 1 suivies de 6 unités de temps pour la partition 2. La partition 2 ordonnance un jeu de tâches périodiques et critiques selon des priorités fixes alors que la partition 1 exécute un ensemble de tâches non critiques selon un algorithme de type *round-robin*. L'automate de la figure 5 permet d'assurer l'isolation temporelle de ces deux partitions afin que les tâches non-critiques n'interfèrent pas sur l'exécution des tâches critiques.

3. PROCESSUS D'INGÉNIERIE D'UN PROGRAMME CHEDDAR

Le développement d'un ordonnanceur avec le langage de Cheddar permet d'adapter l'outil au contexte d'utilisation. L'évolutivité est assurée tout d'abord par l'interpréteur qui permet l'exécution du nouvel ordonnanceur. Cette évolutivité est complétée par la possibilité de produire automatiquement une nouvelle version de Cheddar à partir d'un modèle d'ordonnanceur. Ce nouvel ordonnanceur est compilé.
Dans ce chapitre, nous décrivons comment, à partir d'un modèle d'ordonnanceur, le canevas de simulation de Cheddar est enrichi.

3.1 Les composants architecturaux de Cheddar

Comme le montre la figure 6, Cheddar est constitué de six composants principaux. Ces composants se répartissent classiquement en deux couches logicielles :

1- La couche basse, pour la gestion des données. Elle est directement dépendante des types de données manipulées par Cheddar et des contraintes associées en termes de règles structurelles et sémantiques. Cette couche de nature générique est fortement réutilisable et évolutive.

2- La couche haute est liée au domaine sous-jacent de la simulation de systèmes temps réel. Bien que cette couche soit de nature très spécifique, elle est cependant évolutive de par l'utilisation d'un procédé de mise en œuvre dirigée par les modèles qui se concrétise par l'utilisation d'un générateur de code spécifique.

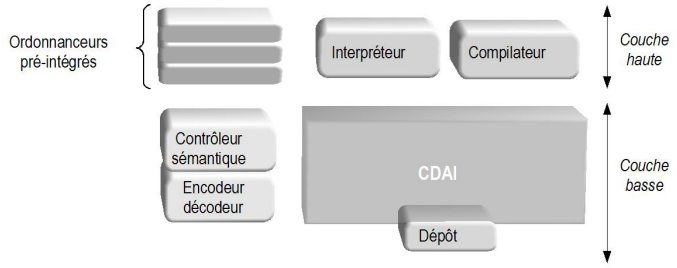

Figure 6 : Architecture logicielle à deux couches de Cheddar

3.1.1. Couche basse

Cheddar comprend un dépôt centralisé pour le stockage des données et des méta-données. Les tâches et les processeurs d'une architecture à analyser sont des exemples de données. Une transition d'un automate ou une instruction de boucle sont elles des méta-données. Le dépôt est un constituant abstrait qui représente un composant physique de stockage de données. Dans la version actuelle de Cheddar, il s'agit de la mémoire.

Le dépôt est encapsulé dans un composant d'accès aux données (CDAI pour Cheddar Data Acces Interface) de sorte que tous les accès en lecture ou écriture de données ou méta-données s'effectuent au travers de ce composant. Le composant d'accès aux données est l'élément architectural central autour duquel s'articulent tous les autres composants de Cheddar.

La couche basse met en œuvre des composants additionnels, spécifiquement liés à la définition des données manipulées au travers de la CDAI :

- Le contrôleur sémantique permet la validation des données et des méta-données.
- Le composant d'encodage et de décodage assure à Cheddar une interopérabilité par échange de données. Ce composant met en œuvre un protocole standard pour l'encodage et le décodage des données et des méta-données vers ou depuis XML.

3.1.2. Couche haute

La couche haute permet la simulation de systèmes temps réel à deux niveaux :

1- Cheddar met en œuvre des algorithmes d'ordonnancement reconnus comme classiques par la communauté. Ces algorithmes d'ordonnancement peuvent être directement utilisés pour un modèle de tâches définit par l'utilisateur. Ces ordonnanceurs sont pré-intégrés sous la forme de paquetages Ada spécialisés (un par ordonnanceur).

2- Des ordonnanceurs spécifiques et non intégrés peuvent être programmés à l'aide du langage de Cheddar (Cf. chapitre 2). Ces programmes sont analysés par le compilateur qui, à partir d'un programme, produit les méta-données correspondantes. Ces méta-données sont des instances de la syntaxe abstraite du langage de Cheddar.

Pour un modèle d'ordonnanceur, l'ensemble des méta-données produites constitue une représentation interne. L'interpréteur exploite cette représentation interne pour le calcul de la simulation proprement dite.

3.2. Procédé dirigé par les modèles

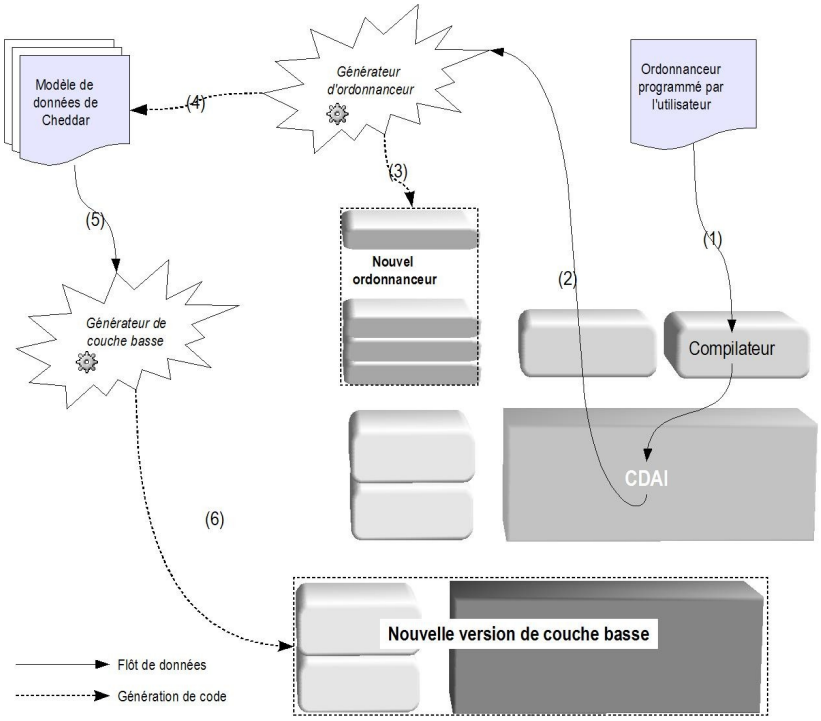

Figure 7 : Évolution incrémentale de Cheddar

Pour la mise en œuvre des deux couches logicielles présentées dans le paragraphe 3.1, un procédé dirigé par les modèles à deux niveaux est exploité. Comme le montre la figure 7, ce procédé assure une évolutivité incrémentale du système en six étapes : un ordonnanceur développé spécifiquement pour une application peut être compilé (1) ; le résultat de la compilation est utilisé en entrée par un premier générateur de code (2) qui d'une part, produit le code du nouvel ordonnanceur (3) et qui, d'autre part, produit une nouvelle version du modèle de données de Cheddar (4). Le générateur de couche basse exploite alors le nouveau modèle (5) pour produire une nouvelle version de la couche basse (6).

3.3 Mise en œuvre de la couche basse

Pour la couche basse, les objets manipulés, comprenant les données et les méta-données, sont spécifiés par le modèle de données de Cheddar. Ce modèle intègre la définition structurelle des hiérarchies et la définition des contraintes des entités du système. À une version particulière de ce modèle correspond une version de la couche basse puisque tous les constituants de cette couche sont automatiquement produits à partir de ce modèle par le générateur de couche basse qui produit les paquetages Ada correspondants.

3.4. Mise en œuvre de la couche haute

Pour un programme donné, le générateur d'ordonnanceurs permet de traduire la représentation interne du programme en un ordonnanceur sous la forme d'un paquetage Ada spécifique intégré au canevas Cheddar. Cette traduction met en jeu deux processus de génération. Tout d'abord, est généré le code de l'ordonnanceur comprenant la mise en œuvre de l'automate et les calculs associés aux états. Ce paquetage est produit dans la couche haute par le générateur d'ordonnanceurs sous la forme d'un ordonnanceur intégré. En complément, le nouvel ordonnanceur doit être déclaré dans le modèle de données de Cheddar, ce qui implique la génération d'une entité complémentaire. Le processus de production automatique de la couche basse est alors exploité. Après recompilation de l'ensemble, le nouvel ordonnanceur fait partie intégrante du canevas Cheddar. L'intégration d'un ordonnanceur spécifique représente un incrément. À chaque incrément, une nouvelle version de Cheddar spécifiquement adaptée au contexte d'utilisation est ainsi produite.

3.5 Modélisation

Classiquement, un procédé dirigé par les modèles met en œuvre un langage de modélisation pour la spécification des modèles et des méta-modèles. Les règles de traduction peuvent être exprimées avec le langage de modélisation choisi ou à l'aide d'autres dialectes ou modèles. Pour la description et la mise en œuvre des modèles et méta-modèles de l'environnement Cheddar, nous utilisons les outils apportés par la technologie STEP ou norme ISO-10303 [10], dont, essentiellement, le langage de modélisation de données EXPRESS [13].

3.5.1. Le standard STEP

Le standard STEP (STEP pour STandard for the Exchange of Product model data) est la norme ISO 10303 élaborée par le sous-comité ISO TC 184/SC4. L'objectif de cette norme est de permettre aux applications informatiques industrielles d'échanger et de partager des données indépendamment des spécificités des différents systèmes informatiques.

Les travaux de normalisation comprennent, d'une part, le développement d'une technologie fournissant des méthodes et outils informatiques neutres pour la description, la validation et la manipulation des informations de définitions de produits et, d'autre part, la spécification de modèles de données standards, appelés protocoles d'application et organisés par métiers industriels. Ces protocoles d'application sont développés dans le cadre strict de la technologie STEP. Ils sont spécifiés dans le langage de modélisation EXPRESS et constituent une bibliothèque de concepts réutilisables pour développer des modèles de données dans différents secteurs industriels [4].

Ces travaux de normalisation sont toujours très actifs. Les développements réçents de la norme visent à permettre l'interopérabilité sémantique avec les outils formels de l'OMG notamment en proposant un méta-modèle d'EXPRESS basé sur le MOF (MOF pour Meta-Object Facility) [19,20] et deux passerelles formelles, l'une vers UML v2 et l'autre vers OWL.

3.5.2 Modélisation avec EXPRESS

Le langage EXPRESS est un langage de modélisation orienté données. Les données sont modélisées au sein de *schémas*. Un schéma comprend la définition de types, d'entités et de contraintes globales. Un schéma peut réutiliser les définitions d'autres schémas. Une entité peut hériter de une ou plusieurs autres entités. Une entité comprend un ensemble d'attributs et de contraintes locales. Un attribut peut être explicite (l'attribut est valorisé explicitement), dérivé (sa valeur est calculée par une expression) ou encore inverse (sa valeur est calculée et indique la cardinalité inverse d'une association entre deux entités).

```
SCHEMA Processors;
        USE FROM Objects;

        ENTITY Generic_Scheduler_Ptr;
        END_ENTITY;

        ENTITY Processor  SUBTYPE OF ( Generic_Object );
           Scheduler : Generic_Scheduler_Ptr;
           Network : STRING;
        DERIVE
                SELF\Generic_Object.Object_Type    :       Objects_Type    :=
Processor_Object_Type;
        END_ENTITY;
END_SCHEMA;
```

Figure 8 : Un extrait du modèle de données de Cheddar : le schéma Processors

La figure 8 montre le schéma *Processors* extrait du modèle de Cheddar. Ce schéma comprend notamment l'entité *Processor* qui spécifie ce qu'est un processeur pour Cheddar. Un processeur est un objet (*Processor* hérite de *Generic_Object*) associé à un ordonnanceur (attribut explicite *Scheduler*) et à un réseau. Le réseau est représenté par son identifiant (attribut *Network*). L'entité *Processor* comprend la redéfinition de l'attribut explicite *Generic_Object.Object_Type*, en attribut dérivé de façon à en fixer la valeur pour une instance de *Processor*.

3.6 Mise en œuvre au sens STEP

La mise en œuvre d'un schéma EXPRESS consiste en son intégration dans un dispositif d'échange de données décrit par la figure 9. Ce dispositif met en œuvre un format d'échange de données standard, ou format pivot [12,14]. Les composants encodeur et décodeur pour le format pivot sont produits automatiquement à partir du schéma EXPRESS qui décrit les données échangées. L'encodeur permet de sérialiser les données échangées vers le format pivot alors que le décodeur effectue le travail inverse de désérialisation. Les instances contenues dans le dépôt sont donc conformes à leur type décrit dans le schéma EXPRESS. Enfin, l'interface standard d'accès aux

données, la SDAI (SDAI pour Standard Data Acces Interface) [11], est utilisée pour construire les composants encodeurs et décodeurs.

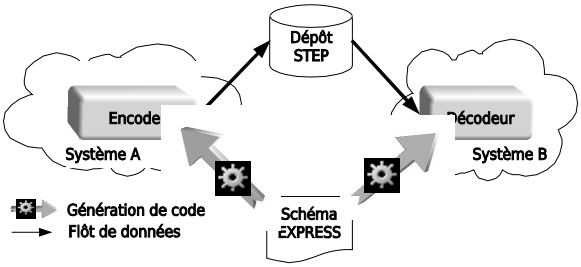

Figure 9 : Processus standard d'échange de données dans STEP

3.7 Génération de code avec STEP

Le processus dirigé par les modèles décrit dans le paragraphe 3.2 exploite deux générateurs complémentaires, un pour générer la couche basse et l'autre pour produire les ordonnanceurs de la couche haute. Classiquement, dans le cadre de l'ingéniérie dirigée par les modèles, la construction d'un générateur de code nécessite la déclaration d'une syntaxe abstraite associée à la déclaration des règles de traduction. La génération de code proprement dite est mise en œuvre comme un processus de compilation.

3.7.1 Déclaration du générateur avec EXPRESS

Pour chaque générateur, un méta-modèle associé à des règles de génération de code est développé en EXPRESS :

- Le méta-modèle est un modèle de données. Il spécifie la syntaxe abstraite du langage source utilisé pour modéliser. Cette déclaration s'abstrait complètement de la syntaxe concrète du langage source.
- Les règles de traduction sont encapsulées dans les entités du méta-modèle sous la forme d'attributs dérivés.

La figure 10 montre une partie de méta-modèle simplifié pour Ada 95 nommé *Ada_For_Cheddar_Meta_Model* utilisé pour Cheddar. Deux entités sont représentées, *class_in_package* et *attribute* :

- *class_in_package* spécifie ce qu'est une classe pour la mise en œuvre du canevas Cheddar. Elle comprend deux attributs explicites, *name* pour le nom de la classe et *attributes*, pour la liste des variables de la classe.
- *attribute* spécifie ce qu'est un attribut avec son nom et son domaine.

La règle de génération de code *ads_code* est associée à *class_in_package*. Elle déclare comment produire le code d'une classe en Ada à partir des données accessibles depuis un *class_in_package*. Le code est en fait calculé par la fonction *class_in_package_ads_code*.

```
SCHEMA Ada_For_Cheddar_Meta_Model;

ENTITY class_in_package;
   name : STRING;
   attributes : LIST of attribute;
DERIVE
   ads_code : STRING := class_in_package_ads_code(SELF);...
END_ENTITY;

ENTITY attribute;
   name : STRING;
    domain : attr_domain;
END_ENTITY;
…
FUNCTION class_in_package_ads_code(cip : class_in_package) : STRING;
LOCAL
  code : STRING;
END_LOCAL;
  code := 'type ' + cip.name + ' is new Generic_Object with '...
  RETURN(code);
END_FUNCTION;

END_SCHEMA;
```

Figure 10 : Extrait d'un méta-modèle simplifié pour Ada/Cheddar

3.7.2. Mise en œuvre du générateur avec STEP

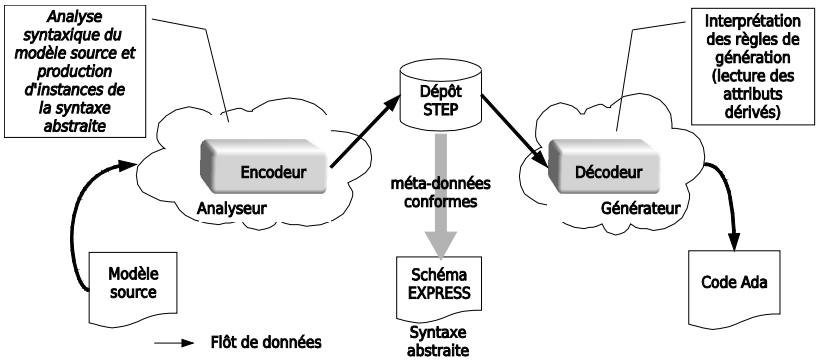

Figure 11 : Génération de code mise en œuvre comme un processus d'échange STEP

Comme le montre la figure 11, la mise en œuvre du générateur de code consiste en une chaîne de compilation intégrée au paradigme d'échange de données STEP [24]. Les données à échanger sont décrites par le modèle EXPRESS de la syntaxe abstraite. Ainsi, on dispose de l'encodeur et du décodeur de méta-données :

- L'analyseur de spécification source utilise l'encodeur pour produire les méta-données.
- Au niveau du décodeur, pour une règle de génération de code spécifiée par un attribut dérivé, le code à produire est obtenu par lecture de cet attribut.

3.8. Génération de code avec Platypus

Platypus [23] est l'environnement STEP que nous utilisons pour construire des générateurs de code suivant la méthode Eugene [24]. Selon cette méthode, le compilateur (typiquement un analyseur Lex/Yacc par exemple) est construit manuellement. Les actions sémantiques de la spécification du compilateur utilisent les points d'entrée de la SDAI pour instancier la syntaxe abstraite. Pour un langage de spécification source, la construction du compilateur peut-être lourde et coûteuse. Cependant, avec Platypus, le développement d'un compilateur pour le langage source n'est pas nécessaire si le langage de modélisation source est EXPRESS et si le méta-modèle est une spécialisation du méta-modèle de Platypus lui-même. Moyennant la spécification du lien de conformité entre les entités du modèle source et celle du méta-modèle (syntaxe abstraite), le processus de génération de code est intégralement automatisé.

4. CONCLUSION ET PERSPECTIVES

Dans cet article, nous proposons un langage adapté à la modélisation d'ordonnanceurs temps réel. En outre, nous proposons un processus dirigé par les modèles permettant, à partir d'un modèle d'ordonnanceur, d'effectuer des simulations d'un système à grande échelle. Ce processus traduit un modèle orienté automate temporisé en un ensemble de paquetages Ada pouvant être intégré au canevas de simulation Cheddar. L'utilisation d'un tel procédé facilite la mise en œuvre de simulateurs temps réel pour l'utilisateur : en effet, ce dernier est capable d'enrichir un canevas de simulation avec de nouveaux algorithmes sans pour autant en comprendre les détails de conception et d'implémentation. Nous pensons que cette méthode est une alternative intéressante à l'utilisation de modèles et d'outils non spécifiquement conçus pour l'analyse de performances de systèmes temps réel (tels que les réseaux de Petri [22] et CPN Tools [32]) ou des outils de simulation spécifiques à une application donnée, qui n'offrent, en général, qu'un faible niveau de réutilisation du logiciel.
Le langage de modélisation d'ordonnanceur proposé par Cheddar soulève un certain nombre de questions concernant l'analyse statique et dynamique des ordonnanceurs ainsi spécifiés. Les automates temporisés autorisent la vérification de propriétés par *model-checking* [3,33]. D'autre part, le sous–ensemble d'Ada utilisé dans le langage spécifique de Cheddar autorise l'application de méthodes d'analyse statique par déduction [36]. Par l'application de ces techniques, nous espérons pouvoir comparer des modèles d'ordonnanceur et améliorer, ainsi, l'analyse de performances de systèmes temps réel dont les ordonnanceurs peuvent être spécifiés avec Cheddar.

Remerciements : Les outils d'analyse AADL de Cheddar utilisent le logiciel Ocarina [34]. Nous remercions donc tout particulièrement l'équipe d'Ocarina pour son aide (B. Zalila, J. Hugues, L. Pautet et F. Kordon). Cheddar est un outil diffusé sous la licence GPL, disponible depuis l'adresse http://beru.univ-brest.fr/~singhoff/cheddar/. Nous tenons à remercier toutes les personnes qui contribuent à ce logiciel. La liste des contributeurs est disponible à l'adresse http://beru.univ-brest.fr/~singhoff/cheddar/#Ref7.

5. BIBLIOGRAPHIE

[1] R. Alur et D. L. Dill : *Automata for modeling real-time systems* ; Proc. of Int. Colloquium on Algorithms, Languages and Programming, vol. 443 of LNCS (1990), p. 322–335, 1990.
[2] Arinc Committee, *Arinc 653 (Avionics Application Software Standard Interface)* : janvier 1997.
[3] G. Behrmann, A. David et K. G. Larsen : *A tutorial on UPPAAL* ; Technical Report mis à jour le 17 novembre 2004, Department of Computer Science, Aalborg University, Danemark, 2004.
[4] M. Bouazza : *La norme STEP* ; Hermès, Documentation multimédia, 1995.
[5] A. Burns, M. Harbour et A. Wellings : *A round-robin scheduling policy for Ada* ; in Reliable Software Technologies, Proceedings of the Ada Europe Conference, 2003.
[6] F. Cottet, J. Delacroix, C. Kaiser et Z. Mammeri : *Scheduling in real-time systems* ; John Wiley and Sons Ltd, 2002.
|7] L. George, N. Rivierre et M. Spuri : *Preemptive and non-preemptive real-time uni-processor scheduling* ; INRIA Technical report n° 2966, 1996.
|8] J. E. Hopcroft et J. D. Ullman : *Introduction of automata theory, languages and computation* ; Addison-Wesley, 2001.
|9] SAE : *Architecture Analysis and Design Language (AADL) AS 5506* ; Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 0.994, août, 2004.
[10] ISO, Part 1 : *Overview and fundamental principles* ; 1994.
[11] ISO, Part 22 : *Implementation method: Standard data access interface specification* ; 1998.
[12] ISO, Part 21 : edition 2, *Implementation method: Clear text encoding of the exchange structur*e ; 2001.
[13] ISO, Part 11 : edition 2, *EXPRESS Language Reference Manual* ; 2004.
[14] ISO, Part 28 : *Implementation method: XML representation of EXPRESS schemas and data* ; 2004.
[15] ISO : Ada *reference manual ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1* (Draft 16).
[16] J. Kay et P. Lauder : *A fair share scheduler* ; Communications of the ACM, vol. 31, p. 44-45, janvier 1988.
[17] M. H. Klein, T. Ralya, B. Pollak, R. Obenza et M. G. Harbour : *A practitioner's handbook for real-time analysis* ; Kluwer Academic Publishers, 1994.
[18] C. L. Liu et J. W. Layland : *Scheduling algorithms for multiprogramming in a hard real-time environnment* ; Journal of the Association for Computing Machinery, vol. 20, n° 1, p. 46- 61, janvier, 1973.
[19] OMG : *OMG's Meta-Object Facility* ; http//www.omg.org/mof/, 2007.
[20] OMG : *OMG Meta-Object Facility (MOF) Specification*, v1.4, 2002.
[21] S. S. Panwalkar et M. Iskander : *A survey of scheduling rules* ; Operations Research, vol. 25, n° 1, p. 45-1, janvier, 1976.
[22] J. L. Peterson : *Petri net theory and the modelling of systems* ; Prentice Hall, 1981.
[23] A. Plantec : *Platypus technical summary and download*, http://cassoulet.univ-brest.fr/mme/, 2007.
[24] A. Plantec et V. Ribaud : *EUGENE: a STEP-based framework to build application generators* ; AWCSET'98, CSIRO-Macquarie University, 1998.
[25] J. A. Pulido, S. Uruena, J. Zamorano, T. Vardanega et J. A. D. la Puente : *Hierarchical scheduling with Ada 2005* ; Proceedings of the 11th International Conference on Reliable Software Technologies, Porto, Portugal, juin 2006.
[26] M. Rivas et M. G. Harbour : *POSIX-compatible application-defined scheduling in MaRTE OS* ; Proceedings of the 14th IEEE Euromicro Conference on Real-Time Systems, Vienne, Autriche, juin 2002.
[27] M. Rivas et M. G. Harbour : *Application defined scheduling in Ada* ; Proceedings of the 12th International workshop on real-time Ada, Viana do Castelo, Portugal, p. 42-51, 2003.
[28] F. Singhoff : *Cheddar Release 2.x User's Guide* ; Technical report, number singhoff-01-2007, disponible à http://beru.univ-brest.fr/~singhoff/cheddar, février 2007.
[29] F. Singhoff, F. Legrand, L. Nana et L. Marcé : *Cheddar: a flexible real-time scheduling framework* ; Proceedings of the International ACM SIGAda Conference, Atlanta, États-Unis, novembre, 2004.
[30] F. Singhoff et A. Plantec : *AADL Modeling and Analysis of Hierarchical Schedulers* ; Proceedings of the International ACM SIGAda Conference, Washington DC, États-Unis, novembre, 2007.
[31] U. Vahalia : *UNIX Internals: the new frontiers* ; Prentice Hall, 1996.
[32] L. Wells : *Performance analysis using CPN Tools* ; Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools 2006, ACM Press, Value-Tools'06, 2006.
[33] B. Bérard, M. Bidoit et F. Laroussinie : *Vérification de logiciels, techniques et outils du model-checking*, Éditions Vuibert, 1999.
[34] J. Hugues, B. Zalila et L. Pautet : *Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina*. In 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07), Porto Allegre, Brésil, juin 2007.
[35] P. Dissaux et F. Singhoff : *Stood and Cheddar: AADL as a pivot language for analysing performances of real-time architectures* ; Conférence internationale ERTS 2008, Toulouse, janvier 2008.

[36] J. Barnes : *High integrity software: The Spark approach to safety and security* ; Addison-Wesley Publishing Company. 2003.
[37] F. Singhoff : *The Cheddar AADL Property sets (Release 2.x)* ; LISyC Technical report number singhoff-03-2007, Février 2007.