Formalisation et structuration des architectures opérationnelles pour les systèmes embarqués temps réel

Jean-Philippe Babau

N° d'ordre 2005 0013 Année 2005

Mémoire d'habilitation à diriger des recherches

présentée devant

l'Institut National des Sciences Appliquées de Lyon
et l'université Claude Bernard de Lyon – LYON I

Formalisation et structuration des architectures opérationnelles pour les systèmes embarqués temps réel

soutenue par

Jean-Philippe Babau

le 12 décembre 2005 devant la commission d'examen composée de

Dorina Petriu, Professeur de l'université de Carleton, DRES, Ottawa	rapporteur
François Terrier, Directeur de recherche CEA Saclay, LIST, Paris	rapporteur
Yvon Trinquet, Professeur à l'IUT de Nantes, IRCCyN, Nantes	rapporteur
Charles André, Professeur à l'UNSA, I3S, Nice	examinateur
Anne Mignotte , Habilitée à diriger des recherches, Lyon	examinateur
Jean-Jacques Schwarz, Professeur à l'UCB Lyon I, LIRIS, Lyon	examinateur
Joseph Sifakis, Directeur de recherche CNRS, Verimag, Grenoble	examinateur
Stéphane Ubéda, Professeur à l'INSA de Lyon, CITI, Lyon	examinateur

Remerciements

Je tiens tout d'abord à exprimer ma gratitude pour Dorina Petriu, François Terrier et Yvon Trinquet qui ont accepté de lire et d'évaluer ce mémoire. Je remercie également les autres membres du jury, Charles André et Joseph Sifakis, qui ont accepté de se pencher sur mon travail. Je remercie aussi bien évidemment Anne Mignotte, Jean-Jacques Schwarz et Stéphane Ubéda, avec qui j'ai eut à collaborer au sein des laboratoires L3i et CITI. Durant ces neuf années passées à l'INSA, leurs conseils critiques et constructifs, leurs encouragements sans cesse répétés, leur soutien sans faille m'ont aidé grandement dans la direction de mes activités de recherche. Enfin, Je souhaite remercier plus particulièrement Dorina Petriu qui m'a fait l'honneur de venir du Canada et Anne Mignotte qui m'a fait le plaisir de revenir à l'INSA dans un contexte universitaire.

Je souhaite également remercier tous les étudiants, Avinash, Pierre-Olivier, Ahmad, Mostefa, Jean-Charles, Belgacem, Julien et Jean-Louis, avec qui j'ai eut à travailler ou avec qui je travaille. Je souhaite, par ces remerciements, leur signifier que leur encadrement a été pour moi d'une aide précieuse, voire indispensable, dans le déroulement des travaux menés.

Mes remerciements vont aussi à l'ensemble des collègues et du personnel administratif des laboratoires L3i et CITI et du département Informatique. Leurs conseils, leurs soutiens, et leurs aides, dans les moments critiques, ont été précieux et fort appréciés. J'ai aussi apprécié la chance d'être dans une atmosphère de travail souvent conviviale et toujours rigoureuse.

Merci à Anne pour ces quelques pas de danse ; merci à Maryvonne pour m'avoir aidé à spécifier tant et tant de contraintes diverses et variées ; merci à Isabelle et Fabrice avec qui je partage mes préoccupations de recherche ainsi que parfois quelques moments de stress et de fatigue.

Enfin, je remercie l'ensemble de mes proches pour m'avoir soutenu, accompagné, supporté et ne pas m'avoir posé trop de questions sur les concepts.

Table des matières

Avant-propos

Ce rapport présente une synthèse des travaux de recherche que j'ai menés au sein des laboratoires L3i (Laboratoire d'ingénierie de l'informatique industrielle) et CITI (Centre d'Innovations en Télécommunications et Intégration de services) de L'INSA de Lyon depuis ma nomination, en 1997, en tant que maître de conférences.

Dans ce rapport, nous nous focalisons sur les travaux les plus récents de cette période qui concernent la formalisation et la structuration des architectures opérationnelles pour les systèmes embarqués temps réel au comportement dynamique.

Organisation du document

Nous introduisons d'abord les systèmes embarqués temps réel (SETR) soumis à des contraintes fortes de qualité de service (QdS) et les problématiques de recherche liées à leur processus de développement. Puis, nous motivons nos objectifs de recherche sur la formalisation et la structuration des architectures opérationnelles, ces dernières étant au cœur de la problématique de correction des SETR vis-à-vis des contraintes de QdS.

Pour la formalisation, nous proposons d'explorer l'utilisation des langages formels SDL « *Specification and Description Language* » [SDL 96] et IF « *Intermediate Format* » [BFG 99] et les techniques formelles associées. Pour la structuration, nous présentons une solution basée sur le paradigme composant [Szy 98].

En perspective, nous proposons d'étendre les travaux à la problématique du déploiement, soit l'étape de construction de l'architecture opérationnelle. Enfin, afin de couvrir l'ensemble du cycle de développement, nous proposons un projet de recherche guidé par l'application. Ce projet pluridisciplinaire concerne les réseaux de capteurs.

I Les systèmes embarqués temps réel

Dans cette première partie, nous définissons le domaine d'étude des systèmes embarqués temps réel. Les systèmes sont vus au travers de leurs contraintes de qualité de service qui impactent fortement leur processus de développement.

Du fait des avancées technologiques qui permettent une plus grande miniaturisation des systèmes, les systèmes embarqués temps réel (SETR), souvent cachés aux utilisateurs, sont de plus en plus présents dans notre environnement et de plus en plus complexes.

Les applications les plus connues des SETR sont les systèmes de transport (voiture, avion, train) et les systèmes mobiles autonomes (robot, fusée, satellite). De même, les systèmes liés à la gestion d'un périphérique (imprimante, souris sans fil), à la mesure (acquisition en temps réel) et les systèmes domotiques (électroménager) sont des systèmes embarqués pouvant posséder des contraintes temps réel liées aux capteurs ou actionneurs utilisés. Enfin, la notion d'embarqué peut été étendue aux objets portables grand public (cartes à puce, assistants personnel, téléphones mobiles, lecteurs vidéo, consoles de jeu).

Le point commun de tous ces systèmes porte sur la spécificité de leurs contraintes. Il s'agit aussi bien de contraintes sur le support d'exécution que de contraintes de Qualité de Service (QdS) [ISO 95][ITU 94], soient :

- des contraintes ergonomiques se traduisant par une taille réduite des systèmes pour assurer leur portabilité ou leur intégration dans un système physique ;
- des contraintes physiques du domaine d'application (contraintes électriques, mécaniques, thermiques) ;
- des contraintes dues à leur enfouissement (absence d'un interface homme-machine (IHM), pas de connexion à un réseau fixe) ;
- des contraintes de coût pour des produits bon marchés;
- des contraintes d'énergie liées à une alimentation autonome ;
- des contraintes dues à la faible quantité de mémoire disponible ;
- des contraintes temps réel liées aux exigences du procédé contrôlé ou aux utilisateurs vis-à-vis des applications multimédia supportées.

Vu la diversité des domaines d'application et des contraintes à considérer, les problématiques de recherche autour des SETR recouvrent aussi bien des aspects matériels que logiciels pour construire des systèmes vis-à-vis d'une certaine classe de contraintes. Nos études se limitent aux aspects logiciels pour la prise en compte de contraintes de QdS liées à l'utilisation de ressources physiques limitées.

Il faut ajouter à ces contraintes de QdS une exigence forte de sûreté de fonctionnement. En effet, lorsqu'un SETR est dédié au pilotage d'un procédé critique (centrale nucléaire, véhicule), une défaillance n'est pas tolérée. Pour les systèmes non critiques (domotique, objets portables), une défaillance n'a certes pas de conséquence grave, mais du fait de leur caractère enfoui, des pannes fréquentes rendent, en pratique, le système inutilisable. C'est alors la contrainte économique qui impose la fiabilité des systèmes.

L'ensemble des contraintes énoncées précédemment conduit les systèmes à disposer de ressources, tant au niveau matériel que logiciel, aux capacités limitées et au comportement temporel prédictible [LW 04]. De son côté, le développement met particulièrement l'accent sur l'implémentation, soit sur les politiques de gestion des ressources physiques et l'organisation du code, afin de prédire les performances du système pour évaluer sa correction.

Du fait de leur aspect réactif, un point essentiel à considérer lors de l'implémentation est la prise en compte des événements externes. On distingue alors les approche synchrones, où les événements sont traités immédiatement dès leur arrivée en temps « nul » (la fin d'un traitement précède toujours l'activation du prochain traitement), des approches asynchrones, où le traitement d'un événement externe est associé, à l'exécution, à une entité concurrente autonome. Dans l'approche synchrone, le parallélisme des événements est considéré dans un modèle de haut niveau (Esterel [BS 91] [Ber 92], Signal [LGL 91] ou LUSTRE [HCR 91]), puis un code séquentiel est obtenu par synthèse de ce modèle de haut niveau. Dans l'approche asynchrone, les entités concurrentes ou tâches sont gérées à l'exécution via les services offerts par un exécutif temps réel ou RTOS « *Real Time Operating System* » [SR 04][Tri 05]

L'avantage principal d'une implémentation synchrone est de produire un code dont le comportement est connu a priori. Ce code peut être alors certifié. Ces approches sont adaptées aux systèmes critiques dont les caractéristiques de QdS sont statiques. A l'inverse, le fait de s'appuyer sur un exécutif temps réel permet à l'exécution :

- la gestion dynamique des modes de fonctionnement (arrêt/relance d'une activité) ;
- la préemption dynamique des actions moins prioritaires ;
- la gestion de tâches à contraintes temps réel souple ;
- la gestion d'événements apériodiques[1] ou sporadiques
- la gestion dynamique des fautes ;
- la gestion de l'asynchronisme dans la communication avec l'environnement externe.

Dans nos travaux, on s'intéresse à des applications possédant divers modes de fonctionnement, des lois d'arrivée complexe d'événements en provenance du procédé contrôlé et à des contraintes hétérogènes de QdS (contraintes temps réel dur et souple, contraintes de temps et d'énergie).

Dans ce contexte, l'approche asynchrone est plus adaptée car elle permet, au travers des tâches, une gestion autonome à l'exécution des activités concurrentes du système. Le RTOS offre ainsi les services nécessaires pour s'adapter aux divers changements du contexte d'exécution. Une partie essentielle du RTOS est alors l'ordonnanceur qui réalise à chaque instant l'élection de la tâche à exécuter sur le processeur. La plupart des exécutifs fournissent un ordonnanceur préemptif à priorité. Lors de la phase de vérification des propriétés temps réel, on doit s'assurer que les tâches se réalisent selon une échéance donnée. Pour cela, on s'appuie sur l'utilisation d'algorithmes optimaux pour l'ordonnancement des tâches [Sha 04].

Dans les projets industriels, afin de maîtriser les performances des systèmes, les SETR sont construits de façon dédiée aussi bien au niveau matériel (ASIC[2], DSP[3], microcontrôleurs, SoC[4], réseaux de terrain, organisation de la mémoire), que logiciel (systèmes d'exploitation, protocoles et organisation du code). Les méthodes de

[1] événement apériodique : la date d'arrivée de l'événement est aléatoire ; événement sporadique : la date d'arrivée de l'événement est aléatoire mais l'intervalle de temps entre deux occurrences de l'événement est borné ;

[2] ASIC : *Application-Specific Integrated Circuit,* composant électronique dédié à une application

[3] DSP : *Digital Signal Processor,* processeur dédié au traitement du signal (vidée, son, images, …)

[4] SoC : *System-on-Chip*, puce électronique comprenant un système informatique complet (processeurs, mémoires, OS, périphériques)

développement des systèmes sont des méthodes spécifiques focalisées sur l'implémentation. Ceci réduit considérablement la réutilisation des technologies développées, la maîtrise des choix de conception (peu ou pas d'abstraction) et la qualité des systèmes produits (peu d'interopérabilité des systèmes, peu d'évolutions, peu de réutilisation). En pratique, cette approche aboutit à des systèmes monolithiques et non flexibles.

Le modèle économique actuel du processus de développement des SETR fait intervenir différents acteurs tout au long du cycle de vie du système pour des projets de plus en plus importants (par exemple jusqu'à 70 ECU[5] pour 100 Mo de code dans une automobile) et aux besoins de plus en plus variés. Les industriels souhaitent donc désormais pouvoir disposer d'outils et de techniques pour développer de manière sure et rapide les logiciels embarqués, soient :

- fournir un cadre formalisé d'échange d'information (modèles à haut niveau d'abstraction) afin d'améliorer la communication entre le client, le fournisseur et les équipes techniques ;
- maîtriser au plus tôt dans le cycle de développement les choix de conception et mesurer leur impact en terme de QdS ;
- permettre l'évolutivité des logiciels embarqués pour intégrer au cours du temps les nouvelles technologies et de nouvelles versions ;
- réduire les temps de développement en améliorant la réutilisation de code, de schémas de programme ou de composants logiciels.

La prise en compte de la complexité des systèmes, le besoin de formalisation du processus de développement, les contraintes des SETR (ressources limitées et spécifiques, prédictibilité, correction) font qu'il est nécessaire de pouvoir disposer d'un support permettant une maîtrise fine du processus de développement et une gestion sûre des ressources utilisées par le système. L'utilisation des principes de génie logiciel doit permettre de répondre au défi de la maîtrise et de l'amélioration des processus de développement. L'intégration des techniques formelles doit permettre d'assurer la correction des applications produites. Ces problématiques définissent mon domaine de recherche depuis 1997.

[5] ECU Engine Control Unit : unité de traitement, Mo : Mega octet

II Processus de développement des SETR

Dans l'objectif d'introduire et de positionner les contributions qui suivent, nous proposons une synthèse permettant de classifier les travaux de recherche qui concernent le processus de développement des systèmes embarqués temps réel.

II.1 Développement par étapes

Le développement par étapes est adapté aux SETR complexes. En effet, il permet de décomposer un problème en sous-problèmes, chaque étape se focalisant sur un aspect du problème, soient : l'expression des besoins et des contraintes de QdS, la définition du support d'exécution, la décomposition architecturale du système, le choix d'un modèle de concurrence et d'une politique d'ordonnancement. De plus dans le cadre de projets importants, l'identification et la formalisation de chaque étape apparaissent comme bien adaptées à la contractualisation nécessaire entre les divers acteurs du développement. Par exemple, dans le domaine de l'automobile, une séparation claire est faite entre les équipes qui spécifient (le constructeur), celles qui conçoivent et implémentent (les équipementiers) et enfin celles qui intègrent et valident (le constructeur). Ces équipes, aux cultures scientifiques souvent différentes, doivent pouvoir communiquer par l'échange de modèles communs. Ces modèles concernent typiquement les étapes de spécification, de conception, d'implémentation et de validation. Dans le domaine des SETR, les contraintes de QdS doivent être exprimées et prises en compte à chacune des étapes.

La spécification permet de formaliser les besoins du système via un ensemble de propriétés représentant les fonctionnalités, le comportement et les performances attendues (cf. figure 1). Du fait de l'aspect réactif des SETR, il est aussi nécessaire de modéliser le comportement de l'environnement [Per 90]. En effet, le comportement temporel du système, et donc sa correction, est directement lié au comportement temporel de l'environnement.

Une fois les contraintes exprimées, le résultat de la conception décrit une solution au problème posé. Dans ce cadre, la conception architecturale a émergé dans années 70 [RK 76] pour aider à la maîtrise de la complexité des systèmes. Une architecture logicielle peut être définie comme un ensemble de contraintes de programmation définissant les éléments logiciels d'un système, ainsi que les relations entre ces éléments [BCK 03]. L'architecture fournit des informations sur l'organisation et la configuration du système, en masquant les détails d'implémentation, considérés comme non pertinents à ce niveau d'analyse. L'architecture est un élément structurant pour la gestion de projet. Elle offre un support pour l'évolutivité et de la réutilisation des systèmes logiciels. Et surtout elle sert de base pour la validation et le codage. En fonction du stade de développement du système, la description de l'architecture peut être une spécification abstraite du système qu'on cherche à concevoir ou bien une description du système opérationnel [DF 05].

Comme présenté dans le projet EAST-EEA[6], on distingue généralement l'architecture technique ou matérielle qui correspond aux machines (processeurs, ASIC, …), aux mediums de communication (réseaux, bus) et aux services associés (exécutif, services de communication) de l'architecture logique ou applicative. Cette dernière décrit une organisation de l'application en modules ou composants et les liens ou connecteurs entre ces composants. Le déploiement correspond au placement des composants sur les

[6] EAST-EEA : Electronic Architecture and Software Technology – Embedded Electronic Architecture, projet ITEA, consortium européen de constructeurs automobiles, équipementier, universitaires www.east-eea.net

entités d'exécution (machines, tâches) et à la mise en place des services nécessaires à leur communication. Il aboutit à une architecture dite opérationnelle. Au final, l'architecture opérationnelle définit l'ensemble des éléments nécessaires à l'implémentation du système, sous forme de programmes concurrents et communicants partageant des ressources communes [Sta 88].

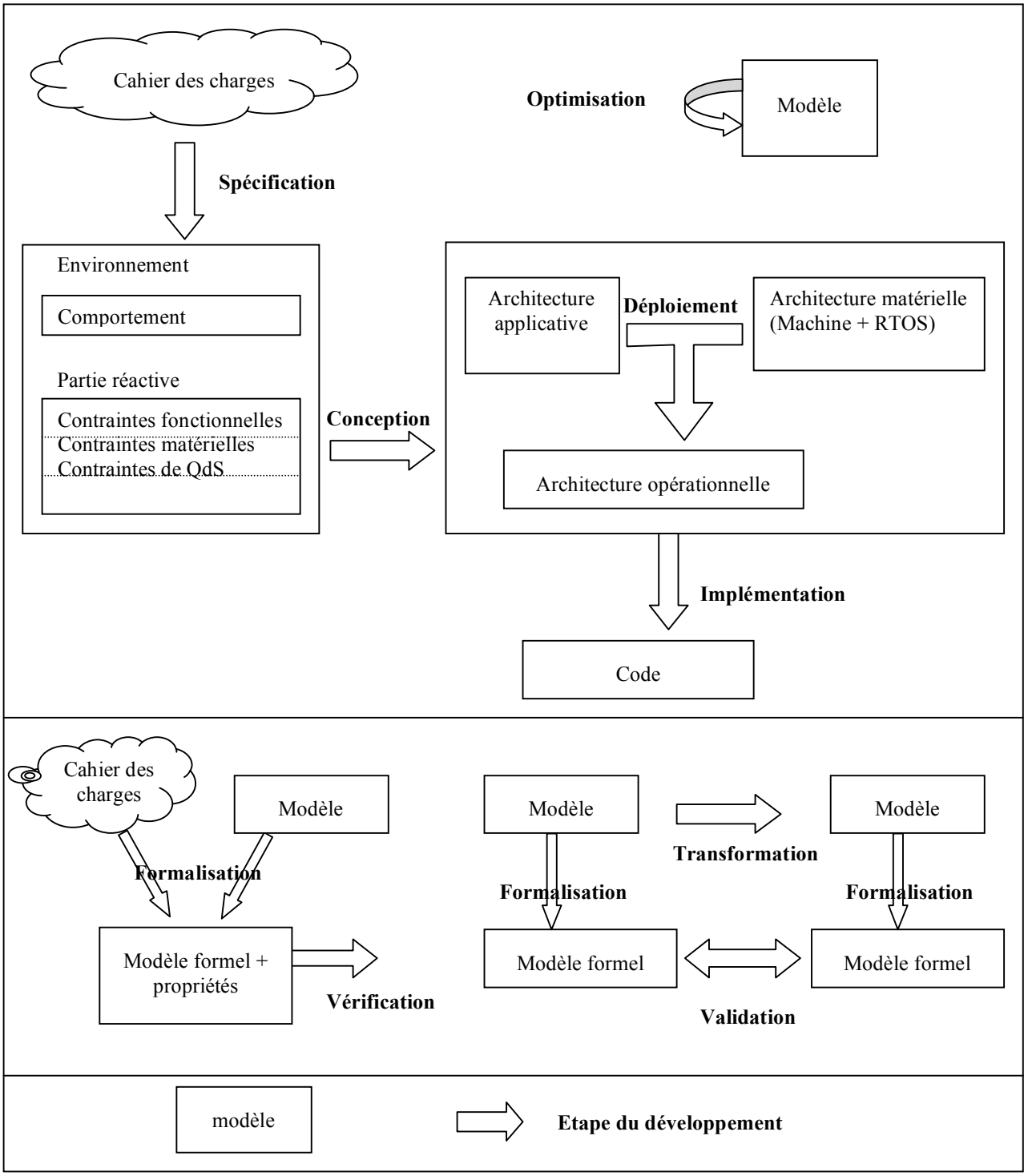

Figure 1 : étapes du développement pour les systèmes embarqués temps réel

Une fois l'architecture opérationnelle définie, les dates d'activation, voire d'exécution, des tâches peuvent être pré-calculées « hors-ligne » [XP 90][GC 02]. On aboutit alors à une séquence qui est ensuite chargée dans une table utilisée par un séquenceur [KDK 89] [OSK] [SZ 92]. A l'inverse, dans une approche, dite « en-ligne », les tâches sont gérées à l'exécution et l'ordonnanceur du RTOS implémente la politique d'ordonnancement. Par la suite, nous considèrerons cette approche afin de conserver un modèle asynchrone à l'exécution.

La construction par étape est donc liée à un ensemble d'opérations dites de transformation dont le raffinement (spécification, conception, déploiement ou implémentation) qui permet d'affiner un modèle et l'optimisation qui permet son amélioration.

L'optimisation correspond à la construction d'un nouveau modèle, équivalent au premier en termes de comportement et de fonctionnalité, mais optimisé par rapport à une propriété donnée. Le *refactoring* porte sur l'amélioration de la qualité des modèles comme la réutilisabilité, la complexité, l'adaptabilité, la compréhension, la lisibilité [Fow 99] [SPT 01]. Pour les SETR, l'optimisation porte sur les propriétés de QdS du code telles que l'espace mémoire et le temps de réponse.

Enfin, au vu de la criticité des applications et des contraintes fortes de QdS, on doit s'assurer de leur correction. L'analyse s'appuie alors sur des modèles formels. On peut ainsi valider une transformation, c'est à dire s'assurer de la conservation de propriétés lors de cette transformation, ou vérifier des propriétés sur un modèle. Typiquement, la vérification de propriétés temps réel sur une implémentation multitâche porte sur le respect d'échéances par les tâches, respect qui dépend de la politique d'ordonnancement des tâches.

Les choix architecturaux étant nombreux et les techniques formelles étant variées, le processus de développement des SETR peut s'avérer rapidement complexe. Pour aider à la maîtrise de ce processus, les travaux de recherche s'intéressent alors à fournir des techniques et outils et pour supporter chaque étape ou transformation, soient :

- fournir des modèles et des paradigmes pour aider à la description des architectures et des contraintes de QdS ;
- développer des techniques et outils pour assurer la correction des systèmes vis-à-vis des propriétés de QdS ;
- automatiser les opérations de transformation afin d'améliorer le processus de développement;
- mettre en place des méthodes de développement afin d'accroître la qualité du processus de développement.

Nous revenons maintenant sur chacun de ces axes de recherche, et plus particulièrement les deux premiers qui sont au cœur de nos préoccupations de recherche, en commençant par la vérification et validation.

II.2 Vérification et validation

Au vu des exigences de sûreté de fonctionnement, de nombreuses études se focalisent sur les techniques formelles. Pour ce qui est de la vérification de propriétés temps réel, il existe des techniques spécifiques d'analyse du pire temps de réponse [Kle 93][Ber 03][TC 94]. De façon plus générale, les techniques formelles s'appuient sur une modélisation exhaustive des comportements du système.

II.2.1 Modélisation formelle par simulation exhaustive

La première étape de la mise en place de techniques formelles par simulation exhaustive est la construction d'un modèle du système, généralement basé sur la sémantique des diagrammes de transition étiquetés ou LTS « *Labelled Transition System* » [ARN 92]. Un LTS est un graphe orienté où un nœud représente un état du système et une transition représente une action de changement d'état, modélisée par un label.

Parmi les techniques de modélisation existantes, nous considérons celles qui permettent de modéliser l'écoulement du temps physique, les systèmes concurrents et communicants et enfin qui possèdent une sémantique de communication asynchrone tels que IF [BFG 99], UPPAAL [LP 97] et Kronos [DOT 96]. Ces formalismes se basent sur les principes définis par les automates temporisés communicants proposés par [AD 94]. Ils permettent de représenter plus simplement le système que les LTS en s'appuyant sur une représentation modulaire à base d'automates communicants. L'ensemble des comportements du système est ensuite obtenu par composition asynchrone des automates, c'est à dire avec entrelacement des transitions. Les différents langages se distinguent par la sémantique de synchronisation des automates (synchrone par rendez-vous pour Kronos et UPPAAL ou asynchrone par échange de signaux pour IF) et la sémantique d'écoulement du temps (temps discret pour IF et temps continu pour UPPAAL et Kronos). L'écoulement du temps est contraint via l'utilisation d'horloges basées sur un temps global, de gardes sur les transitions (IF, UPPAAL et Kronos) et d'invariants sur les états (UPPAAL, Kronos). L'expression des contraintes, respectivement des actions, sur les horloges reste limitée à une comparaison avec une valeur entière positive, respectivement une initialisation ou un arrêt. Pour d'autres gardes ou actions, les modèles peuvent en effet devenir rapidement indécidables [BD 00] [Bou 05]. Mais même lorsque les systèmes sont décidables, la modélisation exhaustive se heurte au phénomène d'explosion combinatoire du nombre d'états et de transitions. Ce phénomène se produit lors de la construction du modèle ou lors de sa vérification et est aggravé par la prise en compte des horloges et des données. Pour prévenir cet effet, il est donc nécessaire de mettre en place des stratégies de réduction des modèles ou de leur représentation :

- utilisation optimisée des variables lors de la modélisation [God 04] ou lors de la génération du comportement global du système [CR 04][Muc 97] ;
- représentation symbolique (régions d'horloge [AD 90]) ou compacte (BDD « *Binary Decision Diagram* » [Bry 92]) ;
- abstraction de certains comportements par construction d'un modèle réduit, équivalent au sens de la propriété recherchée, au modèle non réduit ;
- réduction d'ordre partiel [GW 94] [VAM 96] pour réduire les entrelacements ;
- une dernière technique est d'éviter, lors de la vérification, la construction (« *vérification à la volée* » [Hol 96]) et la mémorisation de tous les états possibles du système [McM 93].

Le phénomène d'explosion combinatoire est, en pratique, rapidement atteint et sa gestion est un point essentiel de l'application des techniques formelles par modélisation exhaustive à des cas réalistes. La mise en place des techniques formelles requiert alors de jouer sur la modélisation en tenant compte des propriétés du système (optimisation des modèles) et en s'appuyant sur les propriétés recherchées (abstraction de certains comportements).

Notons au passage que les techniques d'équivalence [Par 81] [Man 74][Mil 81], utiles à la réduction des modèles, sont aussi utilisées pour la validation des modèles afin d'assurer la conformité des transformations. Ces techniques se concentrent sur le comportement des systèmes.

Une fois le système formellement modélisé, on doit formaliser les propriétés à vérifier, classiquement des propriétés de sûreté (« quelque chose de mauvais n'arrivera jamais ») ou de vivacité (« quelque chose de bon arrivera »). On fait alors la distinction entre les propriétés comportementales liées au temps logique, des propriétés temps réel liées, elles, au temps physique. Pour la formalisation des propriétés comportementales, on fait appel aux logiques temporelles, dont les plus connues sont LTL (« *Linear Temporal Logic* ») [Pnu 81],CTL *(« Computation Tree Logic »*) [CE 81], CTL* (regroupement de LTL et CTL). Pour les propriétés temps réel, il est nécessaire d'introduire des extensions aux logiques temporelles MTL (*« Metric temporal Logic »*) [Koy 90], TCTL (« *Timed CTL »*) [ACD 93] ou d'utiliser une logique spécifique comme la logique temps réel [JM 86]. Enfin, de plus en plus d'études proposent de « programmer » les propriétés par la mise en place d'observateurs. Un observateur est un automate qui scrute l'état du système pour évaluer sa correction [BGO 04][RMJ 04]. Une fois le modèle et les propriétés formalisés, la vérification fait appel à des techniques de *model-checking* : littéralement « vérification du modèle » [Mer 01].

Il existe de nombreux outils implémentant les techniques de vérification et de validation. Ils sont directement intégrés dans les environnements de modélisation (UPPAAL, KRONOS), ou alors multi-plates-formes comme CADP « *Construction and Analysis of Distributed Processes* » [GLM 02][7]. Il nous semble que cette dernière approche est préférable car elle offre des possibilités de manipulation du LTS pour, par exemple, sa réduction avant vérification.

En pratique, la mise en place des techniques formelles est souvent guidée par les possibilités des outils et contrainte par le phénomène d'explosion combinatoire. Au delà des formalismes et des outils, l'utilisation des techniques formelles est confrontée à la problématique de leur mise en œuvre. Il faut alors proposer des méthodologies dont, de notre point de vue, les étapes sont :

- faire le choix d'un formalisme adapté pour la modélisation du système ;
- définir une stratégie pour s'assurer que le modèle formel représente bien le système à modéliser ;
- mettre en place une stratégie de réduction des modèles orientés par les propriétés à vérifier ;
- faire le choix d'une technique de modélisation des propriétés à vérifier.

[7] CADP, conjointement développé par l'action VASY de l'INRIA Rhone-Alpes et le laboratoire Verimag, propose de nombreux outils pour la vérification formelle www.inrialpes.fr/vasy/cadp/

La thèse menée au sein du CITI par Karen Godary [God 04], thèse à laquelle j'ai eut à collaborer sur le plan des techniques utilisées, me semble une bonne illustration de la mise en place d'une telle méthodologie. La thèse a permis l'évaluation formelle des pires temps de réponse des services du protocole TTP/C « *Time Triggered Protocol* » [TTT 03] en présence de fautes. L'étude propose les étapes suivantes :

- comparaison, vis-à-vis de l'explosion combinatoire, de modèles formels pour les services TTP/C : réseaux de Petri temporels [Mer 74], SDL / IF et UPPAAL ;
- mise en place de modèles formels à base d'automates temporisés UPPAAL ;
- vérification des modèles par la vérification de propriétés extraites de la spécification TTP/C ;
- mise en place de techniques de réduction (basées sur la sémantique de l'application et sur une modélisation au plus juste) pour limiter le phénomène d'explosion combinatoire
- instrumentation du *model-checking* pour la mesure de performance.

Enfin, la thèse s'inspire des résultats obtenus pour proposer des modèles paramétrés, selon le contexte applicatif, des caractéristiques temporelles des services étudiés. Ces modèles permettent alors d'évaluer rapidement le pire temps de réponse des services. On se rapproche alors des approches analytiques où l'on ne cherche pas à caractériser l'ensemble des comportements temporels du système mais seulement le pire cas. En particulier, la vérification de propriétés temps réel (respect des échéances) d'un système multitâche s'appuie sur des techniques analytiques pour l'évaluation du pire temps de réponse des tâches.

II.2.2Ordonnancement

La vérification des échéances est liée aux algorithmes d'ordonnancement temps réel. Dans le cadre d'un système monoprocesseur des algorithmes d'ordonnancement sont connus :

- par une affectation de priorités statiques établies selon la plus petite période RM ("*Rate Monotonic*") [LL73] ;
- par une affectation de priorités statiques établies selon la plus petite échéance DM ("*Deadline Monotonic*") [LW 82] ;
- par une affectation de priorités dynamiques établies selon la plus petite échéance dynamique ED ("*Earliest Deadline*") [LL 73].

Ces algorithmes sont optimaux pour des tâches préemptibles indépendantes et périodiques, modèle de tâches relativement simple. Ces techniques ont donc été étendues pour intégrer d'autres caractéristiques des tâches :

- tâches apériodiques et sporadiques [LSS 87][SLS 88][SSL 89] [SB 96] ;
- tâches possédant des contraintes de précédence [CSB 90][HKL 91][GL 95] ;
- tâches partageant des ressources en exclusion mutuelle [Ka 82][SRL 90][CL 90][Ba 91] et ayant aussi des contraintes de précédence [SS 94] ;
- tâches à durée variable [Gar 99] ou à période élastique [BA 02].

et d'autres contraintes de QdS :

- contraintes de gigue[8] [YL 01][BRC 00] ;
- contraintes d'énergie [SRS 03][AMM 04].

[8] gigue : décalage vis-à-vis d'une date régulière spécifiée

De même, afin de considérer les systèmes répartis (multiprocesseurs) ou distribués (plusieurs machines en réseau), il est nécessaire d'étendre les techniques à :

- l'ordonnancement dans un contexte multiprocesseur [DL78][PST 97][GFB 03] ;
- l'allocation de tâches sur les processeurs [GJ 79] ;
- l'ordonnancement des messages [THW 94][BBR 02].

Les résultats précédents ont été établis pour les systèmes temps réel durs. Dans un contexte temps réel souple, l'ensemble des tâches peut éventuellement ne pas être exécuté. Pour traiter les surcharges temporelles, il n'existe pas d'algorithme optimal mais seulement des heuristiques dont les principes peuvent être résumés par :

- une tâche est décomposée en deux parties, une partie obligatoire et une partie optionnelle [SLC 91] ;
- les tâches sont supprimées, en cas de surcharge, selon le niveau d'importance donné par l'utilisateur [Del 94] ;
- selon la charge, une requête est traitée par une tâche rapide mais de qualité moyenne ou une tâche de bonne qualité mais prenant plus de temps [Str 95] ;
- l'utilisation de méthodes basées sur le calcul imprécis [LLN87] ;

En conclusion, les algorithmes d'ordonnancement permettent d'établir « au mieux » les niveaux de priorité des tâches pour un contexte donné. Pour assurer la correction du système, il faut de plus développer des techniques propres à la vérification du respect des échéances. L'analyse d'ordonnançabilité, permet alors, sous certaines hypothèses sur les applications, de définir des conditions nécessaires et suffisantes d'ordonnançabilité simples et analytiques. Pour des tâches périodiques indépendantes (soient C_i : pire durée d'exécution de la tâche τ_i ; R_i : échéance de la tâche τ_i; T_i : période de la tâche τ_i,) à échéance sur requêtes (R_i=T_i), n étant le nombre de tâches ; ces conditions sont [LL 73] :

Condition suffisante, RM ou DM : $\sum_{i=1}^{n}\left(\frac{Ci}{Ti}\right) \leq n(2^{1/n} - 1)$,

soit une charge processeur maximale de 69% pour n *assez* grand

Condition nécessaire et suffisante, ED : $\sum_{i=1}^{n}\left(\frac{Ci}{Ti}\right) \leq 1$

soit une charge processeur maximale de 100%

Les conditions nécessaires et suffisantes proposées s'appliquent à des modèles d'applications simplifiés voire simplistes. Pour une configuration de tâches plus réaliste, des techniques analytiques d'évaluation d'une borne au pire temps de réponse d'une tâche [Ber 03] [TC 94] ont été développées. Parmi celles-ci, l'analyse RMA (« *Rate Monotonic Analysis* ») [Kle 93] est sans conteste la plus célèbre. Elle est intégrée dans des outils de développement [RRM 04] et supportée par un standard de modélisation [SPT 03] ; L'analyse RMA, comme les techniques d'analyse d'un pire temps de réponse [Ric 03], fournissent souvent des bornes non réalistes des pires temps de réponse et ne considèrent que des architectures statiques. Pour répondre à ces problèmes, il a été proposé plus récemment des approches basées sur la modélisation exhaustive de l'ensemble des comportements temporels du système [Pet 99][WH 03][LR 04][HGT 03].

II.2.3 Conclusion sur la vérification et la validation

L'ordonnancement temps réel, ainsi que les techniques d'analyse associées, forment un domaine de recherche en soi (voir les conférences du domaine telles RTSS (« *Real-Time System Symposium* » ou RTAS « *Real-Time Application Symposium* » et le journal RTS « *Real-Time System* »). L'objet du rappel effectué n'est pas de faire un état de l'art exhaustif, que l'on pourra trouver dans [Sha 04]. On retient de ces études les points importants qui influencent le développement des systèmes visés :

- la nécessité d'intégrer des techniques d'ordonnancement temps réel ;
- l'optimalité des algorithmes pour certaines classes de problème ;
- l'objectif de recherche d'une borne du pire cas d'exécution ;
- la vision simplifiée des architectures opérationnelles (tâches périodiques, graphe de précédence, …) ;

De leur côté, les techniques par modélisation exhaustive sont adaptées aux cas complexes où les techniques analytiques n'offrent pas de solutions réalistes. Au niveau de leur mise en place, on retient les points suivants :

- il existe des techniques formelles de validation pour des propriétés comportementales ;
- la vérification par simulation exhaustive est adaptée aux comportements dynamiques et complexes ;
- la mise en œuvre des techniques formelles nécessite de faire un choix de formalisme et d'outil adaptés pour faire un modèle correct et exprimer les propriétés à vérifier ;
- il est essentiel de maîtriser le phénomène d'explosion combinatoire.

Enfin, on note que la vérification de propriétés temps réel s'appuie sur les caractéristiques temporelles du système, en particulier les durées, qui sont fortement dépendantes de la construction finale du système. La vérification de propriété de QdS implique donc que le système ait été implémenté, ou que l'on puisse, a priori, prédire ou estimer les performances du système avant implémentation.

II.3 Modèles et paradigmes

Avant de valider et vérifier un système, il faut bien évidemment le construire. A cet effet, la littérature propose des langages et des principes architecturaux pour supporter les diverses étapes du développement, soient :

- des langages pour la prise en compte des spécificités du domaine que sont le support d'exécution et les contraintes de QdS ;
- l'introduction de la concurrence d'exécution dans les architectures ;
- l'amélioration de la qualité, au sens du génie logiciel, des architectures produites ;
- la formalisation des modèles produits afin de faire le lien avec les techniques formelles.

Nous présentons maintenant les principes exprimés par les travaux liés à ces domaines de recherche.

Les langages communément utilisés pour la description des systèmes (UML « *Unified Modeling Language* » [UML2], SDL, MSC « *Message Sequence Chart* » [MSC 96]) ne possèdent pas de mécanismes permettant de décrire précisément les contraintes de QdS et les supports d'exécution. De nombreux travaux proposent alors d'enrichir les langages existants :

- par ajout de nouveaux éléments syntaxiques [SDR 03][DHH 95] [BB 90] ou de commentaires formels [BL 97];
- par spécialisation des langages existants [BR 98] [LGT 98];

ou de définir de nouveaux langages :

- langage spécifique de spécification SA-RT « *Structured Analysis for Real-Time* » [HP 87] ; langage spécifique pour la description des architectures « *Architecture Description Languages* » [BEJ 96] [FGF04] [HHK 01] [FDT 01] ; langage spécifique, DSL « *Description Specific Language* », pour décrire des ordonnanceurs [ML 05] ;
- profils UML pour décrire les contraintes de temps [SPT 03] [MGD 01] ou la QdS [QoS 04] ;

Du fait de leur aspect réactif et afin de préparer l'analyse d'ordonnançabilité, un point essentiel de la construction des architectures porte sur l'introduction de la concurrence. On note alors un niveau d'abstraction plus ou moins important entre un langage de haut niveau, a priori indépendant de la cible d'exécution, et un langage proche de l'implémentation, par exemple LACATRE. Dans les premiers, afin de faciliter la mission du concepteur, la concurrence d'exécution est introduite à l'aide de concepts de haut niveau comme les objets actifs[9] et les objets passifs [GT 03]. Ces approches permettent de masquer au concepteur la gestion des tâches en définissant des contraintes de haut niveau sur l'implémentation. Elles proposent des boîtes à outils qui s'appuient sur des boîtes grises à instancier lors de la conception. C'est en particulier le cas des modèles objet Accord [TFB 96], IBM Rational Rose ® Real Time [Ros], Rhapsody ® [Rha], RTGOL [BS 98] et UML-RT [BR 98].

L'introduction de la concurrence lors de la construction du système pose plus globalement le problème de la sémantique d'exécution des architectures. La totale indépendance des architectures vis-à-vis de l'implémentation passe soit par l'absence de sémantique d'exécution, soit par la définition a priori de contraintes sur l'exécution, soit enfin par la mise en place d'outil performant lors du raffinement. D'un côté, l'absence d'informations sur l'exécution implique l'impossibilité de vérifier les propriétés temps réel de l'architecture. De l'autre, définir des contraintes a priori revient à effectuer un choix, qui peut s'avérer prématuré sur une sémantique d'exécution et sur les performances attendues du système. Enfin, les travaux sur la synthèse dans le cadre des systèmes multitâches sont récents et non encore formalisés, en particulier pour l'intégration des politiques d'ordonnancement (cf. chapitre II.4). En pratique, les architectures possèdent une sémantique d'exécution proche de celle définie par l'implémentation.

Au sujet des langages utilisés, afin de documenter plus efficacement les architectures produites, les formalismes graphiques semblent les mieux adaptés. Ils fournissent une vue globale et simple de la structure du système. De leur côté, les langages textuels apparaissent, eux, plus adaptés à la description des détails de sémantique ou

[9] Un objet actif contrôle l'exécution de ses services alors que les services d'un objet passif sont exécutés par un objet actif.

d'implémentation des systèmes. Dans les langages graphiques, les langages de modélisation UML et SA-RT permettent de réaliser simplement une description de la structure, du comportement et des contraintes des systèmes. De son côté, SDL est bien adapté pour la spécification comportementale de systèmes discrets. Parmi les ADL, on peut citer CLARA qui permet la description graphique de l'architecture applicative [Fau 02]. LACATRE, lui, est un langage spécifique pour la description de l'architecture opérationnelle multitâche basée sur un RTOS. Enfin, dans le domaine synchrone, on peut citer SyncChart [And 96] qui offre un langage graphique de haut niveau pour la programmation Esterel.

Afin d'être validé et vérifié, un modèle doit être exécutable, c'est à dire posséder une sémantique permettant son interprétation. Pour autant, la description d'une architecture s'appuie le plus souvent sur des langages, orientés utilisateurs, semi-formels (ADL, UML, SA-RT). Un langage semi-formel est un langage qui ne propose la description formelle que d'une partie du système (par exemple la structure avec les ADL), ou laisse certains points de sémantique à préciser lors d'étapes ultérieures de raffinement (*variation point* d'UML, sémantique informelle du comportement avec SA-RT). Afin de fournir une sémantique à ces langages, de nombreuses études se proposent de coupler des langages semi-formels avec les langages formels, en particulier pour préciser la sémantique temporelle. Parmi les nombreux travaux dans ce domaine, on peut citer les travaux associant UML à un langage formel [APR 02][EMP 03][GOO 04], SA-RT et les réseaux de Petri [ELP 93], COTRE (décomposé en « User COTRE » et « Verification COTRE »), l'ADL CLARA et les réseaux de Petri temporels [Dur 98].

Cette formalisation des modèles permet d'appliquer des techniques formelles à base de simulation exhaustive. L'analyse d'ordonnançabilité, elle, se base sur un modèle simplifiée et normalisée de tâches (cf. II.2.2). Ce modèle reste théorique et la formalisation passe par l'extraction des caractéristiques temporelles du système selon cette vue. Une première solution est de limiter les schémas de programme utilisés afin d'être compatible avec les modèles de l'ordonnancement temps réel [MGD 01]. A l'inverse, si l'architecture du système est complexe, il est nécessaire de développer un outil afin d'extraire l'ensemble des comportements temporels possibles de l'application, dans le respect du cadre de modélisation imposé, par exemple en découpant les actions en sous-actions [BC 96].

La maîtrise des architectures des SETR pose plus globalement le problème des relations entre les architectures applicatives, matérielles et opérationnelles. Dans ce contexte, les approches se proposent d'explorer les paradigmes communément utilisés dans les systèmes d'information tels que CBSE « *Component Based Sofware Engeeniring* » [Szy 98] et MDA « *Model Driven Architecture* » [MDA].

En exprimant clairement les dépendances requises d'un composant, l'approche CBSE aide à la structuration et à la composition de modules (composition de composants hétérogènes, déploiement d'un composant applicatif sur une plate-forme). De son côté, l'approche MDA, en faisant une séparation claire entre les modèles indépendants des plates-formes et les plates-formes elles-mêmes, permet le déploiement d'architectures applicatives sur des plates-formes hétérogènes.

Pour autant ces approches peinent à s'imposer dans le domaine des SETR. En effet, les implémentations existantes de ces approches considèrent des systèmes classiques (systèmes d'information, interfaces utilisateur, systèmes distribués, applications WeB) sans

prise en compte des contraintes de QdS liées à l'utilisation de ressources limitées. De plus, à l'exécution, le système utilise généralement les services d'un intergiciel de type CORBA « *Common Object Request Broker Architecture* » [CORBA] ou d'une machine virtuelle de type Java [Java]. Les architectures considérées ne possèdent pas alors les propriétés de prédictibilité des SETR.

Des solutions existent pour la description des SETR en suivant le paradigme CBSE, mais restent limités à une classe de systèmes particuliers : il s'agit généralement de décrire une organisation spécifique du code en s'appuyant sur un RTOS sous-jacent pour un type de contrainte de QdS donnée [Pec 99] [OLK 00] [SAH 01] [UVH 01][FSL 02] [IN 02]. Au niveau de MDA, les études proposent d'abstraire les architectures matérielles (machines [BDD 03], capteurs/actionneurs [HB 03]) pour assurer l'indépendance vis-à-vis des plates-formes. Malheureusement, ces études n'intègrent pas la gestion des contraintes de QdS liées à l'utilisation de ressources limitées. L'utilisation de paradigme de haut niveau requiert donc une adaptation des principes pour intégrer les contraintes de QdS et s'appuyer sur des plates-formes d'exécution prédictibles.

En conclusion, le développement des SETR requiert des modèles pour décrire les architectures, les contraintes de QdS, introduire la concurrence et les politiques d'ordonnancement. Ces modèles doivent être compréhensibles par un concepteur, posséder une sémantique formelle pour assurer leur correction et assurer un lien vers l'implémentation. Leur mise en place requiert alors l'utilisation et l'adaptation des langages et paradigmes classiquement utilisés dans les systèmes d'information, soient :

- l'utilisation de formalismes graphiques pour décrire les architectures et les contraintes de QdS ;
- l'adaptation des paradigmes pour intégrer des contraintes de QdS et des supports d'exécution prédictibles ;
- l'utilisation de boîtes grises à instancier lors du développement ;
- la définition d'une sémantique d'exécution (concurrence et ordonnancement), généralement proche de celle définie par les RTOS ;
- la définition d'une sémantique formelle, souvent guidée par les modèles issus des techniques formelles.

La modélisation des architectures est alors un compromis entre des modèles de bas niveau (maîtrise des performances mais moins de réutilisation) et des modèles à haut niveau d'abstraction, masquant les détails d'implémentation (modèles réutilisables mais pouvant être difficilement exploitables dans un contexte embarqué et temps réel). Les langages proposés dans la littérature font généralement une hypothèse implicite sur les supports d'exécution supportés et les règles de structuration sont dictées par la nécessaire prédictibilité des architectures produites. Par exemple, dans l'approche définie par [KNA 02], un composant applicatif n'est activable que périodiquement et par un seul stimulus, ceci dans le but de simplifier l'étape de validation et d'implémentation. De même, dans l'approche MAST [MGD 01] le modèle objet considéré par l'approche est implicitement limité aux schémas de programme admis par l'analyse RMA (tâches périodiques avec relation de précédence). Au final, les règles de structuration des architectures sont souvent conduites par l'implémentation et la validation. Les approches se placent implicitement au niveau des architectures opérationnelles afin de maîtriser les caractéristiques de QdS.

II.4 Transformations

Au delà des outils et techniques pour la description des architectures et des contraintes, la recherche porte sur l'automatisation des transformations inhérentes au développement par étape.

Pour le raffinement, les approches proposent des techniques pour traduire un modèle de haut niveau vers un modèle plus proche de l'implémentation en précisant certains aspects. Par exemple, [MKM 00] propose de produire une architecture matérielle, décrite en VHDL, à partir d'une spécification SDL en précisant le protocole de communication à base de boîtes aux lettres. Pour les SETR, un point clé du raffinement est l'introduction de la concurrence. Déjà en 1993, l'approche CODARTS [Gom 93] proposait des règles informelles pour guider le concepteur dans la mise en place d'une architecture multitâche à partir d'une spécification SA-RT. Nous avons de notre côté proposé [SJH 99] une traduction automatique d'une spécification décrite à l'aide d'automates temporisés de type CRSM (« *Communicating Real time State Machines* », automates communicants temporisés [Sha 92]) vers une implémentation multitâche LACATRE. Dans la même idée, des études [WH 01] proposent des heuristiques de placement des activités applicatives sur les tâches du système. En fait, vu le nombre de raffinements envisageables entre un modèle de haut niveau et une implémentation multitâche, de plus en plus d'approches proposent de rationaliser la traduction par l'évaluation de critères de performance [GH 05] [FSA 05].

En fait, si on met en place des hypothèses fortes sur la sémantique d'exécution du modèle de haut niveau, la traduction automatique, soit la génération automatique de code, est possible. C'est le cas des approches synchrones où il est possible de générer un code C à partir d'une spécification formelle. Pour les approches asynchrones, le code est généré à partir de l'architecture opérationnelle [Sch 95] [TFB96] ou à partir de l'architecture applicative, de l'architecture matérielle et de règles de placement des entités applicatives sur les tâches [Tau] [Fau 02]. Dans ce cas, une machine virtuelle permet d'implémenter les entités concurrentes et les éléments de communication de l'architecture applicative en s'appuyant sur les services de l'exécutif sous-jacent. Un langage peut être alors dédié à la modélisation de la projection des entités applicatives sur les machines et les tâches [Sta 01].

Dans un contexte temps réel, la génération de code doit intégrer les politiques d'ordonnancement temps réel. A ce jour, les politiques considérées par les modèles et outils sont soit figées (ordonnancement à priorité fixe dans les modèles objets Rhapsody, Rose Real-Time, UML-RT, ED pour le modèles objet Accord et RTGOL) soit à implémenter manuellement sur le code généré. L'opération peut alors s'avérer complexe. Par exemple, dans un contexte objet, afin d'éviter le phénomène d'inversion de priorité, il est nécessaire d'implémenter manuellement des politiques spécifiques à héritage de priorité pour la gestion des files d'attente [SPF 98], soit en modifiant la machine virtuelle proposée, soit en modifiant le code généré.

Pour les opérations d'optimisation de la QdS, les efforts portent sur la minimisation de l'utilisation de ressources comme la réduction de la quantité mémoire utilisée [HMK 96] ou la limitation du nombre de tâches par la mise en place de *pool* de tâches [CS 03].

Les techniques de transformation existantes sont limitées à une classe de problème et font très tôt une hypothèse forte sur la sémantique d'exécution. Il n'existe pas à ce jour de techniques et d'outils assurant une mise en place optimisée de la concurrence et des politiques d'ordonnancement à partir d'un modèle de haut niveau pour aboutir à un code embarquable. Pour pallier ce manque, une solution est alors d'effectuer une transformation manuelle dont on valide a posteriori la correction. Le concepteur peut toujours s'inspirer de principes établis pour construire ses architectures et son implémentation, mais reste libre de ses choix. Pour l'aider, une solution est de définir des solutions types validées pour des classes de problèmes identifiés, soit des *design pattern* [GHJ 95] pour le temps réel [Dou 02]. Les solutions proposées restent informelles et doivent s'appuyer nécessairement sur une maîtrise fine du comportement temporel des architectures opérationnelles.

II.5 Méthodes

Une méthode a pour objectif de couvrir l'ensemble des étapes du cycle de développement en définissant les différentes étapes du développement et leur enchaînement. Par exemple, le déploiement est découpé en deux étapes, une de placement des composants sur les machines et une sur les tâches, assurant ainsi la construction du système par raffinements successifs [AKZ 96]. Les méthodes proposent pour l'ensemble des étapes des langages et des principes de modélisation, des règles de transformation et d'implémentation et des techniques formelles pour assurer la correction des systèmes.

Par exemple, DESS[10] [LQV 01] s'appuie sur UML pour la modélisation de la structure et une sémantique temps réel et formelle (Esterel, Kronos) pour la vérification. De son côté, EMPRESS[11] [EMP 03] s'appuie sur UML-RT, lui même basé sur ROOM « *Real-Time Object-Oriented Modeling* » [SG 94] pour la modélisation de la structure et du comportement. La sémantique est fournie par UML+, automates temporisés selon [KP 92]. L'implémentation s'appuie sur le fait que UML-RT peut produire directement du code en utilisant les outils existants comme Rational ROSE-RT®. Dans le projet OMEGA[12] [GH 04], la description du système s'appuie sur le langage UML, les diagrammes de séquence et une structuration à base de composants. OMEGA s'appuie sur IF, les PVS [SOR 93], les LSC[DH 01] et OCL[WK 98] pour préciser la sémantique temporelle, comportementale et les contraintes. Enfin, une technique à base d'observateurs permet d'exprimer les propriétés à vérifier.

Les méthodes sont le fruit de coopérations nationales ou internationales mélangeant des partenaires académiques et des industriels (projets européen OMEGA, EAST-EEA, DESS, EMPRESS, COTRE [FGF 04]) ou le fruit de l'expérience industrielle (méthode Octopus de NOKIA [AKZ 96], AUTOSAR[13]). Elles nécessitent en effet de maîtriser, les langages de modélisation, les techniques formelles et de proposer des cas d'étude réalistes.

[10] DESS « Software Development Process of Real-Time Embedded Software Systems » : ce projet ITEA définit un processus à base de composants pour les applications temps réel, www.dess-itea.org

[11] EMPRESS « Evolution Management and Process for Real-time Embedded Software Systems » : projet ITEA pour la gestion des évolutions dans les SETR, www.empress-itea.org

[12] OMEGA « Correct Development of Real-Time Embedded Systems » : projet IST pour la définition d'une méthodologie de développement, basée sur UML, pour les SETR, www-omega.imag.fr

[13] AUTOSAR « Automotive open system architecture » : consortium d'industriel pour la mise en place de standards ouverts pour les architectures automobiles, www.autosar.org

II.6 Conclusion

La correction d'un SETR est fortement liée à la mise en œuvre de la concurrence au travers des politiques d'ordonnancement utilisées. Les choix effectués lors de l'implémentation sont fondamentaux car ils influencent directement les performances de QdS et la correction des systèmes. Afin d'améliorer le processus de développement, il est pour autant nécessaire de raisonner sur des modèles de plus haut niveau que le code, soit les architectures. Ces dernières permettent de se concentrer sur la structure du système en masquant les détails non pertinents d'implémentation. La description des architectures applicatives s'appuie implicitement sur une sémantique d'exécution et donc sur une architecture opérationnelle sous-jacente. De même, la génération automatique d'une architecture opérationnelle à partir d'un modèle de haut niveau requiert la maîtrise des performances de cette dernière. L'architecture opérationnelle, vue comme une abstraction de l'implémentation, est donc au cœur du processus de développement des SETR. Afin d'améliorer sa mise en place, il faut alors travailler à définir sa sémantique et proposer un cadre de qualité, au sens du génie logiciel, pour l'organisation du code, soient :

- spécialiser les langages et paradigmes existants en intégrant des contraintes de QdS, une sémantique de concurrence et des techniques d'ordonnancement ;
- proposer des solutions orientés utilisateurs (aspect graphique, boîte grises) ;
- réaliser un compromis entre un modèle dédié à une plate-forme et un modèle abstrait, générique et indépendant des plates-formes ;
- fournir une sémantique formelle pour assurer la correction des architectures ;
- utiliser des techniques formelles en définissant des méthodologies de mise en œuvre ;
- fournir des règles de génération de code en assurant leur correction.

Dans les études menées, nous nous plaçons au niveau des architectures opérationnelles multitâches pour des systèmes monoprocesseur utilisant un RTOS. Afin d'aider à maîtriser leur mise en place, nous proposons de travailler à leur formalisation et à leur structuration.

II.7 Nos problématiques de recherche

Le L3i a développé des compétences dans les environnements de conception graphique d'application multitâche avec le langage LACATRE [Sch 95]. La partie graphique permet de concevoir le squelette opérationnel de l'application. Le code textuel est généré automatiquement et sert de squelette à la partie fonctionnelle de l'application. Dans le cadre de mon intégration au L3i, j'ai d'abord étudié, en collaboration avec l'équipe Slovène de Maribor, à des adaptations de LACATRE pour intégrer l'exécutif temps réel PEARL [Hal 91] et pour assurer la description de l'architecture matérielle [GCB 98]. Je me suis aussi intéressé à l'extension de LACATRE pour le système d'exploitation Windows® [Alk 99]. J'ai enfin participé à des travaux sur la traduction automatique d'une spécification exprimée à l'aide des CRSM vers une architecture multitâche [SJH 99]. Ces études sur la mise en place, manuelle ou automatique, d'une architecture multitâche ont posé la question de la validité de l'architecture vis-à-vis des contraintes exprimées, en particulier vis-à-vis d'une spécification formelle de type CRSM.

Pour la vérification de propriétés temps réel, je m'étais déjà intéressé, lors de ma thèse, à la vérification de propriétés temps réel pour des architectures multitâches au comportement statique [BC 98]. Une étude menée au L3i sur l'utilisation des CRSM a démontré la pertinence d'une traduction de LACATRE vers les automates temporisés CRSM pour la vérification de propriétés comportementales [HLM 99]. Il était alors intéressant de se poser la question de la vérification de propriétés temps réel sur des architectures multitâches au comportement dynamique.

Ces deux questions autour de la validation et de la vérification posent plus généralement la question de la formalisation d'une architecture multitâche pour la mise en place de techniques formelles. Pour la formalisation, nous avons considéré le langage SDL, car il est formel, asynchrone et que de nombreuses études proposent des extensions à SDL pour la description des SETR. De son côté, IF est un formalisme de haut niveau, dont la sémantique comportementale est identique à celle de SDL et qui permet de combler les lacunes de SDL pour la description de l'écoulement du temps.

Au sujet des techniques formelles, nous avons constaté qu'il existe des techniques de validation pour les propriétés comportementales basées sur les relations d'équivalence. Il était alors intéressant de s'intéresser à leur mise en œuvre pour la validation comportementale d'une architecture opérationnelle multitâche vis-à-vis d'une spécification. Le point dur est ici de considérer des comportements temporels dynamiques de l'environnement (par exemple des rafales de stimuli) et asynchrones de l'application (mémorisations, décalages temporels). Pour la vérification de propriétés liées au temps physique (sûreté, échéances) sur une implémentation multitâche, pour considérer des architectures au comportement dynamique, il était intéressant d'évaluer des techniques par simulation exhaustive.

Ces problématiques liées à la formalisation pour la validation et la vérification des architectures opérationnelles multitâches a été l'objet des études qui ont conduit aux thèses d'Ahmad Alkhodre [Alk 04] et Mostefa Belarbi [Bel 03].

Au-delà de la formalisation, l'amélioration de la maîtrise des architectures opérationnelles passe par une structuration de qualité, au sens du génie logiciel. L'approche orientée objet est largement reconnue et utilisée pour implémenter les principes du génie logiciel et ainsi aider à la maîtrise de la complexité croissante des architectures. Elle est apparue dans les années 90 comme une solution pour améliorer la structuration des systèmes temps réel [TT 92][NT 93][TFB 96][Bab 00]. Dans ce sens, le projet RTGOL « *Real-Time Graphical Object oriented Language* », mené au sein du L3i, propose une architecture à base de méta-objets propice à la mise en place de politiques d'adaptation temps réel [Str 95] [LLN 87]. RTGOL propose une description des contraintes temps réel et de leur sémantique de propagation au sein des objets, une description de la concurrence d'exécution et l'intégration de politiques d'ordonnancement [BS 97] [BS 98]. Enfin, la mise en place d'objets interfaces assure une séparation claire entre les communications avec l'environnement externe et le cœur de l'application. RTGOL apparaît comme particulièrement adapté à la structuration des systèmes temps réel souple [CS 03].

Ces études sur les approches objets m'ont amené à m'intéresser à d'autres paradigmes pour la structuration des systèmes comme l'approche CBSE. Dans ce domaine,

France Telecom R&D a développé une compétence forte en définissant un modèle abstrait générique de composants, appelé Fractal [BCS 03] et un *framework* pour le domaine de l'embarqué de Fractal appelée Think [FSL 02].

Think permet la construction optimisée d'application et d'exécutifs dédiés. Par contre Think n'intègre pas la prise en compte de politique de gestion de la QdS liées à l'utilisation de ressources limitées. Nous avons donc proposé de collaborer avec France Telecom autour des modèles à composants pour la mise en place d'une architecture de gestion de la QdS dans les systèmes embarqués mobiles. Le point dur est ici de considérer des comportements dynamiques de l'application et de l'environnement (arrivée/arrêt d'un composant, profils variables de QdS) ainsi que des contraintes hétérogènes de QdS (temps réel dur et souple, contraintes diverses de QdS).

On peut trouver, dans la littérature, des principes pour la gestion dynamique de la QdS dans les systèmes distribués, des architectures pour la gestion dynamique de contraintes de QdS telles que la sécurité ou la persistance dans les systèmes informatiques classiques, et des architectures de gestion statique de la QdS dans les systèmes embarqués. En nous inspirant de ces travaux, nous avons proposé une architecture de gestion de la QdS, appelée Qinna , à base de composants, pour des SETR à composants.

Ce sujet d'étude a permis de lancer la thèse de Jean-Charles Tournier [Tou 05] en collaboration avec France Telecom R&D.

En résumé, les premiers travaux de recherche menés au L3i de 1997 à 2000 se sont focalisés sur la description et la mise en place d'architectures multitâches pour des systèmes temps réel. Depuis 2000, mon objectif de recherche a été de travailler à la formalisation et à la structuration des architectures opérationnelles pour des SETR afin d'assurer leur correction et d'améliorer la qualité, au sens du génie logiciel, des architectures produites.

Nous présentons maintenant les travaux menés autour de la formalisation des architectures à l'aide de SDL et de IF.

III Développement formel à l'aide de SDL et IF

Afin de formaliser la description des architectures opérationnelles multitâches, nous proposons d'utiliser le langage formel SDL et de mettre en œuvre des techniques formelles par modélisation exhaustive.

L'étude vise les systèmes temps réel confrontés à un comportement dynamique.

III.1 Introduction

La maîtrise d'une architecture requiert de s'appuyer sur des modèles structurés, formels et à haut niveau d'abstraction, masquant les détails d'implémentation. L'objectif de la formalisation est de permettre l'utilisation de techniques formelles pour valider l'architecture vis-à-vis des propriétés attendues, ici des propriétés temporelles.

Nous considérons un comportement dynamique de l'environnement (rafale d'interruption) et du code (création/destruction de tâches, plusieurs modes de fonctionnement) et nous travaillons au niveau de l'architecture opérationnelle.

Le langage SDL est un langage formel, normalisé par l'ITU-T[14], asynchrone, à haut niveau d'abstraction, dédié à la description des systèmes discrets [Mam 00]. De plus, de nombreuses études proposent des extensions des outils pour la description, la validation et l'implémentation des SETR à l'aide de SDL. En particulier, IF [BFG 99] possède la même sémantique comportementale que SDL, possède une sémantique d'écoulement du temps physique et offre un point d'accès vers les techniques formelles basées sur la simulation exhaustive.

Notre objectif est alors de proposer un cadre d'utilisation du langage SDL pour la description des architectures opérationnelles multitâches et la mise en pace de techniques formelles pour la validation et la vérification de ces architectures.

Avant de présenter notre contribution, nous résumons les travaux existants autour des deux domaines de recherche concernés par l'étude, soient l'utilisation de SDL pour la description des SETR et l'utilisation de techniques formelles pour valider des propriétés comportementales et vérifier des propriétés temps réel. Puis, nous concluons par un bilan et des perspectives.

III.2 Etat de l'art

III.2.1 SDL pour les SETR

Principes de SDL

Pour décrire l'architecture d'un SETR, un langage doit être structurant, formel, à haut niveau d'abstraction et intégrer des éléments pour décrire les contraintes de temps et de concurrence. Les réponses apportées par SDL sont les suivantes :

- SDL assure la structuration d'un système SDL via une décomposition modulaire arborescente en *bloc* dont les feuilles sont des *process*. Un *process* peut être créé ou détruit dynamiquement ;

[14] ITU-T : International Ttelecommunication Union, organisme de normalisation, www.itu.int/home/

- le comportement de chaque *process* est modélisé, de façon déterministe, par un diagramme d'état. Les *process* s'exécutent en parallèle et communiquent entre eux de manière asynchrone via des signaux. Au final, la composition des *process* définit la sémantique comportementale du système ;
- SDL possède une version graphique et une version textuelle ;
- le langage d'action et la description des données, basée sur les types abstraits de données, sont indépendants d'un langage d'implémentation ;
- SDL inclut les notions d'héritage et de généricité (typage des *blocs*, *process* et données).

SDL permet donc une modélisation graphique et formelle, à haut niveau d'abstraction et la structuration d'architectures concurrentes au comportement dynamique. De plus, il existe des outils industriels, Real-TimeDeveloperStudio™ [RTD] et Tau™ [Tau], qui offrent des possibilités d'édition, de simulation, de test et de génération de code pour la mise au point des modèles. Enfin, c'est un langage utilisé dans l'industrie que nous avons expérimenté avec succès pour le prototypage de systèmes embarqués [BA 01][Tou 02].

Par contre, pour la description d'une architecture opérationnelle dans le contexte des systèmes temps réel, SDL doit être étendu pour :
- fournir une sémantique temporelle ;
- préparer la génération de code.

Formalisation de SDL pour le temps réel

SDL ne possède pas d'élément de spécification clair et précis de l'écoulement du temps et ne fournit pas une description complète de la façon dont le modèle doit s'exécuter dans le temps [BGK 00]. Il n'est donc pas possible d'assurer l'activation d'une transition à une date exacte et de définir des durées d'exécution. Il s'ensuit qu'il n'y a aucun moyen de vérifier, en l'état de la sémantique, si le modèle respecte bien les contraintes temps réel exprimées. Afin de répondre à ce manque, les travaux peuvent être classés en deux catégories.

Les premières approches (TSDL [BB 90], QSDL [DHH 95], SDL-RT [SDR 03]) proposent d'enrichir SDL par addition des nouveaux éléments syntaxiques ou de commentaires formels. Dans les secondes, on couple SDL à un outil capable de combler ses lacunes au niveau de la modélisation du temps (par exemple de SDL vers les automates temporisés IF [BGM 01] [Gra 02] ou UPPAAL [He 02]). Ces approches enrichissent les modèles SDL avec les informations suivantes :
- dates de tir des transitions, dates de génération d'événement cycliques ;
- durées des actions, durée et fiabilité des communications;
- unité de temps spécifiée en temps physique ou en *tick* ;
- dérives sur les horloges locales ;

Une fois la sémantique temporelle définie, on peut passer à une étape de validation. Les outils commerciaux sont orientés vers la spécification, la simulation ou le test des aspects fonctionnels et comportementaux. De leur côté, l'utilisation de langages formels tels IF, QSDL, TSDL, UPPAAL permet d'assurer une validation temporelle. Par exemple, l'outil QUEST, basé sur QSDL, permet, par l'analyse du graphe de couverture, de réaliser des évaluations de performance [QUE 97].

Une fois un modèle SDL construit et validé, on peut passer à la phase d'implémentation.

SDL et l'implémentation

Afin de produire un code temps réel à partir d'un modèle SDL, il est nécessaire de fournir des informations supplémentaires au modèle original. Il faut d'abord différencier les parties modélisant l'environnement du système des parties réactives. Parmi ces dernières, il faut définir un partitionnement logiciel/matériel [SSD 97]. Pour la partie logicielle, il faut modéliser le déploiement des *process* sur l'architecture matérielle. A cet effet, il existe trois approches :

- dans l'approche « intégrée », on représente le support d'exécution à base de *bloc*-type SDL modélisant les machines et les tâches ; le déploiement consiste alors à insérer les *process* dans ces blocs ; enfin on annote le modèle pour fournir des informations sur les performances des machines et les priorités des tâches [SDR 03] [WT 04];
- dans l'approche « orthogonale », on décrit, dans un autre formalisme que SDL, l'architecture matérielle (machines, tâches, mediums de communication et exécutifs), et le placement des *process* sur les entités d'exécution [Tau];
- dans l'approche par synthèse, on annote simplement le modèle avec les contraintes temps réel à respecter. Un outil de synthèse se charge ensuite de produire le code adéquat [ADL 03].

A partir des informations sur le déploiement, des générateurs de code implémentent le comportement des systèmes en s'appuyant sur un RTOS sous-jacent [Tau][SDR 03][ADL03]. L'optimisation du code porte sur l'insertion de code C dans le modèle SDL [SDR 03] ou sur l'optimisation de la place mémoire [HMK 96].

Méthodes

Il existe peu de méthodes pour le développement de SETR à l'aide de SDL. On peut citer les travaux récents menés sur la modélisation des systèmes temps réel à l'aide de SDL et des langages de trace MSC [WT 04]. Ces travaux proposent d'utiliser les MSC pour décrire les séquences d'actions à réaliser par le système. Ce modèle est annoté de contraintes temporelles que sont les périodes d'activations, les délais et les pires durées d'exécution des actions. Ces dernières sont donc, supposées être connues avant placement des actions sur les machines. SDL sert, lui, à représenter une architecture matérielle distribuée, en modélisant les machines par des *bloc* SDL et les mediums de communication par des canaux SDL. Ce modèle est ensuite annoté pour fournir des informations sur les caractéristiques des machines et réseaux. Enfin, l'approche propose d'utiliser des heuristiques [WH 01] pour le placement des actions sur les machines et les tâches. L'architecture opérationnelle est obtenue par synthèse des modèles SDL et MSC et sert de base à la vérification temps réel, basée sur une analyse du pire temps de réponse [WF 00].

Les principes généraux de cette approche sont l'utilisation d'un langage pour décrire les besoins (MSC annotés) et les architectures (*bloc* type SDL et commentaires formels). Par contre, l'approche n'intègre pas des comportements dynamiques mais considère des systèmes distribués. Enfin, l'approche ne permet pas d'évaluer diverses implémentations, l'architecture opérationnelle étant obtenue par synthèse.

Conclusion sur SDL pour la description des systèmes temps réel

Pour le développement de SETR, au niveau des méthodes, on note l'intérêt de posséder plusieurs modèles, dont un exprimant les contraintes, en particulier de QdS, et un décrivant l'architecture. Pour la description des architectures et des contraintes, l'utilisation de SDL s'appuie sur des extensions ou des spécialisations du langage afin de décrire la sémantique temporelle et d'exécution.

Pour exprimer la sémantique temporelle, l'approche par modification syntaxique de SDL implique des adaptations sémantiques du modèle SDL modifié. Des analyses de performance sont possibles mais le modèle n'est plus conforme au modèle SDL originel. La mise en place de commentaires formels permet de conserver le modèle original tout en fournissant des informations pouvant être interprété par un outil adéquat. Enfin, la traduction (sans modification de la sémantique comportementale) de SDL vers un autre langage assure la coexistence de SDL avec son extension. L'avantage d'une telle approche est la totale indépendance qui existe entre la spécification étendue et la spécification SDL. La spécification des aspects non fonctionnels peut être entièrement détachée de SDL pour la vérification de propriétés spécifiques. Dans le cas, par exemple de IF, il est alors intéressant de définir un comportement standard en SDL, puis d'affiner le comportement temporel sur le modèle IF associé.

Pour intégrer la sémantique d'exécution, l'approche par modification syntaxique possède les désavantages cités précédemment. La mise en place de commentaires formels ou la spécialisation de la structure SDL par des informations supplémentaires (tâches, niveaux de priorité) permet de conserver le modèle original tout en fournissant des informations structurelles pouvant conduire la génération de code. Enfin, l'approche par synthèse d'un modèle de haut niveau (seules les contraintes sont exprimées) requiert un travail important pour la génération de code temps réel efficace, souvent limitée à une solution pré-programmée (heuristique).

III.2.2 Validation et vérification

Principes de IF

Au niveau de la sémantique temporelle, IF correspond à une représentation « à plat » d'un modèle SDL (suppression de la structure hiérarchique) (cf. figure 2). La structure étant neutre vis-à-vis du comportement, le modèle IF possède le même comportement que le modèle SDL dont il est issu. De plus, IF fournit une sémantique d'écoulement du temps. On peut ainsi contraindre le domaine temporel de tir des transitions par des dates fixes ou des intervalles de tir. L'évolution du temps se fait de manière discrète et lorsqu'une transition est tirable, elle est tirée en temps nul. Comme IF ne propose pas d'invariant d'état, il préserve l'absence d'interblocage sur la composition de *process* ne possédant pas d'interblocages. Enfin, la transmission des signaux peut être perturbée (ré-ordonnancement des signaux, perte, retard, diffusion).

Une fois un modèle IF établi, il est possible de l'optimiser (outil IF2IF) en supprimant les données inutiles [Muc 97] et le code mort. On peut ensuite générer le diagramme LTS correspondant à l'ensemble des comportements possibles du système et utiliser ainsi les outils formels de la boîte à outils CADP : Aldebaran, Evaluator et TGV [GLM 02]. En particulier, Aldebaran implémente un ensemble de relations d'équivalence utiles à la validation des systèmes. De son côté, Evaluator permet de vérifier des propriétés de logique temporelle.

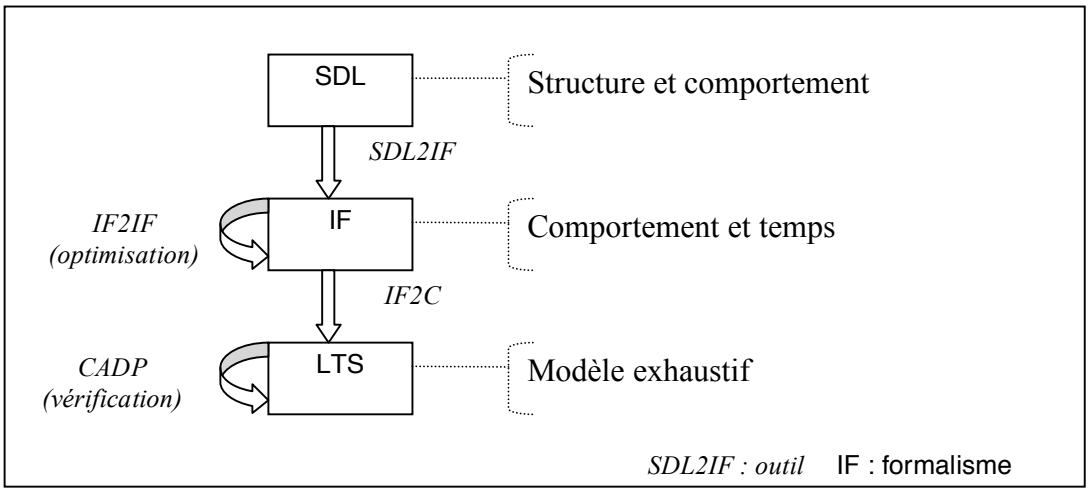

Figure 2: outils et formalismes pour la mise en place de techniques formelles implémentées par l'outil CADP sur une description de système SDL en utilisant IF.

Validation comportementale

Afin de valider la transformation de modèles, on doit s'assurer que les modèles, avant et après transformation, sont équivalents. A cet effet, on trouve dans la littérature plusieurs relations d'équivalence dont les plus connues sont la bisimulation forte [Par 81], l'équivalence de trace [Man 74], l'équivalence observationnelle [Mil 80], l'équivalence de branchement [GW 96] ou encore la F-simulation [RSR 01].

La relation de bisimulation forte exprime le fait que deux modèles représentent exactement le même comportement. L'utilisation classique de cette relation est la minimisation d'un LTS, vis-à-vis du nombre d'états et de transitions, pour trouver une écriture minimale d'un modèle donné (cf. figure 3.1).

La bisimulation forte implique que toutes les propriétés sont conservées lors de la transformation. Au delà de l'équivalence stricte entre deux modèles, on est souvent amené à valider, lors d'une transformation, la conservation d'un ensemble de propriétés. Dans ce cas, la relation d'équivalence doit être adaptée aux propriétés à conserver. Par exemple, lorsqu'on veut conserver une propriété de trace, l'équivalence porte sur le langage exprimé. Par exemple, si on souhaite conserver la propriété « on a *a* puis, *b* ou *c*», les automates de la figure 3.2 sont dits trace-équivalents car ils simulent cette propriété de deux façons différentes. L'équivalence de trace est adaptée à la conservation de propriétés de logique temporelle linéaire (pas de mémorisation des chemins).

Lors d'un raffinement, le comportement du système est modifié par l'ajout d'informations, soient de transitions pour un LTS, sur le modèle initial. Les modèles ne possèdent alors pas le même alphabet (ensemble des labels). Ils ne peuvent donc pas avoir la même trace ni, a fortiori, être fortement bisimulables. Dans ce cas, l'équivalence des modèles se base sur l'observation des labels communs aux deux automates, labels dits observables. Les labels non observables sont masqués par une action invisible, communément notée τ. L'équivalence observationnelle des modèles exprime alors le fait que le modèle transformé simule le modèle initial, du point de vue des actions observées.

On ne tient alors pas compte des transitions non observables τ (cf. figure 3.3). On parle aussi parfois de bisimulation faible.

L'équivalence observationnelle ne permet pas de conserver certaines propriétés liées à la structure des systèmes. Pour pallier ce manque, d'autres relations d'équivalence ont été définies comme l'équivalence de branchement qui conserve les propriétés sur les chemins (cf. figure 3.4).

Toutes ces approches, ou leurs versions dérivées, considèrent que les systèmes possèdent, d'une certaine façon le même comportement. Or il arrive que la correction d'un système ne soit effective qu'à partir d'un certain moment, soit par exemple après une phase d'initialisation. La F-simulation exprime alors le fait qu'une conception est F-équivalente à une spécification si et seulement si le LTS de conception engendre le même comportement que le LTS de la spécification après une certaine séquence d'événements (cf figure 4).

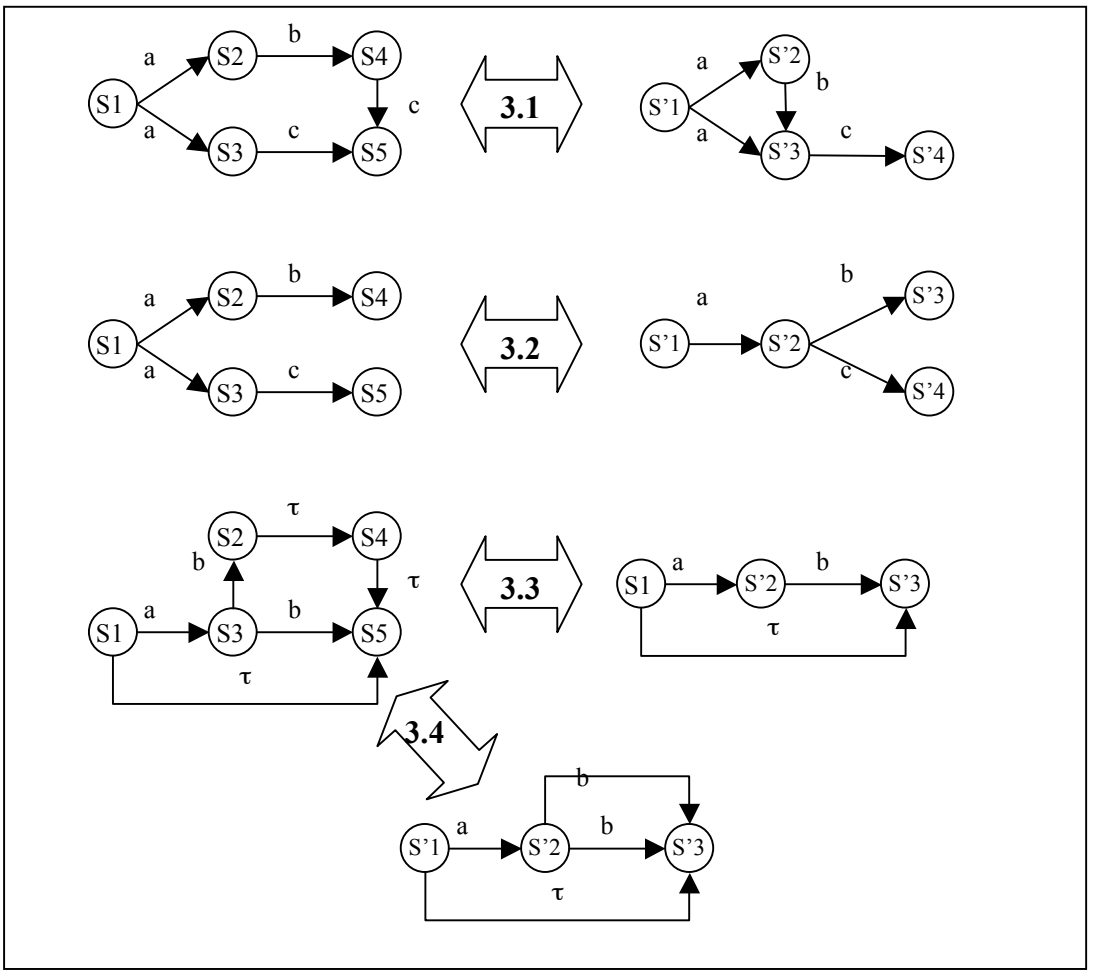

Figure 3: illustration des relations d'équivalence : 3.1 : bisimulation forte, 3.2 : équivalence de trace, 3.3 : équivalence observationnelle, 3.4 : équivalence de branchement.

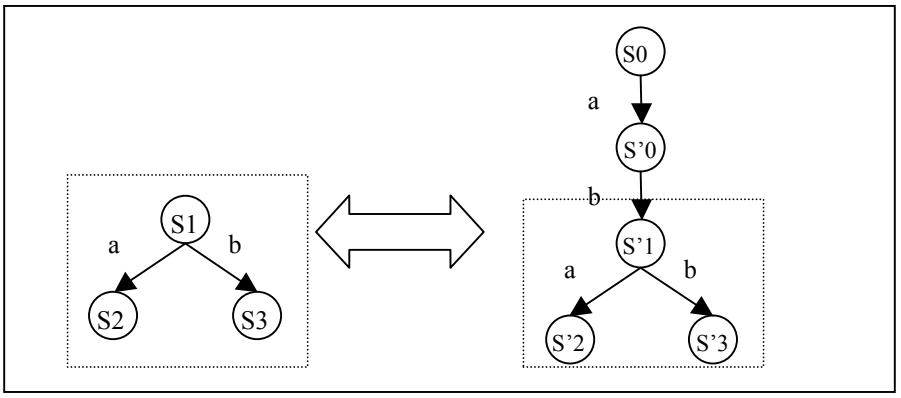

Figure 4: illustration de la F-équivalence : s1 et s'1 sont bisimulables

En conclusion, la notion d'équivalence dépend des propriétés visées et doit être spécialisée avec précaution selon le domaine concerné. Par exemple, l'équivalence observationnelle peut ne pas conserver des états de blocage (transition τ vers un état terminal) et n'est donc pas adaptée à la validation de propriétés de sûreté. Valider l'équivalence de deux modèles revient donc à préciser les abstractions sur les modèles (mise en place des actions invisibles) et à choisir ou définir une relation d'équivalence.

Après les travaux sur la validation comportementale, nous présentons les travaux autour de la vérification de propriétés temps réel à base de modélisation exhaustive.

Vérification basée sur une simulation exhaustive

Pour faire une analyse réaliste des pires temps de réponse d'un système multitâche au comportement dynamique, nous préconisons d'explorer les techniques par simulation exhaustive. Cette approche est d'ailleurs de plus en plus étudiée dans la littérature pour la vérification d'échéances dans les systèmes multitâches [Pet 99][WH 03][LR 04][HGT 03] et plus généralement pour la modélisation des systèmes temps réel [Gra 02][AGS 02].

La mise en place des techniques formelles passe par le choix d'un formalisme, la mise en place correcte des modèles, la réduction de ces derniers et le choix d'une technique de vérification (logique). Les études référencées ci-dessus proposent de produire un modèle formel du comportement temporel de l'application à base d'automates temporisés communicants. Le modèle du comportement temporel représente l'environnement (procédé à contrôler), l'architecture applicative (usuellement, un automate modélise chaque entité concurrente), la politique d'ordonnancement et le support d'exécution :

- au niveau de l'environnement, on modélise les dates d'arrivée des événements en provenance du procédé à contrôler et les dates de mise à jour de données lues sur le procédé. Le temps discret est ici suffisant car on modélise des dates d'occurrences d'événements ;
- au niveau de la politique d'ordonnancement, comme il n'existe pas de relation d'ordre dans la sémantique des automates communicants (deux transitions tirables sont tirées dans un ordre aléatoire), on peut soit imposer une relation d'ordre par synchronisation des entités concurrentes, soit imposer une relation d'ordre lors de la simulation du modèle, soit modéliser explicitement un ordonnanceur via un automate spécifique. La première solution est adaptée à un ordonnancement à priorité fixe. La deuxième permet de simuler tout type d'ordonnancement (par

exemple ED) mais oblige à instrumenter le simulateur du modèle. Enfin, la dernière solution est adaptée à la modélisation d'ordonnanceurs complexes ;
- Pour modéliser la préemption, comme il n'existe pas de possibilité de préempter une transition, on doit ajouter deux états « *en-exécution* » et « *préempté* » pour chaque automate modélisant les entités concurrentes ;
- pour la modélisation effective des durées d'exécution, le temps passé dans un état « *en-exécution*» ou « *préempté*» ne peut être connu a priori. En effet, ce temps dépend de l'analyse du comportement temporel global du système. Pour pallier ce problème, une tâche est modélisée soit par une séquence de sous-actions non préemptibles, soit par une séquence de durées unitaire, dans le cas d'un ordonnancement préemptible. Une autre solution est d'utiliser des techniques intégrant des notions de gestion du temps telles que proposées par les automates hybrides [Hen 96]. Il est alors possible d'arrêter une horloge (préemption de la tâche) et de la réinitialiser (élection de la tâche). Enfin, il est à noter que le temps discret permet de ne modéliser qu'un cas (par exemple le pire cas) ou un ensemble de cas (différents modes d'exécution) ;
- l'architecture matérielle est vue au travers des durées d'exécution des tâches et des temps de communication dans un système distribué.

Une fois la modélisation du système établie, les propriétés à vérifier sont modélisées par l'utilisation de logiques ou la mise en place d'observateurs. Pour limiter le phénomène d'explosion combinatoire, au-delà des techniques de réduction fournies par les outils, les solutions considèrent implicitement des modèles abstraits de l'implémentation. Par exemple, on ne modélise pas tous les éléments de l'application tels les routines d'interruption, le contrôleur d'interruption, les mediums de communication entre les tâches et enfin les durées sont fixes.

III.2.3 Conclusion sur SDL, IF et les techniques formelles pour les SETR

SDL est un langage graphique permettant la structuration et la description formelle du comportement de systèmes à événements discrets concurrents et dynamiques.

Afin de conserver la sémantique des modèles, pour la description des architectures opérationnelles multitâches, nous préconisons d'une part la traduction des modèles SDL vers un autre langage, en l'occurrence IF, pour définir une sémantique d'écoulement du temps. Pour fournir les éléments nécessaires à la génération de code nous proposons une spécialisation de SDL par introduction de *bloc*-type pour décrire la structure du support d'exécution (machines et tâches) et des commentaires formels pour préciser les politiques d'ordonnancement.

Au niveau de la méthode, nous proposons d'utiliser SDL pour exprimer à la fois les besoins et décrire l'architecture. L'utilisation de IF permet alors l'utilisation de techniques formelles pour valider le comportement de l'architecture vis-à-vis des besoins exprimés et vérifier des propriétés temps réel.

Pour la validation, on note que les relations d'équivalence concernent des propriétés comportementales et sont à adapter selon le type de propriétés. En particulier, les relations doivent être étendues pour considérer des flots d'événements tels que définis par la logique temps réel [JM86]. En effet, la notion d'occurrence n'existe pas dans les automates temporisés communicants. De son coté, la vérification réaliste de propriétés temps réel de

systèmes au comportement dynamique passe par la mise en place des techniques formelles par simulation exhaustive. On note que l'abstraction de medium de communication entre tâches ne permet pas de valider des propriétés sur le code telles que « on ne dépose jamais de message dans une boîte aux lettres pleine ». Or, ces propriétés sur l'état des mediums de communication dépendent du comportement temporel du code, soit l'ordonnancement des actions de dépôt et de retrait dans la boîte aux lettres.

Au final, le développement formel passe par la mise en place d'un cadre formalisé pour aboutir à la mise en place d'architectures types, validées, et réutilisables. Nous proposons donc :

- de spécialiser SDL pour décrire formellement une architecture opérationnelle multitâche et d'utiliser IF pour fournir une sémantique d'écoulement du temps ;
- d'adapter les techniques de validation et de vérification pour des comportements dynamiques.

III.3 Notre contribution

III.3.1 Architecture

Principes généraux

Notre approche définit trois niveaux, dont deux exprimés en SDL (cf. figure 5). Le premier niveau, issu de la spécification et noté modèle de contraintes MC, permet de formaliser les fonctionnalités attendues du système ainsi que les contraintes temps réel. Sa mise en place est nécessaire pour valider l'architecture opérationnelle. Dans un deuxième niveau qui est au cœur de la proposition, on décrit l'architecture opérationnelle, appelée ici modèle d'exécution ME, supposée s'appuyer sur une machine monoprocesseur. L'objectif de l'étude étant de proposer un cadre formel à la description de cette architecture opérationnelle, les principes suivis pour la mise en place du modèle d'exécution sont laissés libre au concepteur. Enfin, le troisième niveau correspond à l'implémentation basée sur un RTOS. Le modèle d'exécution décrivant l'ensemble des éléments nécessaires à l'exécution du système, le squelette opérationnel du code (tâches, éléments de communication) est obtenu automatiquement par un générateur de code.

Afin d'aider le concepteur à décrire son système, on propose pour chaque niveau (MC et ME), une boîte à outil d'éléments type SDL (signaux, *process*, *bloc*). Chaque élément possède un comportement prédéfini et est à instancier lors de la description du système. Pour chaque niveau, il existe un modèle exprimé en IF pour préciser la sémantique temporelle. (SMC, SME et SMI pour *Semantic* MC et *Semantic* ME *Semantic Model of Implementation*). Le comportement IF est obtenu par traduction directe du comportement SDL (outil SDL2IF) pour les deux premiers niveaux. L'écoulement du temps est alors précisé par l'instrumentation des gardes temporelles. Le code ne possédant pas de modèle SDL associé, nous avons proposé des règles de modélisation spécifiques pour produire le SMI. Enfin, les LTS générés à partir de IF servent de base à la validation et la vérification du système. On valide le comportement du modèle d'exécution vis-à-vis du modèle de contraintes et on vérifie le respect de propriétés de sûreté et temps réel sur le code final.

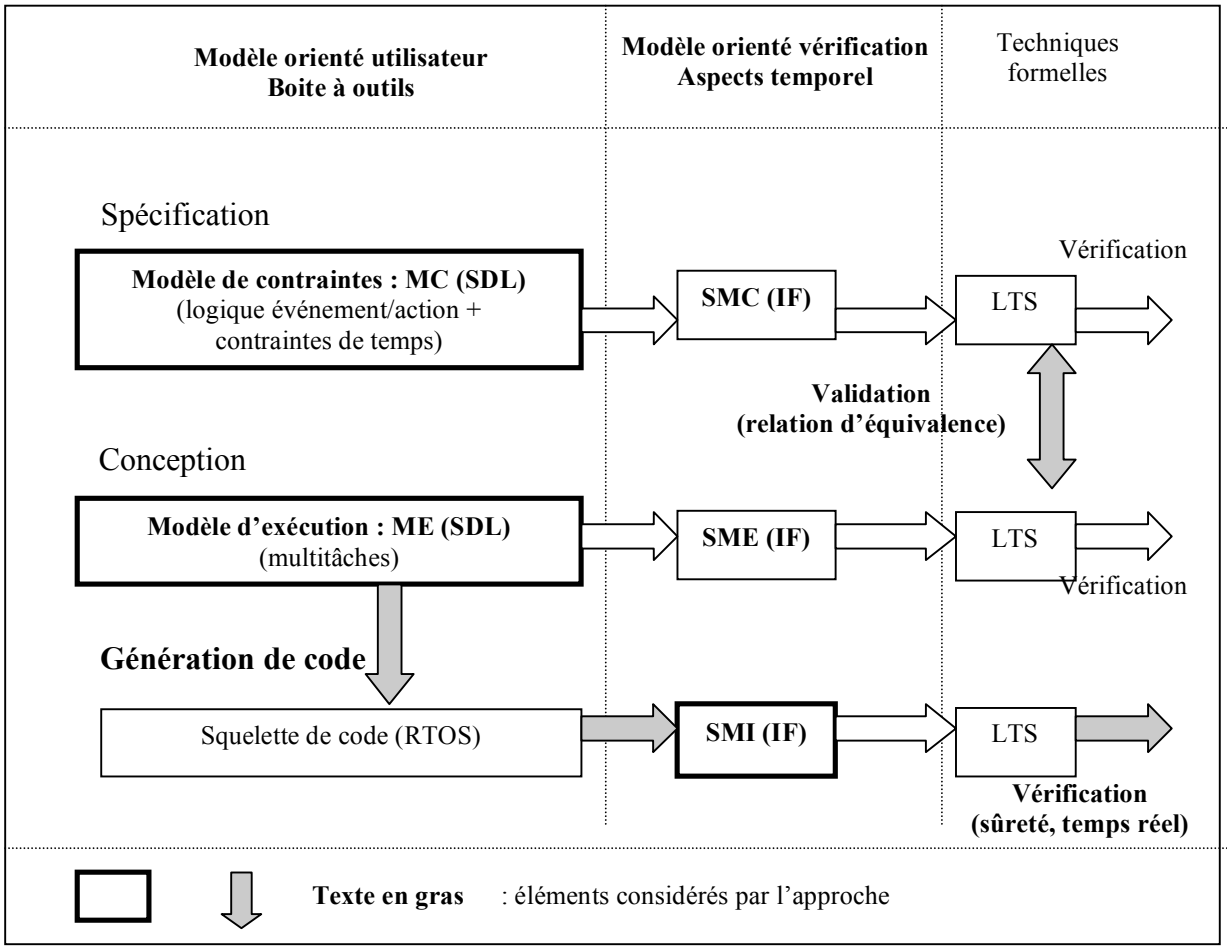

Figure 5 : étapes du développement formel de systèmes temps réel à l'aide de SDL et IF

Modèle de contraintes

Dans les systèmes temps réel, la spécification décrit, d'une part, l'environnement à piloter et d'autre part la partie réactive, soient les fonctionnalités attendues du système. Comme, on se limite au comportement temporel du système, l'environnement est modélisé via des *process* type qui émettent, à des dates spécifiées, des stimuli vers la partie réactive. Les stimuli sont modélisés par des signaux SDL typés (mise à jour de donnée périodique, émission d'un événement sporadique, …) et véhiculent les contraintes temporelles (date d'arrivée, durée de validité, échéance). La partie réactive suit le paradigme classique des systèmes réactifs « Evénement - Condition – Action ». Une action est ici simplement modélisée par un label. L'expression des contraintes temporelles en paramètre des signaux permet de détailler l'expression de politique temps réel spécifiques (filtrage temporel, réaction à une faute temporelle) par des *process* type. La formalisation du modèle IF assure la possibilité de réaliser des vérifications comportementales sur les modèles de contraintes et d'exécution (vivacité, sûreté, …). Cet aspect n'a pas été évalué lors de l'étude.

Modèle d'exécution

Une fois les contraintes exprimées, l'étude se focalise sur la phase essentielle de définition de l'architecture opérationnelle. C'est à ce niveau que l'on décrit comment l'application doit s'exécuter par l'introduction des tâches. L'approche propose une boîte à outils composée de de *blocs* type qui introduisent les notions de routines (d'interruption ou d'alarme), de tâches (périodique, sporadique ou serveur) et des règles d'interconnexion de ces divers éléments.

Le modèle d'exécution permet de préciser le comportement temporel de l'application en fixant les dates exactes d'activation des actions. La spécification décrit un comportement logique du système : suite à l'occurrence d'un événement, selon l'état du système, une action doit être effectuée dans un temps borné. En s'appuyant sur des entités d'exécution données (tâches, serveurs à scrutation, éléments de synchronisation et de communication, priorités), on introduit des informations précises pour définir, à quelle date, suite à l'occurrence d'un événement, l'action doit être activée. Par contre, si on précise quand les actions sont effectivement activées, on ne sait pas quand elles sont effectivement exécutées. Ceci reste le résultat de l'analyse d'ordonnançabilité, dépendant des durées d'exécution fournies par l'implémentation.

L'ensemble des contraintes de concurrence étant figé, le modèle d'exécution apparaît comme une abstraction de l'implémentation.

Implémentation

Afin de vérifier la complétude du modèle d'exécution, nous proposons de réaliser un générateur automatique de code. Il est à noter que la construction du code ne doit pas modifier le comportement temporel défini par le modèle d'exécution. Un prototype a été réalisé qui permet de définir les opérations à réaliser lors de la génération de code, soient :

- les routines et les tâches sont générées à l'identique du modèle d'exécution ;
- la communication entre entités (sémantique de communication FIFO imposée par SDL) s'appuie sur les éléments fournis par l'exécutif, sémaphore booléen ou boîte aux lettres FIFO. La taille maximale des éléments de communication dépend de la dynamique du système et est vérifiée a posteriori (cf. chapitre III.3.2) ;
- la politique d'ordonnancement et les niveaux de priorités sont imposés dans le modèle d'exécution par des commentaires formels. Selon les niveaux fournis par le RTOS, il peut être cependant nécessaire de réaliser un ajustement des niveaux [LS 86].

Pour valider les principes de génération de code, nous avons considéré deux cibles bien distinctes : un OS classique, en l'occurrence type Win32®, et un RTOS standard, en l'occurrence VxWorks®. Il est évident que la mise en place du générateur de code aurait pu faire l'objet d'une étude plus approfondie, par exemple en terme d'optimisation du code, mais ce n'était pas l'objectif de l'étude et c'est un travail de recherche en soi (cf. chapitre III.4.2 sur les perspectives à cette étude).

III.3.2 Vérification et validation

L'avantage d'avoir exprimé la sémantique en *IF* est de pouvoir utiliser des outils pour vérifier des propriétés temporelles et valider la correction de la transformation du modèle de contraintes vers le modèle d'exécution.

Validation

Il faut s'assurer que la mise en place du modèle d'exécution, en particulier dans le cadre de mémorisations et de décalages temporels, n'aboutit pas à la perte ou à l'activation d'actions non souhaitées. Afin d'établir cette correction, nous avons défini une méthodologie et proposé une règle d'équivalence originale.

Les deux modèles de contraintes SMC et d'exécution SME sont d'abord traduits en leur LTS équivalent. Afin de les comparer, un renommage des labels et la mise en place d'une base de temps commune sont nécessaires pour assurer la cohérence des deux modèles. Après vérification de l'absence d'interblocage des LTS obtenus, une réduction est effectuée afin de ne conserver que les labels correspondant aux événements et actions définis comme observables par la spécification.

Selon les cas, les modèles obtenus ne sont pas équivalents. En effet, au niveau de la spécification, lorsqu'un événement survient, l'action correspondante doit être activée immédiatement (cf. figure 6.a). Alors, qu'à l'exécution, du fait des mémorisations et des décalages temporels, plusieurs événements peuvent être mémorisés avant qu'une action soit déclenchée (cf. figure 6.c). Cette configuration se retrouve en particulier lors de la prise en compte des rafales d'interruption. Pour intégrer cet effet, nous avons proposé une relation d'équivalence, notée N-équivalence, des LTS obtenus. Cette équivalence reste comportementale, elle dépend du comportement temporel d'activation des actions et doit permettre de contrôler que toute action à activer est activée et qu'aucune action non activable ne doit être activée.

Pour toute réaction définie par le SME par le comportement « l'événement *e* déclenche l'action *a* », noté « *e-a* », on accepte au niveau du SMC toutes les permutations de « *e-a* », soient : « *e-a* » (bisimulable), « *e-a-e-a* » (bisimulable), « *e-e-a-a* » (non bisimulable) (cf. figure 6). Par contre, les comportements « *a-e-e-a* » , « *e-a-a* » ou « *e-a-a-a* » ne sont pas considérés comme corrects. En effet, soit il n'existe pas d'action, label « *a* », pour chaque événement, label « *e* », soit il existe une réponse mais sans événement déclenchant correspondant.

Au fond, si on considère chaque occurrence de l'événement *e*, notée *e(i)*, le SME réduit est bien fortement bisimulable avec la propriété exprimée par la spécification : il existe une occurrence *a(i)* par événement *e(i)* (cf. figure6.b). Par contre, un modèle de spécification intégrant l'ensemble des occurrences de *e* aboutirait à un modèle infini et le modèle d'exécution ne représente lui qu'un ensemble de cas parmi l'ensemble des combinaisons possibles. Pour conclure, la relation d'équivalence proposée permet de considérer les effets d'une implémentation asynchrone (retards et décalages) pour une spécification synchrone. Le modèle de contraintes définies donc implicitement un sur-ensemble, l'ensemble de toutes les permutations possibles, de comportements acceptables.

Il est à noter que la relation de N-équivalence dépend fortement du comportement temporel de l'environnement. Une architecture opérationnelle est donc considérée comme temporellement correcte vis-à-vis d'une spécification que pour un contexte d'exécution donné.

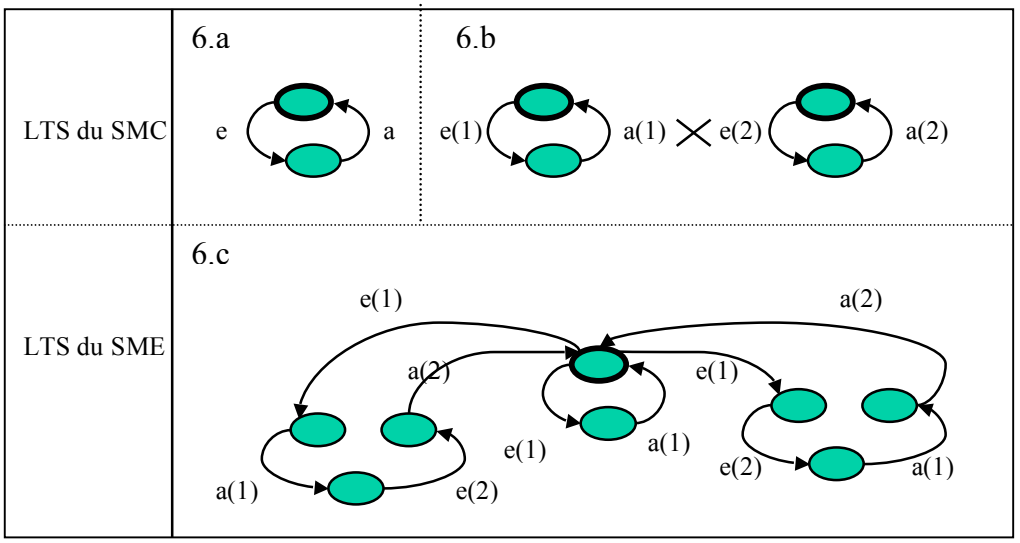

Figure 6 : illustration d'un comportement N-équivalent pour un couple « événement-action » (« e-a ») défini par le SMC où l'occurrence de l'événement peut être deux fois active au niveau du SME.

Vérification

Pour ce qui est de la vérification des échéances, nous avons d'abord proposé la mise en place d'une analyse de type RMA. Cette étude est à rapprocher des études déjà menées dans le même domaine [MGD 01]. Si elle n'apporte pas de propositions originales, elle permet de vérifier la complétude des modèles et la possibilité d'effectuer des analyses temps réel.

Comme noté au chapitre II.2.2 sur la vérification temps réel, l'analyse RMA est adaptée à des comportements statiques, afin de traiter des schémas de programme au comportement dynamique (création/destruction dynamique de tâches, durées variables), il nous est apparu opportun de développer des techniques spécifiques de vérification par simulation exhaustive. Nous ne faisons pas d'hypothèses sur l'organisation du code autre qu'un ordonnancement préemptif à niveau de priorité fixe, classique dans le domaine visé. L'objectif est de vérifier des propriétés temps réel auxquelles nous ajoutons des propriétés de sûreté sur le code. Les premières portent sur le respect des contraintes temps réel de type échéance. Les dernières portent sur la bonne utilisation des objets mis en place par le RTOS, en particulier les éléments de communication entre tâches, soient les sémaphores et les boîtes aux lettres.

Comme noté au chapitre II.2.1 sur la mise en œuvre des techniques formelles, il est nécessaire de choisir un formalisme, modéliser le système et vérifier la correction des modèles, réduire le phénomène d'explosion combinatoire et mettre en place une technique de vérification. Dans notre approche, le formalisme a été choisi, en l'occurrence IF.

Le modèle IF se concentre sur le comportement temporel du système au travers des quatre éléments qui sont : l'environnement, l'application, le RTOS et l'architecture matérielle. L'environnement est modélisé via des *process* IF qui donnent les lois d'arrivée des événements en provenance du procédé. Cette approche permet de décrire des comportements temporels classiques (événement périodique ou sporadique) mais aussi complexe (rafales). Ensuite, chaque objet de l'exécutif temps réel est modélisé via un

process, soient un *process* par entité d'exécution (tâche ou routine) et par élément de communication (sémaphore ou boîte aux lettres). Le comportement de chaque *process* dépend évidemment des spécificités de l'exécutif considéré. L'architecture matérielle impacte le comportement temporel du code au niveau des durées d'exécution et de la gestion des interruptions. Un *process* modélise alors le comportement du contrôleur d'interruption. Afin de modéliser un ordonnancement préemptif, une tâche est découpée en une séquence de durées unitaires.

Si on ne précise pas un ordre d'exécution entre les *process,* l'entrelacement des transitions aboutit au phénomène d'explosion combinatoire. A cet effet, le modèle défini un ordre d'exécution des *process* compatible avec l'ordonnancement réel des éléments de l'application par le RTOS. On définit quatre classes de priorité (décroissante) correspondant à l'environnement (automates générateurs d'événements externes), au matériel (contrôleurs et routines d'interruption), à l'exécutif (éléments de communication) et enfin à l'application (les tâches). Dans les trois premières classes, les *process* étant indépendants entre eux (hypothèse de l'étude), quelque soit leur ordre d'exécution, l'état final observable pour les propriétés étudiées est le même. Afin de réduire les entrelacements, on propose alors d'appliquer une réduction d'ordre partiel par classe de priorité. Pour la dernière classe, on s'appuie sur l'ordre de priorité des tâches pour définir un ordre d'exécution des *process*.

Une autre technique pour la réduction du modèle est d'utiliser les modes d'exécution de l'application. L'application est alors vue comme une séquence de modes : phase d'initialisation de l'application, phase moteur, mode dégradé, arrêt du système. Chaque mode est caractérisé par un état de début et un état de fin. Un état du système correspond à l'ensemble des états des éléments de l'application. Par exemple, la fin de la phase d'initialisation est caractérisée par un état global ou chaque élément est dans un état « créé ». Une fois les modes défini, le modèle est décomposé en un ensemble de modèles par mode, obtenus par abstraction des autres modes. Au final, les propriétés sont vérifiées par mode.

Au niveau des propriétés, nous proposons d'abord des propriétés de sûreté, locales à un *process* IF. Elles concernent :

- l'utilisation des objets, par exemple : « un sémaphore utilisé a été préalablement créé (initialisé) et non détruit ». Cette vérification est impossible statiquement à la compilation car elle dépend du comportement dynamique de l'application modélisée ;
- le respect des tailles maximales des éléments de communication, par exemple : « on ne dépose jamais de message dans une boîte aux lettres pleine ». Cette propriété de type « producteur/consommateur » dépend évidemment de la dynamique de l'application ;
- la qualité du code, par exemple : « on ne dépose pas de jeton dans un sémaphore booléen à vrai » ou « un sémaphore n'est pas libéré du fait de l'expiration d'un *timeout* ».

Ces propriétés locales sont vérifiées par l'ajout d'un état *error* ou *warning* aux transitions concernées. Ensuite, il suffit de vérifier sur le LTS les propriétés de logique temporelle : « le système n'est jamais dans l'état *error* ou *warning*. Lorsqu'une erreur est détectée, les outils fournissent l'enchaînement d'événement qui aboutit à l'erreur et apporte ainsi une aide au concepteur pour corriger le système.

Nous proposons aussi de vérifier des propriétés temps réel, globales à plusieurs *process* IF. Elles expriment des contraintes sur :

- un intervalle maximum entre deux événements, par exemple pour spécifier une échéance;
- un intervalle minimum entre deux événements, par exemple pour exprimer une contrainte sur la régularité de productions d'un événement en sortie .

Les propriétés globales sont vérifiées par l'ajout, dans le modèle IF, d'un observateur modélisé par un *process* IF. L'observateur a pour rôle d'observer le comportement temporel du système. Dans notre approche, l'observateur mesure les performances (utilisation de compteurs) et vérifie les propriétés temps réel. La mise en place d'un observateur requiert d'instrumenter le modèle par l'ajout d'envoi de signaux vers l'observateur dans les transitions concernées. Les hypothèses à vérifier par l'observateur sont :

- être non intrusif, soit pas d'émission de signaux vers le système observé ;
- le seul blocage admis est lié à des transitions vers un état d'erreur ;
- l'automate est déterministe ;
- l'observateur est plus prioritaire que le système observé.

Dans les études menées, on ne considère qu'une seule instance d'observateur active à la fois. Pour assurer le suivi d'un flot d'événements, on considère que l'arrivée d'un nouvel événement alors que le précédent n'a pas été traité est une erreur.

Pour valider l'approche et le modèle, nous avons comparé les résultats obtenus à ceux qui sont fournis par des techniques analytiques [LSD 89] sur un exemple simple et montré que l'approche fournit des estimations réalistes des pires temps de réponse. Enfin, au niveau de la complexité des modèles, pour un système composé de huit éléments (trois tâches dont un *main*, deux routines d'alarme, trois sémaphores dont un signalant la fin de l'application), le nombre d'états obtenus est d'environ mille états. Pour information, si on ne considère que la phase moteur de l'application (phase définie par l'état de départ « tous les éléments sont créés » et l'état de fin « un jeton est déposé dans le sémaphore de fin »), le nombre d'état redescend à une centaine, soit un nombre faible pour la simulation exhaustive.

III.3.3 Conclusion

L'approche propose une séparation claire entre les modèles orientés utilisateurs pour la description du système et les modèles orientés pour les techniques formelles. Les premiers, basés sur le langage graphique SDL, s'appuient sur une sémantique applicative : événement et actions pour la spécification [ABS-1 02], tâches et routines pour l'architecture opérationnelle [ABS-3 02]. Les seconds sont basés sur une sémantique comportementale et temporelle, propice à la validation et à la vérification.

Afin d'aider le concepteur, l'approche propose des boîtes grises à instancier pour décrire les modèles de contraintes et d'exécution. Cette approche permet la réutilisation de composants, de construire et valider des assemblages génériques mais aussi de contrôler la construction du système en limitant les schémas de programme disponibles.

Le modèle de contraintes permet de décrire un comportement logique, annoté de contraintes de temps, et est à rapprocher de SA-RT ou UML. Il possède une sémantique synchrone évitant de modéliser tous les cas acceptables, soit un entrelacement de situations possibles. Le modèle d'exécution est à rapprocher de LACATRE. Il permet de décrire l'architecture multitâche avec abstraction des mediums de communication. Il possède une sémantique asynchrone afin de gérer les comportements temporels complexes, en particulier la communication avec l'environnement externe. Enfin, le code possède aussi une sémantique asynchrone afin d'assurer un comportement dynamique du système à l'exécution.

La mise en oeuvre des techniques formelles a demandé une extension des relations d'équivalence pour prendre en compte les phénomènes de mémorisation et de décalage temporel. Au niveau de la vérification, l'approche préconise, au delà des techniques proposés par les outils, de s'appuyer sur la sémantique des applications (priorités et modes de fonctionnement) pour réduire le phénomène d'explosion combinatoire.

III.4 Bilan et perspectives

III.4.1 Bilan

L'approche développée propose une boîte à outils pour la description d'une architecture opérationnelle multitâche et des propriétés attendues à l'aide de SDL. La boîte à outils SDL est une boîte grise instanciable dont le comportement est prédéfini par des *blocs* type SDL [ABS-2 02]. Ensuite, l'approche se focalise sur la sémantique temporelle des modèles afin d'assurer la validation des propriétés comportementales selon la logique événement-action et la vérification de propriétés temps réel. Enfin, l'architecture est implémentable sur un système monoprocesseur utilisant les services d'un RTOS.

L'originalité de l'approche est de spécialiser SDL tout en conservant sa sémantique et d'utiliser ses capacités de généricité (type). Les travaux proposent ensuite des méthodes de validation et vérification originales (nouvelle relation d'équivalence et observateurs à rapprocher des études récentes menées dans [Graf 04]). La relation d'équivalence intègre les phénomènes de mémorisation et de décalages dus à une implémentation asynchrone [AKB 04]. Par rapport aux approches existantes, la méthodologie de vérification s'appuie sur un modèle réaliste, non abstrait, de l'implémentation (modélisation des mediums de communication et des routines d'interruption) [BBS-1 04] et propose des règles de modélisation et d'abstraction pour contenir le phénomène d'explosion combinatoire (ordre partiel, modes de fonctionnement) [BBS-2 04]. La vérification, considérant un modèle fin du code, permet par la même occasion de traiter diverses propriétés de sûreté du code.

On peut enfin noter que les études apportent un début de réponse pour la prise en compte de la notion d'occurrence d'événement par des formalismes basés sur les automates temporisés communicants. Cette notion, formalisée par un compteur d'événements dans la logique temps réel [JM 86], est utile pour caractériser un flot d'événements ou de données.

III.4.2 Perspectives

Nous proposons maintenant des perspectives à ces travaux par l'extension de la relation d'équivalence, l'amélioration et l'extension des modèles pour la vérification, la mise en place de règles de génération de code optimisées et par l'utilisation d'autres langages et formalismes.

La relation d'équivalence proposée est limitée au cas simple où la spécification exprime que « un événement déclenche toujours une action ». Il serait intéressant d'étendre cette relation à des cas plus complexes où un événement déclenche une action en fonction de l'état du système. Pour étendre la relation proposée, il faudrait alors intégrer les retards dus à l'implémentation pour la mise à jour de l'état du système. Ces retards entraînent des états intermédiaires du système non décrits dans la spécification. On entrevoit alors deux pistes. La première est de reprendre la spécification en intégrant des aspects asynchrones pour la prise en compte des événements tels que proposé par le langage asynchrone ELECTRE [ER 86]. Une autre piste serait de définir et formaliser une nouvelle relation d'équivalence pour ce type de comportement. On pourra alors s'inspirer des travaux récents [FVC 05] qui définissent une relation d'équivalence de type observationnelle, notée ε-proximité, entre une spécification et une implémentation intégrant des décalages temporels dus à l'implémentation.

La relation d'équivalence est définie en rapport à une spécification et pour un comportement fixé de l'environnement. Il serait alors intéressant de définir une règle d'équivalence pour une famille de comportements possibles de l'environnement. Par exemple, on pourrait assurer la validité d'une architecture pour des stimuli périodique dont la période est comprise entre Pmin et Pmax, ou pour des stimuli en rafale dont le nombre de rafales est compris entre nmin et nmax.

La relation d'équivalence revient à considérer une implémentation asynchrone pour une spécification synchrone. Pour les systèmes distribués, de plus en plus de propositions vont dans le sens des approches GALS (Globalement Asynchrone et Localement Synchrone) [Cha 84]. Selon cette approche, le système est distribué sur des nœuds synchrones, les nœuds communiquent de façon asynchrone [BCG 99] [BCG 97]. Dans ce contexte, on devrait pouvoir étendre la relation d'équivalence proposée pour prouver la conformité d'une implémentation distribuée asynchrone par rapport à une spécification synchrone. En effet, la distribution entraîne des retards et des mémorisations selon le même principe qu'une distribution de code multitâche.

Nous proposons d'étendre et généraliser la mise en place des observateurs pour vérifier des propriétés temps réel sur une architecture multitâche. L'objectif est alors de fournir des règles formalisées et automatiques de mise en place d'observateurs dans des automates temporisés communicants afin de traduire des propriétés exprimées en logique temps réel [JM 86]. En particulier, l'observateur doit intégrer la notion d'occurrence d'événement (plusieurs occurrences actives simultanément). Ceci requiert de formaliser le type de propriété à considérer, fournir des règles de traduction automatique des propriétés vers un observateur et de fournir les hypothèses sur le modèle observé pour assurer la correction de l'observation.

Nous avons modélisé le procédé externe uniquement via des dates d'envoi d'événements. Cette approche, quoique classique en temps réel, considère implicitement

que les événements externes en provenance du procédé sont indépendants les uns de autres et possèdent un comportement constant au cours du temps. En fait, dans les systèmes de contrôle de procédé, la réaction à un stimulus produit l'envoi d'un signal vers le procédé qui modifie l'état et donc le comportement du procédé. Il serait alors nécessaire de réfléchir à une modélisation plus fine du procédé pour intégrer l'impact du comportement du procédé sur la production de stimuli.

Pour les systèmes complexes, il serait intéressant de définir une méthodologie de vérification par étapes. On pourrait par exemple décomposer le système en modules indépendants. L'indépendance, étant une notion relative, ce terme reste à préciser. Puis par module ou composant logiciel, on pourrait établir la vérification de propriétés locales au module, par exemple pour les éléments de communication internes au module. Puis l'approche devra proposer des règles de réduction des modèles, par exemple par abstraction des éléments de communication. Les modèles réduits permettront d'aboutir à une représentation minimale du comportement temporel de chaque module et serviront de base à la vérification de propriétés globales. Cette perspective pose plus généralement le problème ouvert et complexe de la composition de composants et de contraintes temporelles.

Dans les modèles proposés pour la vérification, l'approche se limite à des politiques d'ordonnancement simples (priorités fixes et pas de politiques de gestion de priorité en présence de ressources critiques). Une extension au travail serait d'intégrer des politiques d'ordonnancement dynamique (ED) et de gestion de ressources (héritage de priorité, priorité plafond). Il serait alors intéressant de mettre en place un *process* IF spécifique modélisant l'ordonnanceur, tel que proposé par [AGS 02]. Cela permettrait en plus de découpler l'application de sa politique d'ordonnancement et ainsi de vérifier la correction de l'application pour un ensemble de politiques d'ordonnancement.

La vérification temps réel s'appuie le code, mais on aurait aussi pu instrumenter le modèle d'exécution avec les durées d'exécution, mesurée à l'implémentation, et réaliser une étude d'ordonnançabilité sur le modèle d'exécution. Cette approche permettrait de produire un ordonnancement « hors-ligne » adaptée aux systèmes critiques. Cette approche permettrait plus généralement d'ouvrir des perspectives sur la synthèse d'ordonnanceur pour des applications multitâches, soit trouver un ordonnancement valide pour une configuration de tâches donnée.

Lors de la génération de code, pour implémenter la communication asynchrone entre *process* SDL, on utilise les services de l'exécutif sous-jacent d'échange de messages par boîte aux lettres. Lorsque l'échange concerne deux *process* servis par une même tâche, cette approche aboutit à un code peu efficace : la tâche doit se renvoyer à elle-même un message. Une solution optimisée serait de remplacer la communication par message par un appel direct de l'opération concernée (communication synchrone). Il faudrait alors définir formellement les hypothèses sur le comportement du système pour assurer la conformité des deux modèles de communication.

De manière plus générale, un SETR possède de nombreuses contraintes de QdS dont par exemple celles liées à un espace mémoire limité. Les règles de génération de code proposées doivent donc être optimisées vis-à-vis de ces contraintes. Une piste est de limiter le nombre d'objet temps réel à l'exécution, par exemple : une même tâche doit pouvoir servir deux entités concurrentes n'étant jamais actives au même instant. Par contre, l'étape

de génération de code ne doit pas modifier la sémantique d'exécution du modèle d'exécution. Il faudrait donc développer, en parallèle des techniques d'optimisation, des méthodologies de validation d'une implémentation, basées sur la bisimulation forte (équivalence stricte des comportements).

L'arrivée d'UML2 annonce le déclin de SDL en tant que langage de modélisation. Pour autant, les travaux présentés dans ce rapport restent pertinents. Dans le cadre de l'utilisation de langage asynchrone à haut niveau d'abstraction, on doit travailler à préciser la sémantique temporelle des modèles. On doit aussi travailler à la mise en place de générateurs de code dédiés efficaces. Enfin on doit assurer des ponts vers des outils de validation formels. A cet effet, on pourra s'appuyer sur le récent projet IST OMEGA qui propose un profil pour le temps réel, nommé *OMEGA-RT*, permettant de définir une sémantique IF pour les diagrammes UML.

Au niveau des langages formels, il serait intéressant d'explorer l'utilisation de langages tels UPPAAL et Kronos qui prennent en compte le temps continu. Pour les systèmes distribués, il serait aussi nécessaire d'explorer des formalismes basés sur les automates temporisés communicants intégrant des aspects stochastiques [KNS 02]. Enfin, certains techniques comme le calcul symbolique peuvent sembler plus adaptés pour la réduction des modèles et sont à explorer.

Le langage SDL est adapté à la modélisation de la structure et du comportement des systèmes. De son côté, les diagrammes de séquence semblent plus adaptés pour la spécification des systèmes. Dans la suite des travaux de [WT 04], il serait intéressant d'évaluer l'utilisation des diagrammes de séquence dans le cadre du modèle de contraintes. L'utilisation des MSC pose des problèmes de sémantique comportementale, en partie résolus par des formalismes tels les LSC [DH 01]. Il reste donc à préciser la sémantique temps réel des diagrammes et étendre la relation d'équivalence dans ce cadre.

IV Composants pour la gestion de la QdS

Après avoir étudié les langages formels pour décrire et valider des architectures opérationnelles, nous nous intéressons à la structuration de ces architectures pour la prise en compte de contraintes de QdS.

Cette étude est le fruit d'une collaboration avec France Telecom R&D autour de son modèle à composants Fractal [BCS 03] et de son *framework* pour l'embarqué Think [FSL 02].

Le domaine d'application privilégié de cette étude est celui des systèmes portables grand public (téléphone, organiseur, lecteur vidéo, jeu).

IV.1 Introduction

Le modèle économique actuel de développement des systèmes portables grand public requiert un besoin d'ouverture et de flexibilité permettant ainsi leur évolution via l'installation, ou la désinstallation, dynamique d'applications ou de services. Ce besoin d'ouverture, couplé aux contraintes de ce type de systèmes (ressources limitées et spécifiques, besoin de fiabilité), font qu'il est nécessaire de pouvoir disposer d'un support permettant, à l'exécution, la gestion, à la fois, dynamique et sûre des ressources utilisées par le système.

Plus précisément, nous nous intéressons à des systèmes au comportement dynamique. On considère des comportements dynamiques de l'environnement (utilisateur), des applications (arrivée/arrêt de composants) et des ressources (profil variable de QdS).

En exprimant explicitement les services offerts et requis, les composants sont des unités facilement manipulables et surtout composables. La programmation par composants est alors adaptée à la gestion dynamique de code. En particulier, Fractal et son *framework* pour l'embarqué Think permettent de construire des systèmes flexibles et adaptables pour les systèmes embarqués [FSL 02] [LES 05].

Notre objectif est d'étendre ces modèles pour intégrer des politiques de gestion de la QdS liées à l'utilisation de ressources matérielles. On se propose alors de définir une architecture, à base de composants, pour la gestion de la QdS d'un modèle à composants.

Avant de présenter notre contribution, nous résumons les résultats existants obtenus autour de la gestion de la QdS dans les modèles à composants sans contraintes de ressources, des modèles de composants pour l'embarqué et des architectures de gestion de la QdS. Nous proposons ensuite une architecture de gestion de la QdS pour les SETR, à base de composants, appelée Qinna. Enfin nous faisons un bilan avant de fournir des perspectives.

IV.2 Composants embarqués et architectures de QdS

IV.2.1 Etat de l'art

Il n'existe pas, à l'heure actuelle, de définition communément admise du terme composant. Cependant, dans le cadre de ce rapport, nous utilisons la définition largement répandue et proposée par Clemens Szyperski dans [Szy 98]:

« Un composant est une unité de composition avec des interfaces fournies et requises contractuellement spécifiées. Toutes ses dépendances externes sont explicites permettant ainsi son déploiement et sa composition par un tiers »

Plus précisément, nous considérons qu'un composant est une entité logicielle exprimant clairement ses dépendances et les services qu'elle fournit [BCS 03][UML2]. Idéalement, toutes les dépendances de contexte doivent être capturées par les interfaces du composant. Ainsi, un composant peut être composé et déployé dans un système. Une vision largement partagée de la composition se fonde sur la notion de contrats entre deux composants à assembler. On distingue quatre niveaux distincts de contrats, organisés en une hiérarchie suivant leur degré de négociabilité [BJP 99] : la signature (interconnection des composants), l'aspect fonctionnel (généralement sous la forme de pré/post conditions), l'aspect comportemental (protocole d'appel des composants) et enfin l'aspect extra-fonctionnel (relation de QdS).

Du point de vue de la gestion de la QdS, il existe deux classes de modèles : soit la QdS n'est pas prise en compte (OSGi [OSGI 03], EnterpriseJavaBean [EJB03], COM/COM+/DCOM [COM 00], soit seuls quelques aspects le sont (EJB [EJB03], CCM [CCM 02], .NET [NET 04]). Ces aspects sont évidemment les aspects inhérents aux applications distribuées tels que la sécurité, la persistance, les transactions et le nommage. Il est intéressant de noter qu'aucun de ces modèles ne se préoccupe de la gestion de la QdS liée à l'utilisation des ressources, ce qui s'explique par le fait que les applications visées ne s'exécutent pas sur des systèmes fortement contraints.

Pour autant, on peut retenir de ces modèles quelques points intéressant pour la gestion des systèmes ouverts et des contraintes de QdS :

- l'encapsulation des composants au sein de conteneurs permet la mise en œuvre du principe de séparation des préoccupations [Par 72] afin d'augmenter les possibilités de réutilisation des composants. L'utilisation de conteneurs permet d'effectuer un filtrage des appels indépendamment des composants (pré et post traitements) ;
- la présence d'une plate-forme d'exécution mettant en œuvre des mécanismes permettant le déploiement de composants en cours d'exécution est une caractéristique souhaitée pour les systèmes embarqués ouverts. De même, la possibilité de créer dynamiquement les composants et de gérer leurs liaisons tout au long de leur cycle de vie est primordiale pour ces systèmes ;
- enfin, l'existence des composants, aussi bien à la phase de déploiement qu'à la phase d'exécution, facilite l'évolution dynamique des applications (ajout ou retrait de composants à l'exécution).

Il existe de nombreux modèles à composants dans la littérature ayant pour objectif soit la construction de systèmes d'exploitations dédiés (OS-Kit [FBB 97], Coyote [NBH 98], 2K [KCM 00], MMlite [HF 98], Pebble [GSB 99], Think [FSL 02]), soit le développement d'applications pour les systèmes embarqués (PECOS [PEC 99], Koala [OLK 00], VEST [SAH 01], SEESCOA [UVH 01], Rubus [IN 02]). Ces modèles visent les systèmes fortement contraints de contrôle de procédé (PECOS, VEST, SEESCOA, Rubus), les systèmes embarqués grand public (MMlite, Think, OS-Kit, Koala, Pebble) ou les systèmes temps réel dédiés à la gestion des trafics réseaux (Coyote). Ces modèles sont caractérisés par plusieurs aspects :

- afin d'éviter la mise en œuvre de plates-formes d'exécution complexes, grandes consommatrices de ressources, ces modèles ne supportent pas le déploiement de composants à l'exécution ;
- à l'exception de Think, les systèmes, une fois compilés, sont monolithiques. Les composants n'existent qu'à la phase de développement et la structure du système en composants est perdue lors de la compilation ;

- ces modèles, excepté Think, se basent sur l'existence d'un micro-noyau temps réel sous-jacent ;
- la gestion de la QdS est limitée aux problèmes d'ordonnancement temps réel des tâches et à la consommation mémoire ;
- les modèles résolvent les problématiques de QdS lors de la compilation des systèmes. Ainsi, les politiques de gestion de la QdS sont fixées par le modèle et ne peuvent pas être modifiées ou étendues ;
- au travers des modèles existants, nous pouvons remarquer que l'efficacité des composants est un critère primordial : les composants sont implémentés soit en langage C soit directement en assembleur ;

Ces modèles, ainsi que leurs technologies associées, ont été développés pour des classes de systèmes particuliers : il s'agit généralement de décrire, avant compilation, une organisation spécifique de l'application. Le concepteur du système peut ainsi exprimer des contraintes sur l'exécution des composants, mais la manière dont celles-ci sont gérées (par exemple pour l'ordonnancement) est implicite aux outils de compilation fournis. Il est à noter le cas particulier de Think qui propose une approche homogène de développement des systèmes embarqués à l'aide de composants. De plus, Think se distingue des précédents modèles par le fait que la structure en composants du système est réifiée à l'exécution et qu'il ne se repose pas sur un micro-noyau car il est lui-même le résultat d'une composition. Cela permet d'avoir un contrôle total des ressources du système, ce qui est primordial pour la gestion de la QdS. Par contre, Think n'offre pas de solution pour la gestion de la QdS.

Enfin, comme notés dans [ART 05], deux principaux obstacles ont été identifiés pour l'adoption de la technologie composant dans les systèmes embarqués :

- il n'existe pas de modèle standardisé de composants pour ce domaine d'application. Ceci est principalement dû au fait que l'ensemble des industriels concernés ont des priorités différentes concernant les caractéristiques devant être offertes par de tels modèles. Il est donc nécessaire de disposer d'un modèle suffisamment générique pouvant être spécialisé. Cela permettrait de mutualiser les efforts et de disposer d'outils communs ;
- les modèles de composants doivent pouvoir supporter la spécification et la prédiction de contraintes temporelles et de QdS aussi bien statiquement que dynamiquement.

En conséquence, il n'existe pas à ce jour de modèle de composants génériques, prenant en compte la QdS liée à l'utilisation des ressources pour des systèmes ouverts, ce qui est l'objet de notre étude. Avant de présenter notre contribution, nous donnons quelques éléments sur les architectures de gestion dynamique de la QdS.

Les architectures de gestion de la QdS ont pour principal objectif de pouvoir gérer dynamiquement la QdS. Pour cela, elles se basent sur le principe de gestion de contrat de QdS dont trois grands principes ont été identifiés [AC 98]. Il s'agit de la spécification, de l'initialisation et de la gestion dynamique, proprement dite. La spécification permet de décrire les données des contrats en termes de performance, importance et politique. L'initialisation permet d'établir les contrats grâces aux phases de *mapping* (traduction) de QdS entre les niveaux applicatif, service, système d'exploitation et ressources, aux tests d'admission et à la réservation des ressources. Enfin, la gestion dynamique des contrats correspond à l'observation des contrats, leur adaptation, ainsi que leur maintenance.

D'après l'architecture proposée par l'ISO [ISO 95], chaque élément est clairement dissocié dans des entités distinctes (configuration, choix, réservation, action). Cette approche permet de découpler les politiques des mécanismes de gestion de la QdS [LCC 75] afin d'augmenter la flexibilité de l'architecture. L'architecture QoS-A [Cam 96] s'appuie sur une structuration en couches et montre que les trois concepts de gestion de contrat sont suffisants pour une gestion dynamique de la QdS. Enfin, l'architecture QuA [ALE 04] démontre l'intérêt, d'un point de vue de la gestion de la QdS dynamique, de pouvoir identifier les composants à l'exécution du système. Cependant, QuA s'appuie sur un système d'exploitation et sur les services d'une machine virtuelle sous-jacente ce qui rend difficile, voire impossible, le contrôle fin des ressources de la plate-forme.

Les principes des architectures de gestion de la QdS fournissent des pistes intéressantes (séparation des préoccupations, conteneurs de composants, contrats de QdS, gestion des contrats) pour la gestion dynamique de la QdS dans les systèmes embarqués ouverts à composants. Cependant, ces architectures ne sont pas adaptées aux systèmes embarqués. En effet, ces architectures se focalisent principalement sur des systèmes distribués et sur des problématiques de gestion de la QdS liée aux communications. De plus, les architectures existantes pour les applications à composants ne permettent pas de gérer finement les ressources matérielles des plates-formes d'exécution, car elles s'appuient sur des machines virtuelles, masquant ainsi les ressources. De ce fait, les politiques de gestion de la QdS pouvant être fournies par l'architecture sont limitées à celles considérées par la machine virtuelle et par le système d'exploitation.

IV.2.2 Conclusion

L'étude des différents modèles de composants, ainsi que des architectures de gestion de la QdS, ont permis d'identifier les principes clés devant être mis en œuvre par une architecture de gestion de la QdS pour les composants. Selon, les modèles de composants dédiés aux applications distribuées (dynamicité, composants à l'exécution, conteneur), ainsi que des modèles dédiés aux systèmes embarqués (efficacité, gestion des ressources) et des architectures de gestion de la QdS, ces principes sont :

- la séparation des préoccupations fonctionnelles et de QdS, la séparation des politiques et des mécanismes de gestion de la QdS ;
- l'utilisation des contrats de QdS et des activités liées à leur gestion dynamique. Ces activités, identifiées dans les architectures de gestion de la QdS, portent sur la spécification, l'initialisation et la gestion dynamique des contrats ;
- l'utilisation du principe de conteneur de composants permettant de définir un point d'entrée unique pour la gestion de la QdS ;
- concevoir un système comme une composition homogène de composants (du niveau ressources au niveau applicatif) et ainsi permettre leur identification pendant l'exécution afin de pouvoir les gérer dynamiquement.

Nous ajoutons à ces principes, la maximisation des niveaux de QdS utilisés par les composants, afin d'utiliser au mieux les ressources matérielles de la plate-forme. L'étude bibliographique montre qu'il y a un manque au niveau de la gestion de la QdS dans les systèmes embarqués à composants. Ce constat, dont font également part les membres du projet Artist [ART 05], nous a amené à proposer une nouvelle architecture de gestion de la QdS, appelée Qinna, qui reprend les principes énoncés précédemment.

IV.3 Notre contribution

IV.3.1 Introduction

Afin de respecter les principes énoncés précédemment, une architecture de gestion de la QdS pour les systèmes à composants doit s'appuyer sur un modèle de composants suffisamment générique et flexible. Pour ces raisons, l'architecture proposée, appelée Qinna, se base sur la définition de composants conforme au modèle Fractal et utilise les propriétés du *framework* à composants Think. En effet, ce dernier permet l'identification des composants en cours d'exécution du système. De plus, Think répond au besoin d'efficacité en étant implémenté par des langages de bas niveau (C et assembleur). Enfin la démarche Factal/Think a démontré son efficacité pour produire des systèmes d'exploitation dédiés : Fractal définit une architecture alors que Think l'implémente afin de servir de *framework* pour construire des systèmes embarqués dédiés[15]. Qinna se situe au niveau de Fractal, les implémentations de Qinna au niveau de Think, en réutilisant les composants Think pour les aspects fonctionnels.

IV.3.2 L'architecture Qinna

L'architecture Qinna est définie au travers d'un ensemble de types de composants (QoSComponent, QoSComponentBroker, QoSComponentManager, QoSDomain et QoSComponentObserver) (cf. figure 7), de types abstraits de données, d'interfaces et de comportements dynamiques.

Structure

Chacun des types de composants est en charge d'une des activités de gestion des contrats de QdS. Cette approche permet de séparer, entre les différents composants, les politiques et les mécanismes de gestion de la QdS, ainsi que les préoccupations fonctionnelles et de QdS du système.

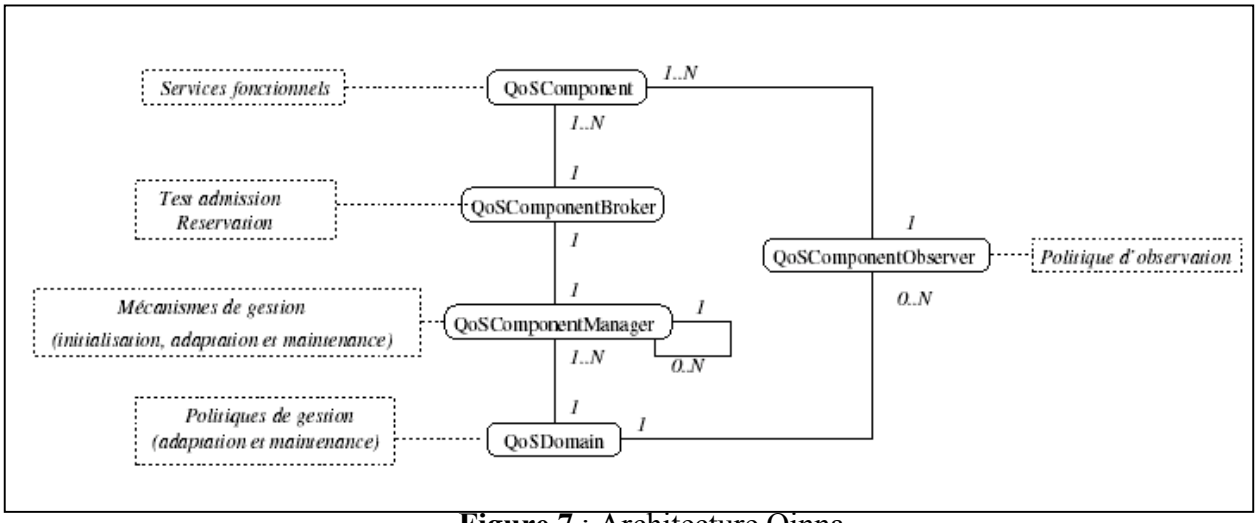

Figure 7 : Architecture Qinna

[15] l'implémentation de Think, Kortex, fournit une librairie de composants, libre de droit, pour construire son propre exécutif http://think.objectweb.org/

Avec Qinna, chaque composant, appelé Component, nécessitant une gestion de la QdS sur ses interfaces devient un QoSComponent. Pour établir un contrat sur un QoSComponent, il faut avoir une vue de la QdS requise par ce composant. C'est le rôle du QoSComponentManager. Ce dernier encapsule une vue orientée QdS du QoSComponent. Il implémente en particulier l'opération de *mapping* (lien entre la QdS offerte par le QoSComponent et la QdS requise sur les services requis). A partir de ces informations, pour établir un contrat, il faut évaluer les quantités de QdS disponibles par QoSComponent requis. C'est le rôle des QoSComponentBroker. Un QoSComponentBroker réalise le test d'admission ainsi que la réservation pour un QoSComponent requis. Il existe en fait un QoSComponentBroker par classe de QoSComponent. Le test d'admission est donc distinct du composant réalisant l'accès proprement, soit le QoSComponent. Les appels à ces mécanismes de réservation sont réalisés suivant les politiques d'adaptation et de maintenance implémentées par le QoSDomain. Au niveau du suivi des contrats, vu qu'il fournit une certaine quantité de QdS, le QoSComponent a en charge la surveillance de la quantité de QdS fournie en regard de celle contractualisée. Enfin, le QoSDomain définit les politiques d'observation de contrats de QdS implémentées par les QoSComponentObservers.

Les QoSComponents implémentent des préoccupations fonctionnelles et de QdS du système, alors que les autres composants de l'architecture n'implémentent que des préoccupations de QdS. Au sein de la préoccupation de QdS, le principe de séparation des politiques et des mécanismes est respecté : le QoSDomain implémente les politiques d'adaptation et de maintenance, tandis que les mécanismes correspondants sont implémentés par les QoSComponentManager.

Données

Afin d'être générique, les types de données considérés sont des types abstraits. Ils s'appuient sur des opérateurs abstraits de comparaison et de composition de QdS. Pour répondre aux problèmes des systèmes ouverts, une architecture doit pouvoir considérer des composants possédant des contraintes de QdS requises hétérogènes. Le choix de Qinna est de définir un opérateur *translate* permettant de traduire des QdS requises hétérogènes en une QdS standard comprise par le système. Dans le cadre des systèmes ouverts, la définition de ces règles de traduction peuvent être difficiles à réaliser et l'hétérogénéité est prise en compte dans la limite prévue par la concepteur. Lorsque l'hétérogénéité n'est pas prévue, Qinna impose une valeur par défaut de la QdS notée *default*. Dans ce cas, le système doit permettre l'auto-configurabilité des caractéristiques de QdS du composant, soit pouvoir déterminer dynamiquement les niveaux de QdS requis. Cette opération est aussi à réaliser lorsqu'un composant a été « calibré » sur un autre système et que les informations de QdS ne sont pas fiables. A cet effet, Qinna propose d'attribuer le paramètre *unreliable* à la table de *mapping*. Dans ces deux cas (*default*, *unreliable*) l'architecture propose la mise en place d'observateurs et les politiques de maintenabilité permettent de faire évoluer les niveaux de QdS requis d'un composant.

Comportement

Dans Qinna, une opération de QdS correspond à la mise en place d'un contrat, à l'adaptation d'un contrat ou à la mise à jour dynamique des données liées à un contrat. Afin de préserver la cohérence du système, chaque opération de gestion de la QdS est atomique et plus prioritaire que les traitements fonctionnels. Enfin, chaque opération est modélisée via un diagramme de séquence représentant l'ensemble des interactions entre les composants pour rendre le service attendu.

La vision décentralisée des contrats de QdS entre les différents composants du système implique la collaboration de l'ensemble de ces composants lors des opérations de gestion de la QdS. L'ensemble des composants gérant les contrats de QdS forment alors un arbre, implicite, qu'il convient de parcourir, soit exhaustivement, soit partiellement, lorsqu'une opération de gestion de la QdS est nécessaire. Cet arbre implicite modélise les liens entre l'ensemble des QoSComponentManager. La gestion dynamique de cet arbre est activée à chaque changement de l'état du système vis-à-vis de la QdS :

- évolution de la QdS fournie par un composant (généralement une ressource matérielle) ;
- évolution de la QdS requise par un composant (généralement un changement de mode dans une application) ;
- modification de la configuration du système (arrivée ou arrêt d'un composant).

Le parcours de l'arbre a pour objectif de trouver un contrat acceptable, soit un ensemble de sous-contrats acceptables, quitte à dégrader des contrats en cours. La dégradation des contrats dépend des niveaux d'importance et des politiques d'adaptation définis au niveau du QoSDomain. A l'inverse, lors de l'arrêt d'un composant ou d'une augmentation de QdS offerte, l'architecture est en charge de maximiser les contrats en cours.

Une fois les principes généraux de Qinna définis, nous proposons des principes de mise en œuvre.

IV.3.3 Mise en œuvre de Qinna

L'intégration de Qinna passe par une structuration particulière des composants du système. On distingue les composants dits « sécables » des composants dits « insécables ». Un composant insécable ne possède qu'une seule instance à l'exécution. C'est le cas, par exemple, du CPU ou du réseau. A l'inverse, un composant sécable peut être décomposé, à l'exécution, en sous-composants du même type. Par exemple, la mémoire peut être vue soit comme un composant insécable (une seule zone mémoire) soit comme un composant sécable (partitionnement de la mémoire en blocs). Pour les composants insécables, Qinna impose l'ajout d'un composant de contrôle d'accès (classiquement appelé ordonnanceur). C'est ce composant qui réalise et contrôle l'accès proprement dit au composant sécable. De ce fait, il est le QoSComponent et est associé au QoSComponentBroker pour le test d'admission. Dans le cas des composants sécables, on propose une division du composant en un ensemble de sous-composants. Par client, on crée un sous-composant qui devient alors un QoSComponent. Cette division nécessite une adaptation des composants existants, mais permet une gestion fine et sûre de la QdS. Par exemple, pour chaque composant utilisant de la mémoire, il existe un QoSMemoryComponent gérant la zone mémoire allouée.

Dans le cadre d'une organisation plus complexe, mélangeant diverses politiques d'ordonnancement [GGV 96] et de partage de composants, nous proposons une structuration hiérarchique des composants. Le contrôle d'accès est alors effectué à plusieurs niveaux créant ainsi une hiérarchie de QoSComponent. Par exemple, les *threads* d'une machine virtuelle JAVA accède, via un ordonnanceur à partage de temps, à une tâche d'un RTOS. L'ordonnanceur d'accès à cette tâche est un QoSTask. Les tâches du RTOS sont elle même ordonnancées par priorité pour accéder au CPU, via un QoSCPU.

Au niveau de la structuration globale de l'application, nous proposons une structuration classique du système selon quatre couches : une couche ressource (CPU, batterie, mémoire, réseau), une couche exécutif (services d'utilisation de la plate-forme : *thread*, sémaphore, …), une couche service (fonctions applicatives réutilisables dans divers contextes comme le décodage vidéo) et une couche application. La QdS est discrétisée et correspond soit à une quantité de ressources (%CPU ou réseau, ko de mémoire), soit à un nombre maximal d'instances pour les services (nombre maximal de *thread* d'un OS, nombre maximal de fenêtres actives simultanément), soit enfin à une QdS utilisateur pour les applications (e.g. n images par seconde pour un lecteur vidéo).

Il semble difficile, voire impossible, de valider formellement l'intérêt des principes mis en place par une architecture, ainsi que l'architecture elle-même. En revanche, afin d'en percevoir les apports et les limites, il est intéressant de l'illustrer, de l'expérimenter et de l'évaluer. Nous présentons maintenant rapidement les éléments marquants de ces expérimentations, effectuées dans le cadre de la thèse de Jean-Charles Tournier, qui nous ont amené à définir des principes de mise en œuvre de l'architecture.

IV.3.4 Expérimentations de Qinna

Afin de s'assurer que les niveaux de QdS utilisés sont conformes aux niveaux contractualisés, Qinna, impose de contrôler dynamiquement la consommation de QdS à partir des niveaux de QdS contractualisés. Les tests, réalisés par les QoSComponents, peuvent être difficiles à mettre en œuvre suivant le service rendu. Dans les cas des composants ressources (CPU, mémoire, réseau) ces tests sont plus classiques et faciles à implémenter. C'est pourquoi, en pratique, l'architecture se base principalement sur la confiance des contrats sur les composants réifiant les ressources.

L'activation d'un contrat peut être coûteuse du fait de l'établissement de l'ensemble des sous-contrats (un par QoSComponent concerné). La minimisation du coût d'établissement d'un contrat revient à un problème d'optimisation de parcours de l'arbre de dépendance des QoSComponentManager. A cet effet, nous proposons trois stratégies :

- la première stratégie consiste à privilégier l'exploration des nœuds ayant par le passé mis en échec l'établissement d'un contrat ;
- pour la seconde stratégie, le QoSDomain peut effectuer un pré-parcours du graphe afin de disposer directement de toutes les informations nécessaires. En contre-partie l'architecture perd en flexibilité, car cela revient à n'avoir qu'un seul composant gérant toute la QdS de tous les composants ;
- la troisième stratégie consiste à effectuer une analyse a priori du système et de remplacer les tests d'admission par des résultats pré-calculés. Cette approche réduit considérablement l'utilité de Qinna, et n'est pas possible dans un contexte fortement hétérogène et dynamique.

La maîtrise du coût d'établissement d'un contrat pose, en fait, le problème du découpage d'un contrat global en sous-contrats, soit le problème plus général de la granularité des composants. Plus un contrat est décomposé en sous-contrats, plus son coût d'établissement est élevé (parcours d'établissement du contrat, adaptation) mais plus sa gestion de la QdS est fine. Une solution permettant de limiter le coût est de définir les QoScomponent comme un regroupement de composants par application ou service applicatif, afin d'obtenir un composant à plus gros grain. Une technique proposée est alors

de limiter la mise en place d'un QoSComponent aux composants possédant des exigences de dynamicité vis-à-vis du système : arrêt, relance ou modification possible du composant au cours de la vie du système.

Le niveau de QdS requis d'un composant n'est pas forcément constant ou prédictible et peut évoluer en cours d'exécution du composant. Dans ce cadre, nous avons évalué Qinna pour un profil générique correspondant à un pic de demande de QdS. Qinna peut traiter un pic de demande de deux façons différentes: soit de façon statique (le niveau de QdS maximal ou réel est contractualisé), soit de façon dynamique (le niveau minimal est contractualisé puis adapté).

La première approche consistant à contractualiser le niveau maximal de QdS est intéressante si l'écart entre les niveaux de QdS maximal et minimal est faible, ou si la durée du niveau de QdS requis au minimum est courte. En effet, cette approche permet d'éviter les adaptations de contrat liées aux politiques de maintenabilité. En contre-partie, elle peut entraîner un gaspillage de QdS. La seconde approche (contrat au niveau réel) est celle permettant d'éviter tout gaspillage de QdS et ne nécessite aucune opération sur le contrat. Cependant, elle nécessite une connaissance fine des profils des niveaux de QdS requis et la mise en œuvre de tests d'admission complexes ce qui est difficile, voire souvent impossible, à déterminer dans un contexte applicatif donné.

La dernière approche est la seule approche possible lorsque les profils de QdS requis sont inconnus. Cette approche s'appuie sur les composants d'observation QoSComponentObserver proposés par Qinna. Deux stratégies de mise en place des observateurs ont été identifiées. La stratégie, dite « à scrutation » (on observe périodiquement l'état du OoSComponent), permet de suivre l'augmentation et la diminution des niveaux de QdS requis, mais induit des retards (dues à la période de scrutation). A l'inverse, la stratégie dite « événementielle » (un événement est généré à chaque violation de contrat), permet de suivre, sans retard, uniquement l'augmentation des niveaux de QdS requis. La stratégie « à scrutation » convient à un profil variable (par exemple l'utilisation variable du réseau), tandis que la stratégie « événementielle » permet de déterminer un niveau de QdS requis stable en partant d'une valeur minimale (par exemple une demande croissante de mémoire).

Les coûts de cette prise en compte s'évaluent en retard et en gaspillage de QdS et dépendent des paramètres d'observation (période d'observation) et des paramètres de maintenabilité.Le *pas* de maintenabilité définit la variation de QdS d'un contrat lors de sa réévaluation. Il est à noter que le QoSComponentObserver à scrutation impacte le comportement temporel des composants en créant des retards (attente d'une période pour réagir), ce qui modifie les propriétés de QdS temporelles du système (l'observation modifie le comportement du système). Ceci peut poser un problème pour des systèmes à contraintes temps réel ou le retard est un élément fort de la correction du système.

Dans le même ordre d'idée que pour le suivi de variation d'une QdS requise, la prise en compte de l'évolution de la QdS fournie par un composant dépend du type de composant concerné. Trois types de composants ont été identifiés : les composants ayant des capacités discrétisées (par exemple un processeur à pallier de vitesse en fonction du niveau d'énergie), les composants dont la capacité varie lentement dans un sens constant (par exemple la batterie) et les composants dont la capacité subit des variations temporaires (par exemple la bande passante d'un réseau). Afin de réagir à l'évolution des capacités de

ces trois types de composants, nous avons identifié trois stratégies. Dans le cas d'une ressource aux capacités discrètes, il convient de modifier la contrainte globale à chaque changement de capacité. Dans le cas d'une capacité à variation continue, les valeurs de la contrainte globale sont discrétisées afin de ne déclencher les opérations d'adaptation que lorsque la capacité atteint un certain seuil. Enfin, dans le cas d'une variation temporaire de la capacité, il est nécessaire d'adjoindre des propriétés temporelles aux valeurs discrétisées de la contrainte globale afin de ne déclencher les opérations d'adaptation que lorsque la variation dépasse un certain seuil depuis un certain temps.

Les expérimentations ont permis de constater que la séparation des activités de gestion de la QdS permet de disposer d'une architecture réutilisable. Par exemple, dans le cas des composants de services ou applicatifs les tests d'admissions sont souvent les mêmes (vérification du nombre d'instances en cours d'exécution). Il suffit donc d'instancier un composant type générique. A l'inverse, les tests d'admission liés aux ressources sont très différents et peuvent être complexes dans le cas, par exemple, de tests d'admissions pour les ressources insécables (CPU, réseau). Dans ce cas, ils peuvent être « préprogrammés » dans des composants et intégrés à l'architecture.

En pratique, un composant pouvant être réutilisé doit prévoir de s'exécuter dans des contextes différents et donc prévoir l'hétérogénéité des contraintes de QdS. De même, lorsqu'un composant est réutilisé dans des contextes différents, ses niveaux de QdS requis doivent être réévalués. Cela signifie que pour que la réutilisabilité puisse être effectuée à moindre coût, elle doit se faire entre des systèmes proposant des caractéristiques matérielles et des contextes logiciels relativement similaires.

Enfin, le coût de l'architecture doit être évalué pour assurer la faisabilité de l'approche et le dimensionnement du système. Ce coût est contextuel, il dépend des applications et de l'implémentation de Qinna (granularité des composants, nombre de niveaux de QdS de chaque composant, politiques d'observations et *pas* de maintenabilité). Nous illustrons le coût de Qinna au travers d'une application de démonstration développée au sein de France Telecom R&D. Cette expérimentation s'appuie sur deux applications concurrentes grandes consommatrices de ressources, soient un jeu multimédia et un lecteur vidéo. Au final, le système est composé de 7 QoSComponent et a été déployé sur un iPaq™ H3800 (processeur strongArm™ à 200Mhz, 60Mo de DRAM, 20Mo de Flash). Les performances obtenues sont les suivantes : 4,36 ms pour établir un contrat, 1,08ms pour l'adapter et 749ko de mémoire utilisé par Qinna. Le surcoût en mémoire est de 1,5%. Le résultat de 1,08ms est à rapprocher de la fréquence de rafraîchissement des images, 50ms. Plus généralement, les expérimentations ont montré que le coût restait raisonnable dans le contexte des systèmes portables grand public.

IV.3.5 Conclusion

Les études montrent que les concepts présents dans l'architecture sont suffisants pour une gestion dynamique de la QdS, mais que leur mise en œuvre revient à effectuer un compromis entre le gaspillage de QdS et le nombre d'opérations à réaliser. En particulier, le coût de gestion de la QdS dépend fortement des niveaux de QdS contractualisés. La contractualisation peut conduire à un gaspillage de QdS, lorsque le contrat sur-estime, respectivement sous-estime, les niveaux de QdS réellement requis, respectivement offerts. A l'inverse, le nombre d'opérations de mises à jour des données des contrats est important

lorsque le contrat sous-estime, respectivement sur-estime, les niveaux de QdS réellement requis, respectivement offerts.

L'identification de ces réponses permet de justifier a posteriori le choix d'une architecture à base de composants et l'utilisation de contrats pour la gestion de la QdS. En effet, les composants sont à la base de la réponse apportée à la problématique de réutilisation. De plus, la mise en œuvre de l'architecture à base de composants permet de quantifier précisément le coût de chaque opération de gestion de la QdS dans une optique d'optimisation de l'architecture. Enfin, les composants permettent la mise en œuvre de politiques de gestion dynamique de la QdS.

IV.4 Bilan et perspectives

IV.4.1 Bilan

L'architecture Qinna permet d'apporter des réponses aux problématiques de gestion dynamique de QdS dans les systèmes embarqués ouverts [TOB-2 5]. Qinna identifie principalement deux domaines d'applications privilégiés que sont les systèmes multimédia et les systèmes temps réel. Dans le cas des systèmes temps réel l'apport se situe au niveau de la structuration des éléments de gestion de la QdS [TOB 04]. Qinna permet de structurer clairement les différents éléments de QdS en identifiant, en particulier, les éléments à mettre en œuvre pour la gestion de l'hétérogénéité (temps réel dur et souple, tâches périodiques et apériodiques [BS 96]) et pour la gestion des modes dégradés des applications temps réel (par exemple, lors de la mise en œuvre du régisseur temps réel [Del 94]). Dans le cas des systèmes multimédia, Qinna apporte en plus les bénéfices liés à la gestion dynamique de la QdS [TOB-1 05],

Bien que Qinna possède les concepts nécessaires à une gestion dynamique de la QdS, leur mise en œuvre conduit à un compromis, effectué par le concepteur du système, entre un gaspillage de QdS (discrétisation du niveau de QdS contractualisé) et un nombre important d'opérations d'adaptation à réaliser par l'architecture (suivi de la variabilité des profils de QdS requis et des capacités des ressources matérielles, niveau de QdS contractualisé proche du niveau réel utilisé).

Les contrats considérés identifient une enveloppe discrétisée de QdS réellement nécessaire ou réellement fournie. Cette approche est à rapprocher des systèmes temps réel où une tâche est caractérisée par sa pire durée d'exécution (constituant une enveloppe sûre) et non sur sa durée réelle (souvent difficile à caractériser). Dans le cas de Qinna, le choix de l'enveloppe revient à effectuer un compromis entre le gaspillage de QdS et le nombre d'opérations d'adaptation à effectuer. La mise en place de contrats est donc une problématique en soi, mais doit aboutir à une meilleure maîtrise de la gestion de la QdS. Enfin, les contrats, tels que définis par Qinna, permettent d'expliciter des liens, et des choix, nécessairement existants entre les offres et les besoins de QdS des différents composants du système.

Enfin, Qinna apporte des éléments de réponse à la problématique de confiance sur l'utilisation des composants, en se basant sur des attributs de fiabilité des tables de *mapping* et des tests de QdS consommés, tests réalisés par les QoSComponents. En

pratique, la confiance de l'architecture se situe essentiellement au niveau des ressources matérielles de la plate-forme.

IV.4.2 Perspectives

Les résultats obtenus permettent d'entrevoir des perspectives de recherche que nous exposons maintenant. Elles concernent les principes de Qinna, son implémentation, sa formalisation et l'extension de Qinna à d'autres contextes.

Au niveau des principes, afin d'augmenter les possibilités de réutilisation apportées par Qinna, la séparation des préoccupations au sein des QoSComponents pourrait être encore plus forte en séparant l'aspect purement fonctionnel et de configuration du composant pour un niveau de QdS donné, tel qu'identifié dans les architectures à méta-niveaux.

L'architecture proposée ne traite actuellement que les composants fournissant une seule interface nécessitant une gestion de la QdS. Il est donc nécessaire d'étendre l'architecture afin de pouvoir gérer plusieurs interfaces fournies avec contraintes de QdS. De même, la définition des types de composants implique la mise en œuvre d'une politique de gestion de la QdS par classe de QoSComponent : il n'est donc pas possible que deux QoSComponents d'une même classe soient gérés de manière différente. Dans cette optique, il serait nécessaire de définir plusieurs tables de *mapping*, chacune étant dédiée à une interface. Les QoSComponentManagers devraient alors gérer l'ensemble des tables de *mapping* des interfaces fournies d'un même composant et devraient, en particulier, gérer les aspects d'inter-dépendances entre les différentes tables.

La seule auto-configuration prévue par Qinna est l'évaluation des niveaux de QdS requis des composants. Dans le cadre des systèmes ouverts, il faudrait prévoir les mécanismes permettant l'accueil d'un composant totalement inconnu, de type « boîte noire ». Une piste possible est de lui associer, à son arrivée, des composants de gestion de la QdS totalement générique et de les spécialiser par apprentissage.

Dans le cadre des expérimentations réalisées, on s'est appuyé sur l'utilisation de composants programmés dans les langages assembleur ou C. L'implémentation de Qinna pour un *framework* tel Julia[16] permettrait détendre l'approche aux machines virtuelles. Ceci pose alors le problème de la mise en place de Qinna dans le contexte d'une machine virtuelle et donc d'un contrôle limité des ressources de la plate-forme. Une solution à investiger serait alors de « fractaliser » la machine virtuelle et d'y intégrer Qinna.

Toujours au niveau de l'implémentation, une piste de recherche porte sur l'optimisation de la gestion dynamique des contrats. Cette optimisation passe par un regroupement pertinent de données (méta-données), afin de regrouper les informations nécessaires à l'établissement d'un contrat et de ses sous-contrats et sur la formalisation des politiques de parcours de l'arbre des sous-contrats.

Pour formaliser l'implémentation, il serait utile de travailler à fournir des règles qui, à partir des caractéristiques du système telles que la granularité des composants, le nombre de niveaux de QdS par composant, les profils de QdS requis des composants ou encore les

[16] Julia est le *framework* du modèle Fractal pour des composants du monde Java. Julia est accessible sur fractal.objectweb.org/tutorials/fractal/

profils des capacités des ressources, permettraient de guider le concepteur dans les nombreux choix à réaliser lors de l'implémentation de l'architecture (mise en place des contrats, des *pas* de maintenance, des politiques d'observation ou des stratégies de suivi de la capacité d'une ressource).

La définition d'un langage de spécification de QdS dédié aux composants permettrait de générer automatiquement les composants Qinna correspondants. Dans cette perspective, le langage CQML « *Component Quality Modelling Language* » [Aag 01] apparaît comme une solution qu'il est nécessaire d'évaluer. CQML permet de spécifier précisément la QdS et les politiques de gestion associées.

Au sujet de la formalisation de Qinna, il serait nécessaire de travailler à la définition d'un méta-modèle de l'architecture. Les intérêts d'une telle définition sont doubles. Premièrement, cela permettrait de vérifier formellement des propriétés de Qinna. Ensuite, cela faciliterait le développement d'outils de génération automatique de l'implémentation de l'architecture suivant un contexte donné. Il serait alors intéressant de développer des outils de transformations de modèles pour intégrer l'architecture Qinna à une architecture donnée, en intégrant en particulier les résultats des travaux portant sur l'implémentation.

L'architecture Qinna se base sur les propriétés du modèle à composants Fractal et de *framework* Think, il serait intéressant d'étudier si Qinna peut être appliquée à d'autres modèles. Cette perspective est actuellement en cours d'étude dans le cadre du projet RNRT PISE[17] visant à intégrer des mécanismes de gestion de la QdS pour les plates-formes OSGi[18] « *Open Service Gateway Initiative* ». OSGi fournit la spécification d'une plate-forme de déploiement de services pour un modèle à composants. Il permet d'adresser les problématiques de gestion du cycle de vie des composants, aspect non traité par Qinna,

Qinna a été établi pour des systèmes monoprocesseurs, une perspective est donc de l'étendre au contexte réparti et distribué. Deux approches peuvent alors être explorées. La première consiste à mettre en place un QoSDomain ayant une vision globale du système permettant ainsi de ne pas modifier l'architecture, approche adaptée aux systèmes multiprocesseurs. La deuxième approche, adaptée aux systèmes largement distribués, est de définir un QoSDomain par machine et de faire collaborer l'ensemble des QoSDomain afin de répondre à des objectifs globaux de QdS.

Enfin, la QdS traitée n'est pas l'unique propriété extra-fonctionnelle devant être gérée dans les systèmes embarqués. Nous pouvons citer, par exemple, les propriétés liées à la sécurité ou à la tolérance aux fautes. Il serait utile de pouvoir disposer d'une architecture permettant la gestion de l'ensemble de ces propriétés (QdS, sécurité et tolérance aux fautes). Il sera alors nécessaire de se pencher sur le problème de la composition des architectures, qui revient dans ce cas au problème de la composition de composants et de propriétés de QdS.

[17] PISE : Passerelle Internet Sécurisée et flexiblE. 2005.
[18] OSGI est un consortium d'industriels créé en Mars 1999, www.osgi.org/

V Perspectives

Après un bilan des études menées, nous proposons de nouvelles pistes de recherche autour des architectures opérationnelles et du déploiement. C'est en effet l'opération qui aboutit à la construction de l'architecture opérationnelle.

Nous proposons ensuite un projet de recherche sur le développement de SETR. Au vu de l'étendue des problématiques de recherche soulevées, nous nous limitons à un champ applicatif que sont les réseaux de capteurs.

V.1 Perspectives sur les architectures

Les travaux menés ont eut pour objectif de formaliser et de structurer des architectures opérationnelles pour des SETR au comportement dynamique. Nos études se sont concentrées sur la formalisation des architectures multitâches, la prise en compte de la notion d'occurrences dans les flots d'événement et l'organisation du code pour la gestion dynamique de la QdS. Les études proposent des boîtes à outils orientées composants pour construire des architectures opérationnelles, des principes d'implémentation de ces architectures et des méthodologies de mise en œuvre des techniques formelles par simulation exhaustive. Nous avons retenu de ces études le besoin de généricité et d'abstraction des architectures (*bloc* type SDL et composants génériques), la séparation des préoccupations dans la description des architectures et la séparation des modèles des contraintes, de l'architecture opérationnelle et de l'implémentation. Au niveau sémantique, il faut veiller à établir un lien formel entre tous les niveaux du développement, proposer des méthodologies de mise en œuvre des techniques formelles et prendre en compte la sémantique des applications pour réduire les phénomènes d'explosion combinatoire.

En prolongement des études menées, on doit pouvoir développer des outils implémentant les principes exposés. Un premier outil envisageable est un outil d'aide à la mise au point d'architectures multitâches qui offre des possibilités de vérification de propriétés temps réel et de validation des propriétés comportementales. On pourrait par exemple étendre ou reprendre les principes de l'outil LACATRE qui offre déjà des possibilités d'édition graphique et de génération de code. De même, on pourrait envisager le développement d'un environnement de conception de système embarqué intégrant des politiques de gestion de la QdS. Il faudrait d'abord développer une bibliothèque de composants générique de gestion de la QdS (*brokers*, *manager*, *observer* type) et de composants standard préprogrammés (ordonnanceur à priorité de CPU, contrôleur d'accès à la mémoire). Ensuite, on doit aider le concepteur à lier les politiques de gestion de QdS à des composants fonctionnels et à assembler les composants pour construire un système.

Au final, les résultats obtenus doivent permettre d'aider à construire des bibliothèques de composants logiciels réutilisables et validés pour certaines classes de problème.

Les études se sont focalisées sur les aspects logiciels pour des systèmes monoprocesseurs utilisant un RTOS. De manière plus générale, la définition d'architectures pour les SETR pose des problèmes dont les plus intéressants et complexes à aborder sont de notre point de vue :

- construire des systèmes par assemblage de composants hétérogènes et donc gérer l'hétérogénéité des paradigmes ;
- améliorer la mise en œuvre des techniques formelles pour assurer la composition des systèmes ;
- considérer des aspects stochastiques ;
- définir et formaliser des principes d'implémentation en tenant compte des diverses contraintes de QdS ;
- étendre les études aux systèmes distribués en intégrant un facteur d'échelle.

Dans nos travaux, nous avons géré une hétérogénéité de paradigme entre une spécification synchrone et une implémentation asynchrone. Les études récentes sur les approches GALS montrent que loin de s'opposer, les approches asynchrones et synchrones doivent se compléter. L'approche synchrone permet d'aboutir à un code efficace et certifiable. L'approche asynchrone apporte des solutions pour la gestion de la dynamicité en assurant la distribution des comportements. Les questions alors à résoudre sont « comment passer de l'asynchrone au synchrone et inversément ? » et « comment associer asynchrone et synchrone ? », soit comment faire cohabiter les deux paradigmes. De même, le lien avec un procédé physique pose la question de la cohabitation des systèmes discrets et continus. Enfin, l'architecture d'un SETR se doit d'intégrer des aspects logiciels et matériels. Il faut donc dans un développement pouvoir coupler des langages de description d'architectures matérielles et logicielles au sémantique différentes. Plus généralement, pour pouvoir tirer avantage des divers langages et paradigmes, il faut développer des outils et techniques pour assurer l'interopérabilité de modèles aux sémantiques diverses.

Au sujet de la formalisation, les pistes proposées ont été de valider et de vérifier la correction des architectures par simulation exhaustive. Afin d'améliorer le processus de développement, la formalisation devrait aboutir à la mise en place d'une théorie, soit une expression logique simple, paramétrée, caractérisant les caractéristiques de QdS d'un système. Une théorie permet alors, de vérifier simplement un composant et surtout de connaître les paramètres qui influencent ses caractéristiques de QdS. Il est à noter que les caractéristiques de QdS dépendent de l'environnement du composant et de son implémentation. La modélisation exhaustive peut être, comme suggérée dans [God 04], une piste à suivre pour extraire une théorie par réduction successive et interprétation des modèles. La mise en place de théories ou du moins l'utilisation de modèles fortement réduits devrait permettre d'aider à la maîtrise de la composition de composants en évitant l'explosion combinatoire due à la composition. De plus, afin de régler les problèmes d'explosion combinatoire, il semble inévitable de s'appuyer sur la sémantique des applications pour faire une modélisation au plus juste des systèmes. A cet effet, il serait intéressant de formaliser, et ainsi réutiliser, des règles de réduction de modèles pour des schémas de programme bien identifiés.

Au niveau des hypothèses de travail, nous avons considéré des systèmes au comportement dynamique mais pour autant prédictible. Pour des domaines moins prédictibles, il faudrait investiguer des modèles probabilistes comme les automates temporisés stochastiques [KNS 02]. La difficulté est alors de veiller à utiliser la technique formelle la plus judicieuse. En effet, la technique de modélisation ne doit pas poser de problèmes techniques supplémentaires, en particulier liés à l'explosion combinatoire, de ceux issues de l'application.

Les travaux menés ont posé le problème de la nécessaire optimisation de l'architecture opérationnelle lors de l'implémentation, soit un compromis à réaliser entre conserver des éléments fortement découplés à l'exécution (gestion dynamique de ces éléments) et assurer l'efficacité des opérations en temps et en espace mémoire utilisé. Ces éléments sont les tâches pour la gestion dynamique de la concurrence et les composants pour la gestion dynamique de modules. Une idée à explorer est la mise en place de méta-données et d'intergiciels sous-jacent pour rendre des services de haut niveau, tout en assurant l'efficacité du code produit. Il faudrait alors effectuer un compromis entre un code obtenu par synthèse et l'utilisation de services d'exécution configurables. Ceci pose plus

largement le problème de la réutilisation et de la configuration de composants soit l'intégration de composants avec contraintes de QdS.

Toutes ces questions autour de la modélisation, de la formalisation et de l'implémentation pose la question plus générale du découpage d'un système en sous-systèmes ou composants hétérogènes (logiciel et matériel ; application, services, intergiciels, OS et ressources, fonctionnel et QdS) et comment assurer l'assemblage de ces composants (interopérabilité des modèles, des sémantiques et des plates-formes) en intégrant les contraintes de QdS. C'est dans ce contexte que sont apparues récemment des propositions autour de l'Ingénierie Dirigé par les Modèles (IDM) [GFB 05]. L'IDM propose la mise en place de méta-modèles avec pour objectif de maîtriser la sémantique des modèles, la correction des modèles et la cohérence des nombreux modèles manipulés lors du développement des SETR [GT 05]. L'IDM apparaît aujourd'hui comme l'outil idéal pour formaliser les architectures et les opérations associées (formalisation et implémentation).

Au delà des problématiques liées à la description, à la formalisation et à l'implémentation des architectures opérationnelles, se pose le problème de leur mise en place, soit le déploiement d'une architecture applicative sur une architecture matérielle. Dans ce contexte, le développement des SETR est, aujourd'hui, confronté à de nouveaux défis.

V.2 Perspectives sur le déploiement

V.2.1 Problématiques

On peut citer la liste non exhaustive suivante de défis actuels qui peuvent donner lieu à des développements dans les systèmes d'information mais qui restent des questions ouvertes pour les SETR :

- l'intégration de composants logiciels sur un support d'exécution donné, par exemple l'intégration par un constructeur automobile de composants fournis par un équipementier;
- le déploiement de composants sur des plates-formes hétérogènes, par exemple le déploiement de services sur un système mobile ;
- assurer l'évolution des systèmes pour la mise à jour des applications ;
- intégrer « à chaud », soit à l'exécution, de nouveaux composants ;
- assurer le passage à l'échelle des systèmes.

Ces questions posent le problème du déploiement d'une architecture applicative sur une architecture matérielle donnée en assurant l'indépendance des modèles.

L'IDM et en particulier les travaux récents de l'OMG sur le MDA, en exprimant clairement la sémantique des modèles et des transformations associées, devraient aider à répondre au défi du déploiement. En particulier, une séparation claire est effectuée entre les modèles indépendants de la plate-forme nommés PIM « *Platform Independent Model* » et ceux liés à la plate-forme nommés PSM « *Platform Specific Model* ». De nombreux travaux ont déjà été menés dans ce sens [FP 02][Wam 03][CMV 03] pour des systèmes d'information. Les approches s'appuient lors du déploiement sur des intergiciels ou

machines virtuelles sans prise en compte de la QdS liées à l'utilisation de ressources. Les études menées [BDD][HB 03] autour des SETR s'intéressent aux architectures matérielles et aux communications avec l'environnement (définition de machines virtuelles, de langages pivot et de règles de déploiement) mais sans prise en compte de la QdS. On retrouve donc le même problème que ceux rencontrés lors de la mise en place du paradigme composant. Les approches fournissent des concepts qui doivent être adaptés au domaine des SETR, soient :

- fournir des modèles pour les architectures applicatives en intégrant des contraintes de QdS ;
- fournir des modèles génériques indépendants des supports d'exécution pour les architectures matérielles. Dans ce cadre, il est nécessaire de réfléchir à des modèles de machines virtuelles, intégrant des contraintes de QdS ;
- établir des techniques de transformation de modèles, correctes vis-à-vis de la QdS, assurant le déploiement des architectures applicatives sur les architectures matérielles.

En s'inspirant des études menées et présentées dans ce rapport, nous donnons maintenant quelques pistes de réflexion pour mener à bien le déploiement de SETR.

V.2.2 Architecture applicative

Afin de valider la conformité du déploiement, il faut fournir une sémantique d'exécution pour l'architecture applicative à déployer :

- si on ne prend pas en compte les durées, le modèle ne peut représenter que des dates d'activations d'actions [ABS-1 02]. A l'inverse, si on modélise des durées sous forme d'intervalles, ces dernières représentent une contrainte sur l'implémentation. Cela revient alors à faire une hypothèse forte sur l'architecture matérielle utilisée, ce qui limite le découplage entre les deux modèles. L'idéal serait alors certainement de proposer des modèles génériques et paramétrés à instancier suite au déploiement ;
- modéliser une politique d'ordonnancement spécifique permet de contrôler le comportement temporel de l'architecture applicative mais réduit fortement la réutilisation des modèles. A l'inverse, modéliser une famille d'ordonnancements possibles, revient à définir un ensemble de solutions acceptables à considérer lors du déploiement. Une solution à investiguer serait de construire, lors du déploiement, un ensemble d'ordonnancements valides, en s'appuyant par exemple sur les techniques de synthèse d'ordonnanceurs [AGS 02].

V.2.3 Architecture matérielle

Afin de pouvoir déployer une architecture applicative sur un support d'exécution quelconque, en s'inspirant des travaux menés autour du MDA, une abstraction des architectures est nécessaire et passe par la virtualisation des plates-formes d'exécution et la mise en place d'intergiciels. Une plate-forme d'exécution virtuelle représente une famille de plates-formes réelles (cf. figure 8). La plate-forme virtuelle permet en particulier de définir une sémantique d'exécution et est donc adaptée au prototypage des SETR et à la validation au plus tôt des architectures [SG 05]. Pour intégrer les contraintes de QdS liées à l'utilisation de ressources, elle doit pouvoir être configurée ou paramétrée lors du déploiement. Une plate-forme réelle est alors soit une implémentation de la plate-forme virtuelle, soit une spécialisation de cette plate-forme pour un contexte d'utilisation donné.

Pour mettre en place une telle démarche, il faut d'abord définir le concept de plate-forme. Pour les systèmes visés, la plate-forme concerne le support d'exécution soient les machines, le système d'exploitation et les services de communication, en particulier avec l'environnement. Dans les systèmes d'information, ces éléments sont classiques (machines de type PC, système de type Windows® ou Linux®, IHM) et peuvent être facilement virtualisés (machine virtuelle Java®, *virtual mouse* de Windows®). Pour les SETR, les cibles sont plus hétérogènes (IHM réduit, OS et machines dédiés) et les contraintes de QdS forment un élément clé de la caractérisation des plateformes pour assurer la correction du déploiement.

Il faudrait donc développer des modèles génériques et abstraits des plates-formes d'exécution. En particulier, les communications avec l'environnement sont un point clé de la correction des systèmes. Il serait ainsi utile de proposer une plate-forme de communication virtuelle avec l'environnement extérieur. Dans ce cadre, on propose de poursuivre les premières études menées dans [HB 03] en intégrant des contraintes de QdS. La difficulté provient du fait que les caractéristiques de QdS, comme le pire temps de réponse, dépend de la politique de déploiement et ne peuvent être donc connues a priori.

V.2.4 .Déploiement

En reprenant les conclusions de l'AS CAT[19], il apparaît que la gestion de la QdS lors du déploiement pose un problème de recherche ouvert sur la configuration des intergiciels vis-à-vis des applications et de leurs contraintes de QdS [Dep 05], soient la configuration du déploiement (lien entre l'applicatif et les services de l'intergiciel) et de l'intergiciel lui même (configuration selon l'applicatif et les contraintes de QdS). En s'inspirant des travaux autour de Qinna, on peut noter que l'implémentation et donc le déploiement doit s'appuyer sur des compromis. Les compromis passent par la contractualisation des relations entre les composants applicatifs et les intergiciels et la « fractalisation » des intergiciels. On doit ensuite définir des stratégies de mise en place et de gestion des contrats entre les divers composants pour assurer une gestion efficace de la QdS.

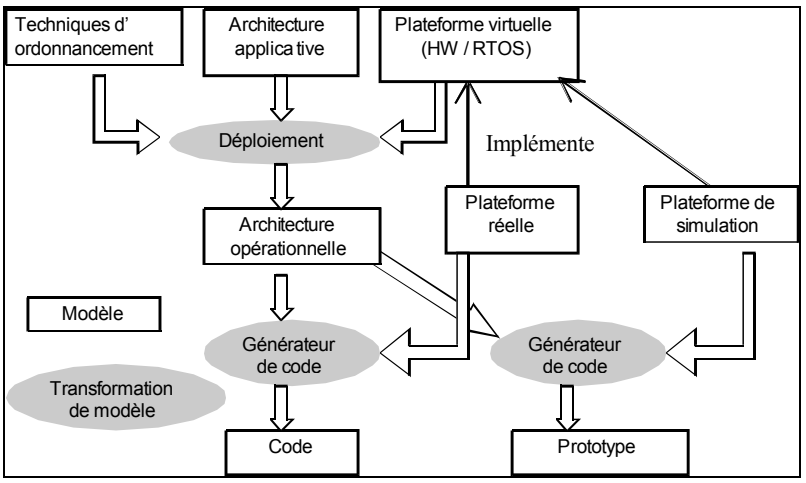

Figure 8 : méthode MDD pour le développement de systèmes embarqués temps réel à base d'une platefrome d'exécution virtuelle

[19] AS CAT : Action Spécifique 195 du CNRS sur les Composants et Architectures Temps Réel

Pour automatiser le déploiement, nous proposons l'étude comparative de diverses stratégies pour une même famille d'architectures applicatives au regard de critères de QdS. Ce travail a été initié pour des implémentations multitâches en environnement monoprocesseur [Bab 02] pour assurer le respect des échéances. L'objectif était alors d'évaluer plusieurs stratégies de mise en place de tâches au sein d'un modèle objet. De même des études récentes proposent des comparaisons pour le déploiement de composants ou d'objets sur une architecture multitâche [FSA 05][GH 05]. L'effort doit aussi porter sur la mise en place de protocoles de communication adéquats entre entités concurrentes pour les systèmes distribués. L'objectif final de ces études doit être de définir des règles de déploiement, correctes et optimisées du point de vue de la QdS, pour des classes d'applications spécifiques.

En conclusion, lors du développement des SETR, les choix à effectuer sont nombreux (modélisation des architectures, techniques de raffinement et de validation, sémantique des modèles, déploiement) et peuvent s'avérer complexes. Il est donc difficile et certainement illusoire de proposer une approche globale et générique qui couvre l'ensemble du cycle de développement. Par contre, pour une famille d'applications donnée, il serait intéressant de proposer des solutions globales formalisées et automatisées pour construire des architectures de qualité et des implémentations efficaces et correctes. Dans ce cadre, nous proposons de poursuivre et d'étendre les études menées en nous concentrant sur un domaine d'application.

C'est l'objet du projet de recherche que nous proposons maintenant. L'application visée est une application de surveillance d'une zone (forêt, autoroute, bâtiment) par un réseau de capteurs. L'objectif est de faire remonter des informations, dont certaines avec une contrainte sur le temps de réponse, vers une application *puit*. Il faut de plus assurer une durée de vie maximale au système difficilement accessible une fois déployé.

Nous avons choisi le thème des réseaux de capteurs car il permettait de proposer un projet réaliste en s'appuyant sur des compétences locales. Le CITI est en effet un membre actif du projet RECAP[20]. De plus, le thème est en prolongement des travaux menés sur les architectures et la caractérisation des flots de données et d'événements. Il intègre des contraintes diverses de QdS et permet d'aborder de nouvelles problématiques (architecture distribuée, aspects stochastiques et facteur d'échelle). Il permet de fédérer diverses thématiques de recherche autour des SETR. Enfin, la partie fonctionnelle ne pose pas de problèmes en soi au vu des applications visées par le projet.

Nous présentons maintenant plus en détail le contexte et les objectifs du projet.

[20] RECAP : plate-forme du CNRS/DSTIC pour les réseaux de capteurs, mise en place par quatre laboratoires en France (le CITI à Lyon, le LAAS à Toulouse, le LIFL à Lille et le LIP6 à Paris)

V.3 Projet de recherche

V.3.1 Contexte de l'étude

La miniaturisation des micro-systèmes électro-mécaniques a permis le développement d'un nouveau type de réseaux : les réseaux de capteurs. Un réseau de capteurs est un système dédié composé d'un ensemble de nœuds autonomes. Chaque nœud, appelé capteur, contient un capteur de données physiques (température, pression, …), une source d'énergie ou batterie et un lien de communication sans fil [DL 05]. Les capteurs sont déployés dans un environnement inconnu, doivent créer spontanément un réseau et s'adapter dynamiquement aux fautes.

Les applications de ces nouveaux réseaux sont multiples qui vont du domaine militaire (surveillance de zones de conflit) aux applications civiles (surveillance de l'activité sismique de zones sensibles, de la résistance des infrastructures et de matériaux suite à un tremblement de terre, de pollutions, incendie, etc.). On propose de classer les réseaux de capteurs au travers des diverses topologies, augmentant ainsi leur complexité :

- capteurs fixes à positions absolues connues, par exemple des capteurs posées le long d'une autoroute ou intégrés dans un bâtiment ;
- capteurs fixes à positions inconnues, par exemple des capteurs « semés à la volée » dans la nature.
- capteurs mobiles à positions relatives connues, par exemple des capteurs fixées sur un véhicule ;
- capteurs mobiles à positions variables, connues ou inconnues, par exemple des capteurs de suivi de biens, de populations animales, de personnes, des capteurs dans le corps humains.

Afin de simplifier les études sur les protocoles et l'agrégation de données, nous considérons dans un premier temps une topologie fixe, soient des capteurs fixes, à position connue ou inconnue. L'objectif des applications basées sur ces réseaux de capteurs est de produire une information de haut niveau selon une contrainte de QdS acceptable. Par exemple, les propriétés recherchées peuvent s'exprimer sous la forme : « des capteurs de chaleur dispersés dans une forêt permettent de signaler un début d'incendie dans un délai maximal de 5 secondes, avec une fiabilité de 95% ».

De notre point de vue, un réseau de capteurs est composé de trois parties : les capteurs, une application *puit* et le bus de communications entre les deux (cf. figure 9). Par capteur, on entend ici le capteur physique et le système de mesure associé. Les capteurs émettent des informations en direction de leur voisinage et en reçoivent donc de leurs voisins. Les données peuvent être alors agrégées, filtrées et ré-émises. Le bus de communication comprend le medium physique et le protocole de communication réparti sur les machines. Au final, l'application *puit* collecte les informations, souvent redondées, diffusées par les capteurs.

L'objectif du projet est de fournir des outils et méthodes pour aider à la construction, au déploiement et à la validation de tels systèmes, soient des concepts architecturaux pour construire et implémenter de tels systèmes, et des techniques formelles pour évaluer les caractéristiques de QdS. Les concepts développés devront être génériques pour pouvoir être étendus à d'autres domaines des SETR.

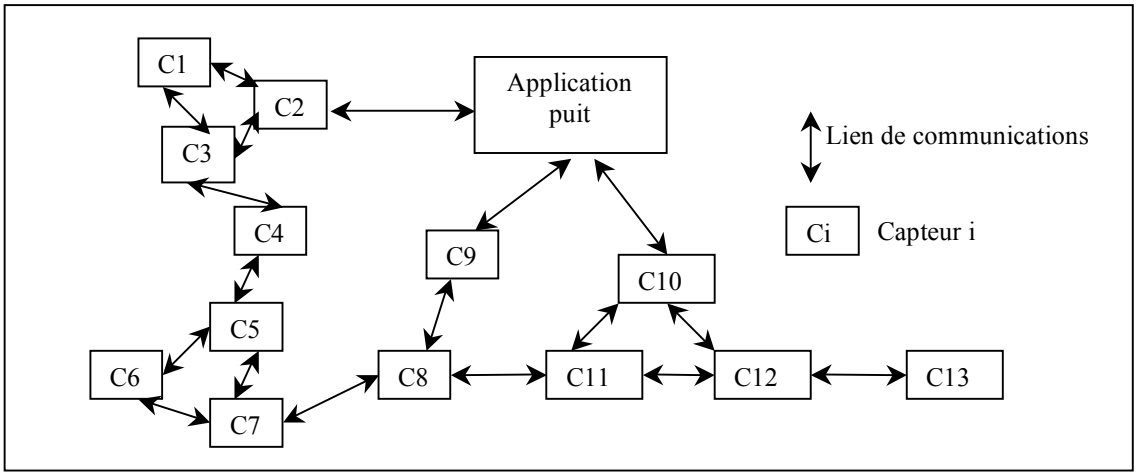

Figure 9 : une architecture matérielle de réseau de capteur

Au vu des contraintes de QdS considérées, il est nécessaire d'évaluer le temps de réponse des systèmes et d'optimiser leur durée de vie. Pour le premier aspect, il est nécessaire d'évaluer les performances des systèmes d'acquisition, de communication et de réfléchir à l'impact de la composition d'informations sur la QdS obtenue. Au niveau de la gestion de l'énergie, il faut réfléchir à la gestion du cycle de vie, soit l'ordonnancement, des capteurs afin d'assurer une durée de vie maximale au système global. Il est à noter que les politiques de mesures et de communication impactent aussi fortement la consommation d'énergie des systèmes.

Nous présentons maintenant les sujets d'étude liés à ce projet de recherche. Chaque sujet est présenté indépendamment et peut être considéré comme un lot. Nous étudierons ensuite les moyens à mettre en œuvre pour la réalisation de ces sujets avant de conclure.

V.3.2 Projets

QdS fournie sur la mesure de capteurs

La QdS fournie par le réseau de capteurs dépend de la QdS fournie par chaque capteur. Le système d'acquisition considéré est composé d'un capteur physique, d'un contrôleur matériel du capteur physique, d'un composant logiciel en charge de l'acquisition des données lues sur le capteur et d'un composant en charge des communications (émission des données) (cf.figure 10).

Dans cette première étude, on s'intéresse à la définition d'outils formels pour l'estimation de la QdS fournie par le composant d'acquisition. La première tâche de l'étude passe donc par une modélisation formelle du système étudié. Dans un premier temps, on propose l'utilisation d'automates temporisés communicants (IF, Kronos ou UPPAAL). A partir des modèles proposés, des techniques seront développées pour extraire les caractéristiques de QdS. Dans les applications visées, on s'intéresse à la fréquence de mise à jour des données, la gigue sur les dates de mise à jour, l'age de la donnée et le taux de pertes. Afin d'extraire les caractéristiques de QdS, une logique doit être proposée décrivant formellement les propriétés attendues. Cette logique doit nécessairement s'appuyer sur le flot de données traité et donc intégrer les notions d'occurrences tels que définies par la logique temps réel. L'étude devra alors proposer une méthodologie formelle d'instrumentation des modèles afin d'appliquer la logique proposée.

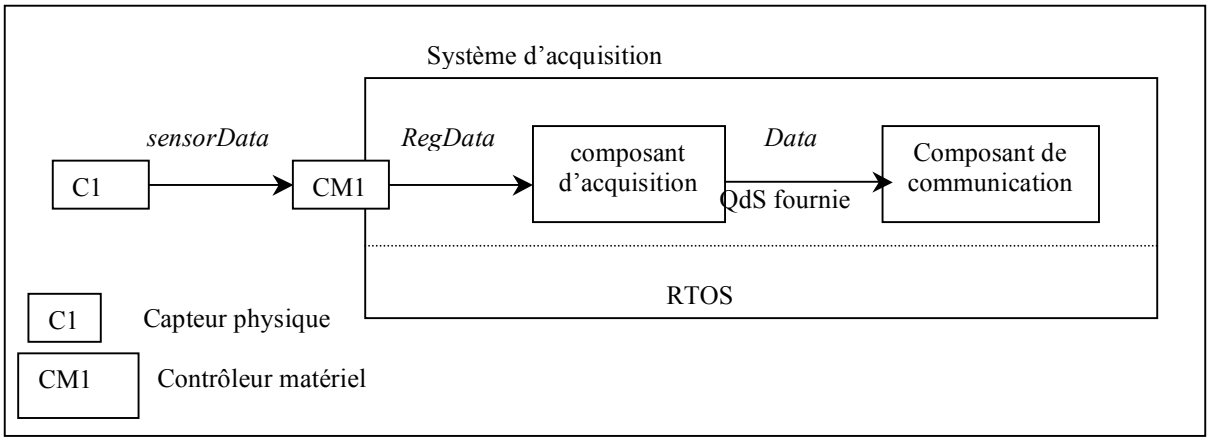

Figure 10 : architecture du système d'acquisition du capteur

Au niveau de la méthode, nous préconisons une approche incrémentale en s'intéressant d'abord à des systèmes simples. Ceci permettra de contrôler la correction des modèles par des approches analytiques. Ensuite, on pourra intégrer des comportements complexes à tous les niveaux :

- émission irrégulière du capteur (rafales) ;
- multi-capteurs (redondance matérielle) ;
- capteurs accessibles via un réseau (multiplexage de données capteurs) ;
- architecture multitâche pour les composants logiciels ;
- points de variabilité temporelle dans les modèles (durée variable, aspects stochastiques).

et à l'évaluation de caractéristiques de QdS complexes :

- nombre maximal de pertes consécutives entre les instants t1 et t2 ;
- nombre maximal de copies consécutives ;
- écart d'age minimal et maximal entre deux données.

Pour valider les résultats théoriques obtenus, on devra réaliser des mesures sur les plates-formes réelles. Dans un premier temps, les résultats obtenus doivent permettre de proposer des théories modélisant le comportement du système étudié du point de vue de la QdS. Dans un second temps, il serait intéressant de mesurer l'impact du choix d'une architecture sur les caractéristiques de QdS obtenues, le but étant alors de proposer des méthodologies de choix ou d'adaptation d'architectures.

Ce travail est un prolongement des travaux menés sur la modélisation en IF et la vérification d'une implémentation multitâche. L'idée est ici de se limiter au domaine des logiciels d'acquisition en se concentrant sur des propriétés typiques de flots de données (taux de perte, fréquence de mise à jour, retard). La maîtrise des performances de ce type de logiciels doit aider à la mise au point de leur architecture. Cette étude a déjà démarré et des premiers modèles exprimés en IF ont été proposés. Ces modèles ont été instrumentés afin d'obtenir les caractéristiques de perte et de retard [BJB-1 05][BJB-2 05]. Les modèles ont été prouvés sur des systèmes simples par comparaison avec une étude analytique à base d'équations diophantiennes. Ils fournissent des résultats réalistes, comparé à une analyse de type RMA et permettent d'aborder des systèmes complexes.

Une fois la QdS du système d'acquisition caractérisée, on s'intéresse à la QdS du système de communication.

Qualité de service pour les réseaux de capteurs sans fil

Dans le cadre des réseaux de capteurs, les réseaux sont sans architecture fixe, constitués de nombreux nœuds contrôlant une zone de couverture réduite et déployés de manière dense dans un environnement hétérogène. Lors de leur déploiement, ces nœuds doivent s'auto-organiser. Ils communiquent entre eux en mode ad hoc, i.e. sans l'utilisation nécessaire d'une station de base. De plus, au cours de son fonctionnement, le réseau doit s'adapter dynamiquement aux changements de l'environnement comme la dégradation de la qualité du lien radio et les pannes de certains nœuds.

Au vu des applications considérées, les protocoles doivent respecter des contraintes de QdS telles le temps de réponse et la tolérance aux fautes. Cette étude a donc pour objectif de proposer des protocoles ad hoc ainsi que des modèles formels permettant de vérifier les propriétés de QdS de ces protocoles. La démarche proposée est la suivante :

- selon les applications envisageables, il faut d'abord définir les caractéristiques de QdS devant être fournis par le réseau ;
- on doit ensuite évaluer les performances des protocoles ad hoc existants, soit estimer les pires temps de réponse pour les différentes opérations (gestion d'erreurs, mode d'accès au médium, routage, etc.) ;
- dans le cas où le pire temps de réponse n'est pas acceptable, il faut proposer des solutions pour soit changer la topologie du réseau (par exemple, ajouter des capteurs à des positions à déterminer dans le but de créer de nouvelles routes), soit proposer de nouveaux protocoles de communications (de niveau MAC et de niveau routage) ;
- au niveau architectural, on doit pouvoir estimer le niveau de redondance de capteurs nécessaire au bon fonctionnement de l'application. En effet, les alarmes doivent être traitées dans les délais, même en cas de perte d'un ensemble de capteurs ou de corruption ou perte de messages ;
- dans la plupart des travaux existants sur les réseaux sans fil les hypothèses sur le lien radio sont excessivement simplifiées et donc peu réalistes. Il faudrait alors prendre en compte des hypothèses plus réalistes permettrait de proposer des protocoles MAC mieux adaptés aux réalités de l'environnement. Cela permettra également d'étudier des erreurs de transmissions réalistes ;
- enfin, il est nécessaire d'évaluer la consommation d'énergie de toute proposition. Bien sûr, ce critère sera probablement antagoniste avec les performances et la tolérance aux fautes, et un compromis pertinent devra être trouvé.

Ces travaux passent par une phase importante de formalisation du système, de son environnement et des différents protocoles de communication grâce à un formalisme ayant un bon pouvoir d'expression (parallélisme, modularité, contraintes, temps) et de validation des temps de réaction et du niveau de redondance. Le modèle devra de plus supporter le passage à l'échelle (les réseaux de capteurs comportant souvent un nombre conséquent de nœuds), ce qui posera alors inévitablement le problème de l'explosion combinatoire.

Ce projet est exploratoire et a été établi en étroite collaboration avec les membres de l'axe protocole du CITI. Ce travail est en effet à la croisée des domaines portant sur les réseaux ad hoc et l'évaluation de performances dans un contexte temps réel. Pour la partie formalisation, ce travail est un prolongement des travaux de thèse effectués par Karen

Godary sur l'évaluation de temps de réponse pour les services d'un réseau de terrain en présence de fautes. Pour la partie protocolaire, une première étude a été réalisée qui propose un protocole temps réel pour la remontée d'alarmes dans des réseaux de capteurs à une dimension, de type « surveillance d'autoroute » [WA 05]. Au final, ce projet couvre des domaines de compétence allant des réseaux ad hoc aux réseaux temps réel en passant par les modèles formels temporisés et stochastiques.

Les études précédentes se focalisent sur la QdS pour un composant donné du système (capteur, réseau). Il reste néanmoins nécessaire d'intégrer l'ensemble des éléments pour assurer une QdS globale au système. Ceci pose alors le problème de la composition des composants et de leur QdS.

Modèle d'architecture pour le système d'acquisition

Le contexte fortement distribué et collaboratif des réseaux de capteurs pose le problème du déploiement des applications, de la sémantique de fusion des informations collectées sur les capteurs et de la QdS résultante. Les applications peuvent être déployées sur des plates-formes au comportement dynamique et sur un nombre de capteurs a priori inconnu. Suite au déploiement, il faut évaluer les performances globales du système, soit ici la fréquence de mise à jour, le pire retard et le taux de perte.

L'objectif de cette étude est de fournir des concepts pour gérer le déploiement des applications et des outils formels pour l'évaluation de la QdS globale du système. On considère une architecture à composants et on fournira des règles d'assemblage et de déploiement de ces composants. Il convient de plus de préciser la sémantique de composition de la QdS et d'assurer l'équivalence des modèles lors du déploiement.

Au niveau de l'architecture applicative, le modèle doit représenter les fonctionnalités attendues du système sans présupposer des capteurs réellement déployés. On proposera alors un modèle d'interconnexion entre des capteurs dit *abstraits*, modélisant une plate-forme virtuelle, et l'application *puit*. Cette vue doit modéliser les principes d'assemblage de données en provenance de capteurs. Par assemblage, on comprend l'interprétation, le formatage, et la fusion de données. La sémantique d'assemblage concerne aussi bien les aspects fonctionnels que temporels (cf. figure 11).

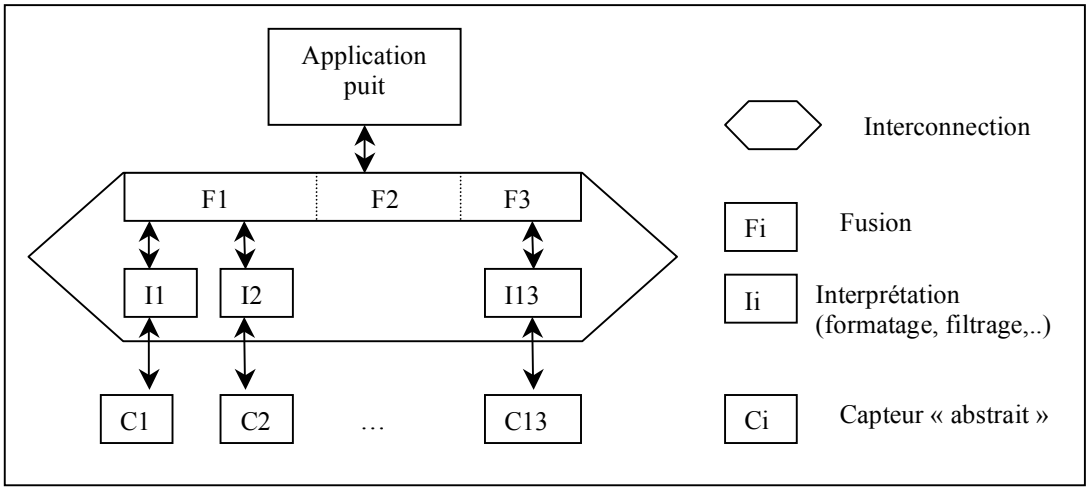

Figure 11 : architecture applicative du système d'acquisition du capteur

L'architecture matérielle, soient ici les capteurs et les protocoles de communication, est considérée comme connue et caractérisée du point de vue de la QdS fournie (cf. projets précédents). L'architecture opérationnelle, elle, est obtenue par déploiement de l'architecture applicative. Le déploiement correspond au placement des composants applicatifs sur les composants matériels et à la mise en place de composants supplémentaires ayant pour rôle la prise en compte des communications et l'adaptation nécessaire, de la plate-forme virtuelle sur la plate-forme réelle au niveau de la QdS.

Pour chaque modèle, l'étude apportera la formalisation nécessaire à l'aide de méta-modèles et d'une sémantique formelle à base d'automates temporisés communicants. De même, le déploiement doit être méta-modélisé et on doit proposer des méthodes formelles de validation de la transformation. Des critères de qualité pourront être définis afin de guider le déploiement.

Ce travail se situe dans le prolongement des études menées autour de la structuration à base de composants et de la formalisation à l'aide de SDL et IF. Des études proches de ce thème ont déjà démarré.

Des principes architecturaux, basés sur MDA, faisant la séparation claire entre l'architecture applicative, la plate-forme d'exécution réelle (le système d'entrée/sortie) et le modèle d'interconnexion entre les deux (filtrage, mise en forme, interprétation et adaptation) ont été proposés [AB-1 05] [AB-2 05]. Chaque composant possède une sémantique de QdS et les travaux proposent des règles de composition et d'interprétation de la QdS. Ainsi, on s'assure de la correction d'un assemblage donné par vérification de propriétés sur la couche d'adaptation et de communication entre la couche d'entrée/sortie et l'application.

De même, des études ont débuté pour modéliser et valider des propriétés de temps de réponse pour des systèmes de télécommunications confrontés au facteur d'échelle. L'étude, menée conjointement avec France Telecom R&D, se propose de définir une approche orientée modèle pour décrire l'environnement (les clients du service), l'architecture applicative, les ressources (machines et mediums de communication) et le déploiement. Ces méta-modèles s'appuient sur le profil UML pour le temps réel OMEGA-RT (LSC pour le comportement, UML pour la structure, IF pour la sémantique). On propose ensuite des politiques de réduction des LTS résultants, basées sur la sémantique des applications. La combinatoire des cas peut en effet être réduite en considérant que les clients possèdent le même comportement [HCB 05].

Au delà du temps de réponse du réseau de capteurs pour transmettre une information mesurée, un réseau de capteurs possède des contraintes fortes de durées de vie, principalement liée à la gestion de l'énergie.

Mise en place d'un cycle de vie

Le contexte des réseaux de capteurs pose le problème de l'énergie nécessaire à la vie de ce réseau, et la manière dont cette énergie est répartie et consommée. On peut traiter ce problème soit en mettant en place des techniques d'optimisation dites globales visant à améliorer la durée de vie du réseau, soit des techniques d'optimisation dites locales visant à améliorer la durée de vie de chaque capteur. L'optimisation locale doit alors s'appuyer

sur un modèle fin des profils de consommation de l'architecture matérielle. De son côté, l'optimisation globale vise une politique d'organisation globale de l'activité du système.

On se propose de travailler aux deux niveaux. A un premier niveau, on se propose d'utiliser des techniques de synthèse de contrôleurs pour définir des politiques d'ordonnancement global de l'activité des capteurs (fréquence d'émission, mise en réveil / reprise) en s'appuyant sur une abstraction forte du matériel. Cette étude a pour objectif de définir un cycle de vie optimum des capteurs afin de maximiser la durée de vie globale du système. Ensuite, l'étude se propose d'optimiser la consommation d'énergie pour chaque capteur en s'appuyant sur une gestion fine du logiciel et du matériel sous-jacent.

Gestion du cycle de vie du réseau

Un objectif des systèmes visés est d'assurer une durée de vie maximale au réseau de capteurs. En présence d'une importante complexité comportementale et de possibles fautes, la nécessité d'une reconfiguration dynamique s'ajoute à la problématique de la correction. Pour traiter de manière efficace l'ensemble des comportements d'un système, l'utilisation des techniques formelles apparaît comme une piste intéressante. Une caractéristique essentielle à étudier est alors la continuité du service (par exemple en présence de pannes) qui peut requérir une réadaptation de la topologie du réseau et/ou des communications entre les capteurs. Nous proposons une démarche axée sur la reconfiguration dynamique, dans le but d'effectuer cette réadaptation. Cette démarche est composée d'une étape hors ligne et d'une étape en ligne

- synthèse : à travers une analyse exhaustive du système, effectuée statiquement (hors ligne), synthétiser le contrôleur qui englobe l'ensemble des stratégies de reconfiguration souhaitées ;
- fonctionnement : le contrôleur synthétisé est composé avec le réseau de capteurs. Il observe à tout instant l'état global du système, et l'empêche d'atteindre des configurations incompatibles avec les objectifs de synthèse ;

La contrainte à minimiser lors de la synthèse est la consommation d'énergie globale du réseau. On pourra alors identifier des stratégies selon des topologies bien identifiées, comme par exemple « griller » un capteur dans une zone dense afin d'économiser un grand nombre de capteurs proches.

Gestion du cycle de vie d'un capteur

Au plus près des circuits électroniques, voire à l'intérieur des composants, sont mises en œuvre des techniques appelées « de basse consommation ». Par exemple, les microcontrôleurs PIC™ de la société Microchip® sont dénommés nanowatt, car leur consommation en mode veille a été optimisée. Par contre, la consommation en mode d'activité ne fait l'objet d'aucune optimisation in-situ dans le circuit, et elle reste à la charge du concepteur de logiciel. Une première stratégie est d'optimiser la consommation au niveau architectural par synthèse orientée par les contraintes de consommation. La deuxième stratégie correspond à la gestion active, à l'exécution, de l'énergie, dont le mode de veille fait également partie. La mise en place de politiques dynamiques de gestion de la QdS pour un capteur passe par la caractérisation des divers composants du système. On retrouve classiquement l'environnement du système (le capteur physique et les autres capteurs), les ressources matérielles du capteur, l'exécutif et l'application embarquée sur le capteur.

Pour caractériser l'activité du réseau de capteur, on propose l'observation des communications d'un réseau réel. L'analyse doit ensuite proposer des modèles heuristiques, tabulés de cette activité. La seconde étape consiste à définir les données et les traitements associés qui sont nécessaires au niveau de l'écriture des logiciels des capteurs. Enfin, on doit considérer des modèles fins de consommation du matériel. Ces informations doivent permettre de construire une vue globale des composants et de leurs profils de QdS. Tous ces modèles doivent permettre de proposer et d'évaluer des stratégies de gestion dynamique de l'énergie. Les stratégies proposées doivent résoudre des compromis afin d'implanter des politiques du type « ne pas mettre en veille un sous-système qu'il faut réveiller juste après, afin d'économiser l'énergie et assurer la continuité de service ».

Au niveau de la structuration, ce projet est un prolongement des études sur l'approche à composants, soit l'application de Qinna à un cas réel. Pour la définition des politiques de gestion de l'énergie, ce projet est exploratoire et s'inspire pour une large part d'un projet BQR (Bonus Qualité Recherche), auquel j'ai participé, déposé par les laboratoires de l'INSA CITI, LAI et CEGELY et soutenu par l'INSA. Ce projet a été établi par les membres du CITI (architectures matérielles), les membres du LAI (synthèse de contrôleurs) et les membres du CEGELY (gestion d'énergie pour des composants matériels). Ce travail est en effet à la croisée de divers domaines portant sur la synthèse de contrôleurs, les architectures matérielles, les systèmes d'exploitation, la compilation et le réseau.

V.3.3 Mise en œuvre et suivi

Collaborations

La mise en œuvre d'un tel projet requiert des collaborations avec les personnes qui utilisent et mettent en œuvre les réseaux de capteurs, soient les *clients* d'un tel projet. Il faut donc identifier une structure (entreprise ou laboratoire) qui utilise des réseaux de capteurs de surveillance. On peut citer le laboratoire de Géophysique [21] qui utilise des capteurs de surveillance de l'activité sismique. A défaut, le groupe de travail doit identifier des projets et construire des cahiers des charges réalistes. Enfin, le projet doit pouvoir être en relation avec ceux qui produisent les technologies, soit ceux qui fabriquent les capteurs. Au sein de l'INSA, on peut citer l'Equipe Microcapteurs et Microsystèmes Biomédicaux du laboratoire LPM. Ces deux laboratoires n'ont pas été contactés pour élaborer ce projet et sont simplement cités à titre d'exemple.

Expérimentation

Bien sûr, les résultats théoriques obtenus dans les diverses études devront être confortés par des expérimentations pratiques. Une plate-forme d'expérimentation commune est une occasion d'échange supplémentaire entre les participants au projet. Une plate-forme d'expérimentation doit permettre d'implanter les propositions, de les valider et de les évaluer en pratique. La plate-forme doit être hétérogène en terme de support d'exécution avec des architectures matérielles à base d'électronique programmable (FPGA[22]), de machines dédiées (microcontrôleurs utilisant un RTOS) ou de machines banalisées (de type PC sous Linux) pour le prototypage. Les protocoles de communication doivent être ouverts et reprogrammables, surtout au niveau MAC. Enfin, la plate-forme doit être ouverte pour permettre son instrumentation ou sa modification si nécessaire.

[21] www-lgit.obs.ujf-grenoble.fr/
[22] FPGA : Field Programmable Gate Array : réseau de cellules programmable

Organisation

Un tel projet requiert une gestion de projet et une organisation des moyens humains et matériels. On propose d'avoir un responsable global du projet et un responsable par lot qui soit un chercheur ou un enseignant-chercheur. La réalisation de chaque lot doit permettre de lancer de une à deux thèses. Afin de faire vivre le projet, de le diffuser et de l'améliorer par des regards externes, il est utile de prévoir le financement de deux ou trois post-docs. Enfin, la mise en place pérenne d'une plate-forme d'expérimentation et son suivi requiert l'activité d'un ingénieur de recherche à temps plein. Pour être mené à bien, un tel projet doit regrouper environ 5 chercheurs, de 5 à 10 étudiants en thèse, 2 à 3 post doc et un ingénieur de recherche. La durée d'un tel projet est estimée à 5 ans. Un budget de fonctionnement est à prévoir, hors plate-forme d'expérimentation, pour assurer les déplacements et les frais de fonctionnement.

Au niveau de l'organisation, on propose de s'appuyer classiquement sur des réunions mensuelles de suivi de projet. Les réunions doivent être l'occasion de présentations par les membres du projet. Ces présentations ont pour objectif de s'enrichir mutuellement, d'échanger des regards et de susciter des collaborations lorsque les thématiques de recherche sont proches. Les présentations peuvent être de trois types :

- état de l'art par les permanents : présentation avec un objectif de formation et de formalisation ;
- travaux en cours par les thésards : présentation avec un objectif de clarifier, faire avancer, proposer des pistes ;
- travaux finalisés avant une présentation, en groupe de travail ou en conférence : présentation avec un objectif de communication des résultats et de pré-soutenance.

V.3.4 Conclusion

Le projet de recherche proposé doit permettre de fournir des outils et méthodes pour construire et valider des systèmes dédiés que sont les réseaux de capteurs. On se limite ici aux capteurs à position fixe. Le projet possède l'originalité de regrouper divers aspects du développement des SETR afin d'assurer le respect de propriétés temporelles telles que la durée de vie et le temps de réponse. Le projet intègre :

- des aspects architecturaux au travers des composants pour construire les capteurs et le système global ;
- l'utilisation de modélisation exhaustive à base d'automates temporisés communicants pour l'évaluation de la QdS des systèmes d'acquisition et de communication ;
- la génération du cycle de vie des capteurs et du réseau, à base de synthèse de contrôleurs ;
- la mise en place de protocoles de communication en mode ad hoc pour le temps réel ;
- la prise en compte d'aspects stochastiques et de facteurs d'échelle dans un environnement distribué ;
- la prise en compte des performances du matériel pour la construction et la caractérisation des performances du logiciel ;
- la mise en place d'une plate-forme de test.

Ce projet regroupe des compétences diverses qui vont des architectures logicielles et matérielles, aux systèmes d'exploitation, aux techniques d'ordonnancement, aux techniques formelles et aux protocoles de communication. Ces compétences permettent de couvrir l'ensemble du cycle de développement des SETR. Les conditions de la réussite d'un tel projet sont, à notre avis, de regrouper et faire collaborer autour d'un même projet de recherche et d'une même plate-forme, des personnes qui utilisent les applications, qui font les technologies (matériel et logiciel) et qui développent les outils et techniques pour assurer le développement des SETR.

Le projet permet de regrouper des personnes aux préoccupations de recherche différentes. Au delà de l'enrichissement inhérent aux projets pluridisciplinaires, l'interaction entre personnes de domaines différents doit aider à formaliser chaque domaine propre et les liens entre les divers domaines. Généralement un domaine de recherche considère un autre domaine comme une boîte noire car il lui est inconnu. Ceci amène alors à faire des hypothèses nécessairement simplificatrices. L'échange d'informations doit alors permettre d'améliorer chaque projet de recherche en fournissant des hypothèses réalistes sur les autres domaines. Au delà, le fait de collaborer sur plusieurs domaines peut amener à définir et lancer de nouvelles problématiques de recherche, à la croisée des domaines.

L'intérêt de choisir un domaine d'application est de pouvoir aborder le problème du développement dans sa globalité. Pour chaque technique proposée (la modélisation, la génération de contrôleurs, la gestion de l'énergie, les protocoles de communication), on pourra s'appuyer sur des hypothèses fortes liées aux spécificités de l'application (activité logicielle, matériel et OS utilisés, architectures, modèles de fautes connues) et à la connaissance des autres domaines.

Enfin, il est à noter que la démarche a été proposée pour démontrer la pertinence d'un projet de recherche sur le développement des SETR orienté par l'application. La démarche n'est donc pas limitée au domaine d'application des réseaux de capteurs et pourrait être développée pour d'autres projets comme par exemple les robots mobiles d'exploration, les robots médicaux ou les applications domotiques.

VI Conclusion

Cette dernière partie permet de conclure ce rapport par un rappel des principaux résultats et des perspectives pour la suite des recherches à entreprendre.

Les études menées ont permis de proposer des principes de mise en œuvre et d'adaptation des langages, paradigmes et techniques formelles pour la formalisation et la structuration d'architectures opérationnelles pour les systèmes embarqués temps réel au comportement dynamique. On s'intéresse dans ces études à la gestion de contraintes de qualité de service liées à l'utilisation de ressources matérielles limitées.

Les études menées ont porté sur l'intégration du paradigme composant, l'utilisation des langages formels SDL et IF et l'adaptation des techniques formelles par simulation exhaustive. On s'est limité à des systèmes dont l'implémentation s'appuie sur une machine monoprocesseur utilisant les services d'un RTOS. Les études se sont plus particulièrement intéressés à proposer :

- une architecture opérationnelle formalisée pour les systèmes multitâches ;
- la mise en œuvre de techniques formelles par modélisation exhaustive (validation comportementale et vérification de propriétés temps réel et de sûreté) ;
- une architecture de gestion dynamique de la QdS pour les systèmes à composants ;
- la formalisation des contraintes par la mise en place de contrats de QdS.

Une bonne architecture se doit d'être à la fois suffisamment abstraite et générique, posséder une sémantique formelle et être implémentable. Les propositions assurent l'abstraction et la généricité des architectures produites par l'utilisation de bloc-type SDL et la mise en place de composants génériques avec Qinna. On note, à ce niveau, l'intérêt porté à la séparation des préoccupations pour la mise en place des éléments de l'architecture. La formalisation permet d'établir des liens avec les techniques de validation et de vérification. Nous avons alors privilégié une approche par simulation exhaustive plus appropriée aux comportements dynamiques. La mise en œuvre des techniques formelles de validation nous a amené à proposer une nouvelle relation d'équivalence comportementale intégrant les phénomènes de mémorisation d'événement et de décalage temporel. Pour la vérification, nous avons proposé un modèle fin du comportement temporel du code et des stratégies de réduction basées sur la sémantique de l'application. La modélisation des propriétés se fait par la mise en place d'observateurs permettant la mesure de performances. Enfin, nous avons développé un prototype de générateur de code pour les architectures multitâches et implémenté un ensemble de cas type avec Qinna.

De notre point de vue, l'apport principal des travaux concerne la prise en compte de la notion d'occurrence d'événements dans les automates temporisés communicants, une modélisation fine du comportement temporel des systèmes pour la vérification et la mise en place à l'aide de composants des concepts nécessaires à la gestion dynamique de QdS.

Les études ont montré que l'effort de structuration à base de composants génériques est un bon moyen de structurer les informations architecturales, même si des efforts sont ensuite nécessaires pour produire un code efficace. Une fois les concepts établis, comme l'ont aussi montré les approches Fractal et Think, il est alors plus facile de dériver des solutions contextuelles pour une application donnée. De son côté, l'utilisation de modélisation exhaustive permet d'avoir une vue d'ensemble des comportements possibles d'un système. Il reste alors à définir des stratégies de réduction et d'interprétation des modèles pour aboutir à des techniques utilisables en pratique. On peut noter que dans les domaines visés, la taille réduite des systèmes (nombre limité de tâches) permet ce type d'approche. Pour ce qui est de développer des modèles indépendamment des plates-formes, de nombreuses études restent encore à proposer. La difficulté vient alors de la

cohérence des modèles vis-à-vis de leur sémantique temporelle, fortement liée à l'implémentation.

On retient de ces études que la mise en place des architectures requiert la définition exacte de ses composants, de leur rôle, et ce à un haut niveau d'abstraction afin de favoriser la réutilisation des concepts. On retient aussi que la mise en place de techniques formelles requiert de maîtriser aussi bien la sémantique des formalismes et des outils que celle des systèmes étudiés. Les problématiques de recherche autour des SETR se pose donc globalement en terme de formalisation des informations manipulées (structure et comportement) soit de sémantique. L'approche IDM, avec sa volonté de tout modéliser, semble alors aujourd'hui la voix à suivre pour décrire, capitaliser et réutiliser les concepts mis en œuvre lors du développement des SETR [GFB 05].

De notre point de vue, l'objectif des études menées ainsi que leurs prolongements doit être de produire des composants logiciels configurables et validés pour un contexte d'utilisation donné. Enfin, des théories doivent être développées construire les systèmes par composition de composants. Ainsi, la construction des architectures opérationnelles et du code pourra être largement automatisée par la réutilisation de composants validés.

Je vais maintenant énoncer quelques points de recherche pouvant donner lieu à des développements ultérieurs. Ces points concernent les architectures et leur formalisation :

- il faut travailler à proposer des modèles permettant le développement de composants indépendamment des plates-formes d'exécution utilisées ;
- la sémantique d'exécution (ordonnancement et concurrence) doit pouvoir être établie formellement au vu des contraintes du système ;
- il faudrait intégrer des mécanismes pour une gestion sûre du cycle de vie des composants du système : gestion des modes de fonctionnement, ajout/modification/suppression « à chaud » d'un composant, adaptation en-ligne, reconfiguration;
- il faudrait travailler à la composition de composants hétérogènes logiciel et matériel ;
- il serait nécessaire d'intégrer la modélisation réaliste du comportement du procédé, d'assurer des liens avec les systèmes continus, le matériel, considérer des aspects stochastiques (perte, fautes) et les facteurs d'échelle ;
- la structuration en composants doit permettre d'aider à la structuration des architectures afin de prendre en compte d'autres contraintes (sécurité, tolérance aux fautes) et d'offrir des capacités d'adaptation aux plates-formes existantes (« fractalisation » des systèmes d'exploitation et des machines virtuelles) ;
- il faudrait automatiser la mise en place d'outils formels pour l'évaluation de propriétés liées à la composition de flots lors de fusion de données et de communications dans les systèmes distribués
- il faudrait étendre la relation d'équivalence pour les comportements complexes (divers modes de fonctionnement), pour valider les techniques de déploiement et pour les systèmes distribués.
- il faudrait développer des méta-modèles afin de développer des outils pour la mise en place et la validation des architectures.

Les dernières perspectives porte sur l'implémentation. Les SETR requièrent une implémentation optimisée afin de respecter les contraintes de place, de consommation et de coûts. Les principes architecturaux étudiés ont pour objet principal d'intégrer des principes

de génie logiciel et la possibilité d'effectuer des preuves. Il est donc nécessaire de réfléchir à des techniques de génération de code efficaces, correctes, orientées par les propriétés de QdS recherchées. En particulier, l'implémentation doit être étudiée dans le cadre des machines virtuelles (mise en place de méta-données, adaptation des plates-formes).

Les études menées doivent aussi être étendues pour considérer les architectures distribuées et faire le lien avec les architectures matérielles. Une solution pour intégrer un maximum de contraintes et considérer l'ensemble du processus de développement est alors certainement de se limiter à un domaine d'application précis.

VII Références

VII.1 Articles de l'auteur

[AB-1 05] J. de Antoni and J.-P. Babau "*A MDA approach for systems dedicated to process control"* 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, 567-570, Hong-Kong (China), Aug. 2005.

[AB-2 05] J. de Antoni and J.-P. Babau "*A MDA-based Approach for Real Time Embedded Systems Simulation*" 9th IEEE symposium on Distributed Simulation and Real-Time Applications, DS-RT'05, 257-264, Montreal(Canada), Oct. 2005.

[ABS-1 02] J.P. Babau, A. Alkhodre, J.J. Schwarz "*Modeling of Real-Time Embedded Systems using SDL*" System on Chip Design Languages, Chap 22, 257-265, Kluwer, 2002.

[ABS-2 02] A. Alkhodre, J.P. Babau, J.J. Schwarz "*Modeling of Real-Time Constraints using SDL for Embedded Systems Design"* IEE Computing and Control Engineering Journal, 189-196, 2002.

[ABS-3 02] A. Alkhodre, J.P. Babau, J.J. Schwarz "*Real time multitasking design based on SDL"* Forum on Design Languages FDL'02, Marseille, Sept.2002.

[AKB 04] A.Alkhodre, A. Khatab, J.-P.Babau, J-J. Schwarz "*SDL and IF for Real time Systems development: specification, design and validation*" IFAC WRTP, 41-47, Istanbul, 2004.

[BBS-1 04] M. Belarbi, J.-P. Babau , J.-J. Schwarz "*Temporal Validation of Multitasking Real-Time Application based on Communicating Timed Automata*" Forum on Design Languages, FDL'04, Lille , Sept. 2004

[BBS-2 04] M. Belarbi, J.-P. Babau, J.-J. Schwarz "*Temporal Verification of Multitasking Real-Time Application Properties based on Communicating Timed Automata*" 8th IEEE symposium on Distributed Simulation and Real-Time Applications, DS-RT'04, 188-195, Budapest, Oct 2004.

[BA 01] J.P. Babau, A. Alkhodre "*A development method for PROtotyping embedded SystEms by using UML and SDL (PROSEUS)"* workshop SIVOEES– ECOOP, Budapest, 2001.

[Bab 00] J.P. Babau "*Object Oriented Design for Real-Time Systems - Response to CE Pereira's Contribution*" Real-Time Systems 18(1), 95-99, 2000.

[Bab 02] J.P. Babau « *Modèle d'architecture orienté objet pour les systèmes temps réel* » Rapport d'activité, CITI, 2002.

[BC 96] J. P. Babau F. Cottet "*Extension of scheduling algorithms in case of conditional or parametric synchronization relationships*", 8th Euromicro Workshop on Real Time Systems, 1996.

[BC 98] J.P. Babau, F. Cottet « *Méthodologie d'analyse temporelle des applications temps réel à contraintes strictes* » Journal Européen des Systèmes Automatisés (APII) , 32 (5-6), 581-608, Sept. 1998

[BJB-1 05] B. Ben Hédia, F. Jumel, J.-P. Babau "*Formal Evaluation of Quality of Service for Data Acquisition Systems"* FDL'05, 579-588, Lausanne, 2005

[BJB-2 05] B. Ben Hédia, F. Jumel, J.-P. Babau « *Qualité de service des pilotes d'équipements pour les systèmes d'acquisitions de données* », Modélisation des Systèmes Réactifs, MSR'05, 47-62, Grenoble, 2005

[BS 97] J.P. Babau, J.L. Sourrouille "*Expressing Real-Time Constraints in an Object-Oriented Approach* " IFAC Workshop on Real Time Programming, 71-76, Lyon, Sept. 1997.

[BS 98] J.P. Babau, J.L. Sourrouille "*Expressing Real Time Constraints in a Reflective Object Model* " Control Engineering Practice , 6, 421-430, 1998.

[GB 05] S. Gérard, J.P. Babau "*Model Driven Schedulabilty Analysis*" Model Driven Engineering for Distributed Real-Time Embedded Systems, Chap 9, 197-204, Hermès, 2005.

[GCB 98] R. Gumzej, M. Colnaric, J.P. Babau, J. Skubich "*Hardware Architecture Components for Real Time Systems Design"* IEEE Slovenia Computer Conference ERK'98, 41,44, , Portoroz, Sept. 1998.

[HCB 05] J.L. Houberdon, P. Combes, J.-P. Babau, I. Augé-Blum "*Validating temporal properties of a deployed application with an MDD approach*" MARTES – MoDELS/UML 2005, Montego Bay (Jamaïque), 37-44, Oct 2005.

[SJH 99] J.-J. Schwarz, K. Jelemenska, Z. Huang, R. Aubry, J.P. Babau "*From CRSM specification to a real-time multitasking execution model*" IEEE International Symposium on Industrial Electronics (Real-Time session), 1, 65-69, Bled (Slovenia) Jul. 1999.

[TOB 04] J.-C. Tournier, V. Olive, J.-P. Babau "*The Qinna Experiment, a Component-Based QdS Architecture for Real-Time Systems"* Workshop on Architectures for Cooperative Embedded Real-Time Systems, in conjunction with the 25th IEEE RTSS, Lisbonne, 2004.

[TOB-1 05] J.-C. Tournier, V. Olive, J.-P. Babau "*A Qinna Evaluation, a Component-Based QdS Architecture for Handheld Systems*" ACM Symposium on Applied Computing, SAC 05, 998-1002, Santa Fe (USA), Mar 2005

[TOB-2 05] J.-C. Tournier, V. Olive, J.-P. Babau "*Qinna, a Component-Based QdS Architecture*" 8th SIGSOFT symposium on CBSE, pp 107-122, Saint-Louis (USA), May 2005,

VII.2 Articles

[Aag 01] J. Aagedal."*Quality of Service Support in Development of Distributed Systems*" PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2001.

[AC 98] C. Aurrecoechea, A. T. Campbell, L. Hauw "*A Survey of QoS Architectures. Multimedia Systems*" 6(3),138–151, 1998.

[ACD 93] R. Alur, C. Courcoubetis, and D. L. Dill "*Model-Checking in Dense Real-Time*" Information and Computation, 104(1), 2-34, Academic Press, 1993.

[AD 90] R. Alur, D. Dill *"Automata for modelling real-time systems*" Automata, languages and programming, LNCS, 443, 322-335, Springer 1990.

[AD 94] R. Alur and D. L. Dill "*A Theory of Timed Automata*" Theoretical Computer Science, 126(2), 183–235, Elsevier Science, 1994.

[ADL 03] J. M. Alvarez, M. Diaz, L. Llopis , E. Pimentel and J. M. Troya "*An Object-oriented Methodology for Embedded Real-time System*", Computer Journal, 46 (2), 123-145, 2003.

[AGS 02] K. Altisen, G. Goessler, J. Sifakis "*Scheduler Modeling Based on the Controller Synthesis Paradigm*" Real-Time Systems Journal, 23(1/2), 55-84, Kluwer, 2002.

[Ait 00] Yamine Aït-Ameur « *Développements Contrôlés de Programmes par Modélisations et Vérifications de Propriétés* » Thèse, Habilitation à diriger les recherches, Université de Poitiers, 2000

[AKZ 96] M. Awad, J. Kuusela and J. Ziegler "*Object Oriented Technology for Real-Time Systems*" Prentice Hall, 1996.

[ALE 04] S. Amundsen, K. Lund, F. Eliassen, R. Staehli "*QuA : Platform-Managed QoS for Component Architectures"* NIK 2004, 2004.

[Alk 04] A. Alkhodre « *Développement formel de systèmes temps réel à l'aide de SDL et IF* » thèse de de l'INSA de Lyon, Septembre 2004.

[Alk 99] A. Alkhodre "*Programmation graphique multitâches pour Windows*" rapport de DEA, 1999.

[AMM 04] H. Aydin, R. Melhem, D. Mosse, P. Mejý´a-Alvarez, ”*Power-Aware Scheduling for Periodic Real-Time Tasks*” IEEE Transactions on computers, 53(5), 2004.

[And 96] C. André “*Representation and analysis of reactive behavior : a synchronous approach*” Computational Engineering in Systems Applications (CESA), 19-29, Lille, 1996.

[APR 02] C André, M-A Peraldi-Frati, J-P Rigault "*Integrating the Synchronous Paradigm into UML: Application to Control-Dominated Systems*", UML 2002, Springer-Verlag, 438-444, Dresden, Oct 2002.

[ARN 02] A. Arnold « *Systèmes de transitions et sémantique des processus communicants* » Masson, 1992.

[ART 05] the ARTIST Roadmap for Research and Development “*Embedded Systems Design*”, LNCS, Vol. 3436, 492 p., 2005.

[BA 02] G. Buttazo, L. Abeni “*Smooth rate adaptation through impedance control”* Euromicro conference on real-time systems, 3-10, Vienna, 2002.

[Ba 91] T. P. Baker, "*Stack-based scheduling of realtime processes*" Real-Time Systems, 3, 67-100, 1991.

[BB 90] F. Bause, P. Buchholz “*Protocol Analysis Using a Timed Version of SDL*” FORTE , Madrid, 1990.

[BBR 02] I. Broster, A. Burns, G. Rodriguez-Navas “*Probabilistic analysis of CAN with faults*” 23th IEEE Real-Time Systems Symposium,269-278, 2002.

[BCG 97] Ballarin, F, M. Chiodo, P. Giusto “*Hardware-Software Co-Design of Embedded Systems: The Polis Approach*” Kluwer Academic, 1997.

[BCG 99] A. Benveniste, B. Caillaud, and P. Le Guernic “*From synchrony to asynchrony”* CONCUR'99, Concurrency Theory, LNCS 1664, 162--177. Springer, Aug. 1999.

[BCK 03] L. Bass, P. Clements and R. Kazman “*Software architecture in practice*”, Addison-Wesley, 2003.

[BCS 03] E. Bruneton, T. Coupaye, J.-B. Stefani “*Specification of the Fractal Component Model*” Technical report, fractal.objectweb.org, 2003.

[BD 00] B. Bérard, C. Durfourd “*Timed automata and additive clock constraints*” information processing letters, 75(1-2), 1-7, 2000.

[BDD 03] P.Boulet, J.L.Dekeyser, C.Dumoulin, and P.Marque “*MDA for soc embedded systems design, intensive signal processing experiment*” FDL03, 2003.

[BEJ 96] P. Binns, M. Englehart, M. Jackson, S. Vestal “*Domain specific software architectures for guidance, navigation and control*” journal of software engineering and knowledge engineering, 6(2), 1996.

[Bel 03] M. Belarbi « *Validation temporelle des applications multitâches temps réel basée sur les automates temporisés communicants* » thèse de l’INSA de Lyon, Décembre 2003.

[Ber 03] G. Bernat "*Response Time analysis of asynchronous real-time systems*" Journal of Real-Time Systems, 25 (2-3), 131 – 156, 2003.

[Ber 92] G. Berry, G. Gontier “*The Esterel synchronous programming language : Design, semantic, implementation*” Science of computer programming, 19(2), 87-152, 1992.

[BFG 99] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier " *IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems* " World Congress on Formal Methods, 307-327, 1999.

[BGK 00] M.Bozga, S.Graf, A. Kerbrat, L. Mounier, I. Ober, D. Vincent “*SDL for Real Time : What is Missing ?*”Proceedings of SAM'00, Grenoble, 2000.

[BGM 01] M. Bozga, S. Graf, L. Mounier, I. Ober, J-L. Roux, D. Vincen “*Timed Extensions for SDL*” Proceedings of SDL-Forum'01, Copenhagen (Denmark), 2001.

[BGO 04] M. Bozga, S.Graf, I. Ober, I. Ober, J. Sifakis “*Tools and Applications: the IF toolset*” International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, SFM-04:RT, LNCS, 3185, Springer, 2004

[BJP 99] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, D. Watkins “*Making components contract aware*” IEEE Computer, 32(7), 38–45, 1999.

[BL 97] H. Ben-Abdallah, S. Leue "*Timing Constraints in Message Sequence Chart Specifications*" Tenth International Conference on Formal Description Techniques FORTE/PSTV'97, Osaka(Japan), Nov. 1997.

[Bou 05] P. Bouyer "*An introduction to Timed Automata*" Ecolé d'été temps reel, Nancy, 79-94, Sept 2005.

[BR 98] B. Selic, J. Rumbaugh "*Using UML for Modeling Complex Real-Time Systems*" ObjecTime Limited, 1998.

[BRC 00] P. Balbastre, I. Ripoll, A. Crespo "*Control task delay reduction under static and dynamic schduling policies*" International Conference on Real-Time Computing Systems and Applications, 522, 2000.

[Bry 92] R. Bryant "*Symbolic Boolean manipulation with ordered binary decision diagram*" ACM Computing Surveys, 24(3), 293-317, 1992.

[BS 91] A. Benveniste, G. Berry "*The synchronous approach to reactive and real-time systems*" , proceeding of the IEEE, 79(9), 1270-1282, Sept. 1991.

[Cam 96] A. Campbell "*A Quality of Service Architecture*" PhD thesis, Computing Department Lancaster University, 1996.

[CAN 91] Robert Bosch GmbH "*CAN specification*" Version 2.0. 1991.

[CCM02] Object Management Group "*Corba Component Model*" CCM V3.0 www.omg.org/technology/documents/formal/components.htm, 2002.

[CE 81] M. Clarke, E. A. Emerson "*Design and synthesis of synchronous skeletons using branching-time temporal logic*" in: D. Kozen (Ed.), Proceedings of the 3rd Workshop on Logics of Programs (LOP'81), Vol. 131 of LNCS, Springer-Verlag, 52-71, 1981.

[CEF 93] E. Clarke , R. Enders, T. Filkorn, S. Jha "*Exploiting Symmetry in temporal logic Model Checking*" Formal Methods In System Design, 9, 77-104, 1993.

[Cha 84] D. Chapiro "*Globally Asynchronous Locally Synchronous Systems*" Thesis, Stanford University, 1984

[CL 90] M.I. Chen, K.J. Lin "*Dynamic Priority Ceilings: a Concurrency Control Protocol for Real Time Systems*" Real Time Systems, 4 (2), 325-346, 1990.

[CMV 03] P.Caceres, E.Marcos, and B.Vela "*A MDA-based approach for web information system development*" wisme,

[COM00] Alan Ira Gordon "The COM and COM+ Programming Primer" Microsoft Technologies Series, Avril 2000.

[CORBA] Object Management Group "*Common Object Request Broker Architecture*" CORBA 3.0 specification v3.0 www.omg.org/corba

[CR 04] F. Cassez, O. Roux "Structural *Translation from Time Petri Nets to Timed Automata*" Workshop on Automated Verification of Critical Systems, AVoCS'04, ENTCS,128,145-160. Elsevier Science Publishers, Juin 2005.

[CS 03] J. Contreras, J.L. Sourrouille, "*Adaptable Objects for Dependability*", Latin American Symposium on Dependable Computing, LADC, LNCS 2847, 181-196, 2003.

[CSB 90] H. Chetto, M. Silly, T. Bouchentouf "*Dynamic Scheduling of Real Time Tasks under Precedence Constraints*" Real Time Systems, 2, 181-194, 1990.

[Del 94] J. Delacroix « *Stabilité et régisseur d'ordonnancement en temps réel* » Technique et Sciences Informatiques,13 (2), 223-250, 1994.

[Dep 05] A.M. Deplanche & All « *Composants et architectures temps reel* » Rapport de recherché de l'AS 195, IRCCyN, RI2005_1, Mars 2005.

[DF 05] A.M. Deplanche, S. Faucou « *Les langages de description d'architectures pour le temps reel* » Ecolé d'été temps reel, Nancy, 31-46, Sept 2005.

[DH 01] W. Damm, D. Harel, "*LSCs: Breathing Life into Message Sequence Charts*" *Formal Methods in System Design*, 19(1), 45-80, 2001.

[DHH95] M. Diefenbruch, E. Heck, J. Hintelmann, and B. Müller-Clostermann. "*Performance evaluation of SDL systems adjunct by queuing models*" SDL-Forum '95, 1995.

[DL 05] J. Delsing, P. Lindgren "*Sensor communication technology towards ambient intelligence*" Measurement science and technology, 16, 37-46, 2005.

[DL 78] S.K. Dhall, C.L. Liu "*On a real-time scheduling problem*" Operations Research, 26(1), 127-140, 1978.

[DOT 96] C. Draws, A. Olivero, S. Tripakis, S. Yovine "*The tool Kronos*" Hybrid Systems III : Verification and Control, LNCS, 1066, 208-219, Springer , 1996.

[Dou 02] B. Douglass "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems" Addison-Wesley,2002.

[Dur 98] E. Durand « *Description et verification d'architectures temps reel : CLARA et les réseaux de Petri temporels* » these de doctorat, Ecole centrale de Nantes, 1998.

[EJB 03] Sun Microsystems "*JSR 153 : Enterprise JavaBeans*" EJB V2.1. Technical report, Java Community Process, 2003.

[ELP 93] R. Elmstrom, R. Lintulampi, M. Pezze "*Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets" Real-Time Systems*, 5(2/3), 249-271, 1993.

[EMP 03] EMPRESS project "*A language and methodology to model real-time embedded system and changes and evaluate system properties*" Edited by D. Sellier, www.empress-itea.org/deliverables/ D4.1_v1.0_Public_Version.pdf, March 2003.

[ER 86] J. P. Elloy, O. Roux "*Electre: A Language for Control Structuring in Real Time*" The Comput. Journal, 29(3): 229-234, 1986.

[Fau 02] S. Faucou « *Description et construction d'architectures opérationnelles validées temporellement* » thèse de l'Université de Nantes,2002.

[FBB 97] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, O. Shivers "*The Flux OSKit : A Substrate for Kernel and Language Research*" Symposium on Operating Systems Principles, 38–51, 1997.

[FDT 01] S. Faucou, A.-M. Deplanche, Y. Trinquet "*Operative architecture design and modelling for the validation of real-time applications*" Emerging Technologies and Factory Automation, 2, 649-652, Antibes, 2001.

[FGF 04] P. Farail, P.Gaufillet , J.-M. Farines , J.-L. Lambert , P. Dissaux , H. Hafidi , P. Michel , M. Filali , J.-P. Bodevaix , F. Vernadat , B. Berthomieu , P.O. Ribet "*The COTRE project : how to model and verify real-time architectures*"European congress on embedded Real-Time Software, Toulouse 2004.

[Fow 99] M. Fowler "*Refactoring: improving the design of existing programs*" Addison-Wesley, 1999.

[FP 02] D. Frankel and J.Parodi "*Using model-driven architecture(tm) to develop web services"* IONA Technologies PLC, White paper, 2002.

[FSA 05] J. Fredriksson, K. Sandstrom, M. Akerholm "*Optimizing resource usage in Component-Based Real-Time Systems*" Component-based software engineering, LNCS 3489, 49-65,Springer, St Louis (USA), 2005.

[FSL 02] J.-P. Fassino, J.-B. Stefani, J. Lawall, G. Muller "*Think : A Software Framework for Component-based Operating System Kernels*" In USENIX Annual Technical Conference, General Track, 73–86, 2002.

[FVC 05] O. Florescu, J. Voeten, H. Corporaal "*Property-preservation synthetsis for unified control and data-oriented models*" Forum on Design Language, FDl'05, Lausanne, Sept. 2005.

[GAM] K. Godary, I. Auge-Blum, A. Mignotte "*SDL and Timed Petri Nets versus UPPAAL for the validation of embedded architecture in automotive*" Forum on Design Languages, FDL'04, Lille , Sept. 2004

[Gar 99] M.K. Gardener "*probabilistic analysis and scheduling of critical soft real-time systems*", PhD Thesis, Dpt of CS, Univ of Illinois at Urbana Champaign, 1999.

[GC 02] E. Grolleau, A. Choquet-Geniet "*off-line computation of real-time schedules by means of petri nets*" journal of Discrete Event Dynamic Systems, 12, 311-333, 2002.

[GFB 03] J. Goosens, S. Funk, S. Baruah "*Priority-driven scheduling of periodic task systems on multiprocessors*" Real Time Systems, 25(2/3), 187-205, 2003.

[GFB 05] S. Gérard, J.-M. Favre, M. Blay-Fornarino « *L'ingéniérie dirigée par les modèles, réflexions de l'AS-MDA* » Génie Logiciel, 73, pp 2-7, Juin 2005.

[GGV 96] P. Goyal, X. Guo, H. Vin "*A Hierarchical CPU Scheduler for Multimedia Operating Systems*" In Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI), 107–121, 1996.

[GH 04] S. Graf, J. Hooman "*Correct Development of Embedded Systems*" European Workshop on Software Architecture: Languages, Styles, Models, Tools, and Applications (EWSA 2004), LNCS ,3047, 241-249, Springer-Verlag, St Andrews(Scotland), 2004.

[GH 05] Z. Gu, Z. He "*Real-Time scheduling techniques for implementation synthesis from component-based sotware models*" Component-based software engineering, LNCS 3489, 235-250,Springer, St Louis (USA), 2005.

[GHJ 95] E. Gamma, R. Helm, R. Johnson, J. Vlissides "*Design Patterns: Elements of Reusable Object-Oriented Software*" Addison –Wesley,1995.

[GJ 79] M.R. Garey, D. S. Johnson "*Computers and intractability : a guide to the theory of NP-Completeness*" Ed Freeman, New-York, 1979.

[GL 95] D. W. Gillies and J. W.-S. Liu. "*Scheduling tasks with and/or precedence constraints*". SIAM Journal on Computing, 24(4):797–810, 1995.

[GLM 02] H. Garavel, F. Lang, R. Mateescu "*An overview of CADP 2001*" European Association for Software Science and Technology (EASST) Newsletter, 4, 13-24, 2002.

[God 04] K. Godary « *Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile* » rapport de thèse, INSA de Lyon, Nov. 2004.

[Gom 93] H. Gomaa "*Software Design Methods for Concurrent and Real-Time Systems*" Addison-Wesley Longman Publishing Co., 1993.

[GOO 04] S. Graf, I. Ober, I. Ober "*Model-checking UML models via a mapping to communicating extended timed automata*" Proceedings of SPIN'04, Barcelona (Spain), Avril, 2004.

[Gra 02] S. Graf "*Expression of time and duration constraints in SDL*", 3rd SAM Workshop on SDL and MSC, 2002.

[GSB 99] E. Gabber, C. Small, J. Bruno, J. Brustoloni, A. Silberschatz "*The Pebble Component-Based Operating System"* USENIX Annual Technical Conference, 267–282, 1999.

[GT 03] S. Gérard, F. Terrier « Des méthodes UML pour la modélisation d'applications temps reel » Ecole d'été temps reel, ETR'03, 19-34, Toulouse, 2003.

[GT 05] S. Gérard, F. Terrier « L'ingéniérie des modèles pour les systèmes temps réel » Génie Logiciel, 73, pp 27-32, Juin 2005.

[GW 94] P. Godefroid, P. Wolper "*A partial approach to model-checking*" Information and Computation, 110(2), 305-326, 1994.

[GW 96] R.J. van Glabbeek, W. Weijland *"Branching Time and Abstraction in Bisimulation Semantics*" Journal of the ACM, 43(3), 555-600,1996.

[Hal 91] W. Halang "*PEARL 91*"Workshop über Realzeitsysteme, 295, 1991.

[HB 03] J.L.Houberdon, J.P.Babau "*MDA for embedded systems dedicated to process control*" workshop SIVOEES, San-Francisco, 2003.

[HCR 91] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud "*The synchronous dataflow programming language Lustre*" proceedings of the IEEE, 79(9), 1305-1320, Sept. 91.

[He 02] A. Hessel "*Timing analysis of an SDL subset in Uppaal*" Master thesis, Department of Information Technology, Uppsala University, 2002.

[Hen 96] T. Henzinger "*The theory of hybrid automata*" IEEE Symposim on Logic in Computer Science (LICS'96), 278--292. IEEE Computer Society Press, 1996.

[HF 98] J. Helander, A. Forin "*MMLite : a highly componentized system architecture*" ACM SIGOPS European workshop on Support for Composing Distributed Applications, 96–103. ACM-Press, 1998.

[HGT 03] P.T. Hieu, S. Gérard, D. Lugato, F. Terrier, "*Scheduling Validation for UML-modeled real-time systems*", Euromicro Conference on Real-Time Systems,.Porto, WIP, Portugal, 2003.

[HHK 01] T. Henzinger, B. Horowitz, C. Kirsch "*Giotto : A time-triggered language for embedded programming*" International Workshop on Embedded Software (EMSOFT), LNCS, 2211, 166-184, Springer-Verlag, 2001.

[HKL 91] M.G. Harbour, M.H. Klein, J. Lehockzy "*Fixed Priority Scheduling of periodic tasks with varying execution priority*" Real-Time System Symposium, 116-128, 1991.

[HLM 99] Z. Huang, A. Legait, M. Maranzana, E. Niel, J. -J. Schwarz, J. Skubich "About Techniques for the Verification of the Behaviour of Real-Time Multitasking Components" IFAC'99 World Congress, Beijing 1999.

[HMK 96] R. Henke, A. Mitschele-Thiel, H. König, P. Langendörfer "*Automated Derivation of Efficient Implementations from SDL Specifications*" 1996.

[Hol 96] G. Holzman "*On-The-Fly Model Checking*" ACM Computing Surveys, 28(4), Article No. 120 1996.

[HP 87] D. J. Hatley, I. A. Pirbhai "*Strategies for Real-Time System Specification*", Dorset House, New York, 1987

[IN 02] D. Isovic, C. Norstrom "*Components in real-time systems*" International Conference on RealTime Computing Systems and Applications (RTCSA'2002), 2002.

[ISO 95] ISO "*Quality of Service Framework*" Technical report, International Standards Organisation ISO/IEC JTC1/SC21/WG1 N9680, 1995.

[ITU 94] ITU-T "*Recommendation E.800 – Terms and definitions related to quality of service*" Tech. report ITU Telecommunication Standardization Sector, 1994.

[Java] Sun MicroSystem "*Java 2 Platform Standard Edition*" J2SE 5.0 API Specification, java.sun.com

[JM 86] Jahanian, A. K.-L. Mok,, "*Safety Analysis of Timing Properties in Real-Time Systems*", IEEE Transaction on Software Engineering, 12 (9), 890-904, 1986.

[Ka 82] C. Kaiser « *Exclusion mutuelle et ordonnancement par priorité* » Technique et Science Informatique, 1 (1), 59-68, 1982.

[KCM 00] F. Kon, R. Campbell, M. Mickunas, K. Nahrstedt, F. Ballesteros "*2K : A Distributed Operating System for Dynamic Heterogeneous Environments*" IEEE International Symposium on High Performance Distributed Computing, 2000.

[KDK 89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, C senft, R. Zailinger "*The MARS approach*" IEEE Micro, 9(1), 25-40, 1989.

[Kle 93] M. Klein & all "*A practionner's Handbook for Real-Time Analysis*", Kluwer Academic Publishers, 1993.

[KNA 02] G. Karsai, S. Neema, B. Abbott, D. Sharp "*A Modeling Language and Its Supporting Tools for Avionics System*s" in Proceedings of 21st Digital Avionics Systems Conference, 2002.

[Koy 90] R. Koymans "*Specifying real-time properties with metric temporal logic*" Real-Time Systems, 2 (4), 255–299, 1990.

[KNS 02] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston "*Automatic verification of real-time systems with discrete probability distributions*" Theoretical Computer Science, 282,101–150, 2002.

[KP 92] Y. Kesten, A. Pnueli "*Timed and Hybrid Statecharts and their Textual Representation*" Formal Techniques in RealTime and Fault-Tolerant Systems, 1992.

[LCC 75] R. Levin, E. Cohen, W. Corwin, F. Pollack, W. Wulf "*Policy/mechanism separation in HYDRA*" ACM Symposium on Operating Systems Principles (SOSP'75), 132–140, Austin (USA), 1975.

[LES 05] O. Layaïda, A. Erdem Özcan, J.-B. Stefani "*A Component-based Approach for MPSoC SW Design: Experience with OS Customization for H.264 Decoder*" *Workshop on Embedded Systems for Real-Time Multimedia, CODES+ISSS*, 2005.

[LGL 91] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire "*Programming Real-Time Applications with Signal*", 79(9), 1321-1336, Sept. 1991.

[LGT 98] A. Lanusse, S. Gérard, F. Terrier "*Real-Time Modeling with UML : The ACCORD approach*" UML98 : beyond the Notation98, LNCS 1618, Mulhouse,1998.

[LL 73] C.L. Liu, J.W. Layland "*Scheduling algorithms for multiprogramming in a hard real time environment*" Association for Computing Machinary, 20 (1), 46-61, 1973.

[LLN 87] J. W.-S. Liu, Kwei-Jay Lin, S. Natarajan "*Scheduling Real-Time, Periodic Jobs Using Imprecise Results*" proc. of the IEEE Real-Time Systems Symposium, 252-260,1987.

[LP 97] G. Larsen, P. Pettersson, and W.Y. Kim "*UPPAAL in a Nutshell*" Journal on Software Tools for Technology Transfer 1(1-2), 134-152. Springer-Verlag, 1997.

[LQV 01] L. Lavazza, G. Quaroni, M. Venturelli "*Combining UML and formal notations for modelling real-time systems*" European Software Engineering Conference, 1à-14, Vienne, 2001.

[LR 04] D. Lime and O. Roux "*A translation based method for the timed analysis of scheduling extended time Petri nets*" proceedings of the 25th IEEE International Real-Time Systems Symposium, 187-196, Lisbon, Portugal, 2004.

[LS 86] J. Lehoczky, L. Sha "*Performance of real-time bus scheduling algorithms*" ACM international conference on Computer Performance modelling, measurement and evaluation, 44-53, 1986.

[LSD 89] J. Lehoczky, L. Sha, Y Ding "*The rate monotonic scheduling algorithm : exact characterisation and average case behaviour*" Proceedings of the 10th IEEE Real-Time Systems Symposium, 166-171 1989.

[LSS 87] J. Lehoczky, L. Sha, J.K. STrosnider "*Enhanced aperiodic responsiveness in hard real time environments*" Proceedings of the 8th IEEE Real-Time Systems Symposium, 261-271,1987.

[LW 04] L. Thiele, R. Wilhelm "*Design for Timing Predictability*" journal of Real-Time Systems, 28(2/3), 157-177, 2004.

[LW 82] J.Y.T. Leung, J. Whitehead "*On the Complexity of Fixed-Priority Scheduling of Periodic Tasks*" Performance Evaluation, 2, 237-250, 1982.

[Mam 00] Z. Mammeri « *SDL, Modélisation de protocoles et systèmes réactifs* » Hermès Science Europe, 2000

[Man 74] Z. Manna "*Mathematical theory of computation*" Mc Graw publisher, 1974.

[McM 93] K. Mc Millan "*Symbolic Model Checking*" Kluwer academic publishers, 1993

[MDA] Object Management Group "*Model Driven Architecture*" guide v1.0.1. http://www.omg.org/mda, 2003.

[Mer 01] S. Merz "*Model Checking : a tutorial overview*" Modeling and Verification of Parallel processes, LNCS, 2067, 3-28, Springer-Verlag, 2001.

[Mer 74] P. Merlin "*A study of the recoverability of computing systems*" PhD thesis, Irvine university of California, TR48, 1974.

[MGD 01] J.L. Medina Pasaje, M. Gonzales Harbour, J.M. Drake" *MAST Real-Time View : A graphic UML Tool for Modeling Object-Oriented Real-Time Sytems* " IEEE Real-Time Systems Symposium, 245-256, London, 2001.

[Mil 80] R. Milner "*A calculus of Communicating Systems*" LNCS, 92, Springer-Velrag,1980.

[MKM 00] A. Muth, T. Kolloch, T. Maier-Komor, and G. Färber. " *An evaluation of code generation strategies targeting hardware for the rapid prototyping of SDL specifications* " IEEE Workshop on Rapid Systems Prototyping, 134-14, Paris, 2000.

[ML 05] G. Muller, J. Lawall "*The Bossa framework for scheduler development*" Ecole d'été temps réel, ETR'05, 23-30, Nancy, 2005.

[MSC 96] ITU-T "*Recommendation Z.120, Message Sequence Chart*" 1996.

[Muc 97] S. Muchnik "*Advanced Compiler Design Implementation*" Morgan Kaufmann Publisher, 1997.

[NBH 98] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, W. Chiu "*COYOTE : A System for Constructing Fine-Grain Configurable Communication Services*" ACM Transactions on Computer Systems, 16(4), 321–366, 1998.

[NET 04] Microsoft. Microsoft "*NET Passport*" www.passort.net, 2004.

[NT 93] L. Nigro, F. Tisato "*RTO++: a framework for building hard real-time systems*", *Journal on OO Programming,* 6(2), 35-47, 1993.

[OLK 00] R. van Ommering, F. van der Linden, J. Kramer, J. Magee "*The Koala component model for consumer electronics software*"IEEE Computer, 3(33), 78–85, 2000.

[OSGI 03] The Open Services gateway Initiative "*osgi service platform release 3*" Technical report, 2003.

[OSK] OSEK Group "*OSEK/VDX Time-Triggered Operating System*" site WEB www.osek-vdx.org/

[Par 72] D. Parnas "*On the criteria to be used in decomposing systems into modules*" Communications of the ACM, 15(12),1053–1058, 1972.

[Par 81] D. Park "*Concurrency and Automata on Infinite Sequences*" Theorical Computer Science, LNCS, 104, 167-183, Springer Verlag, 1981.

[Pec 99] PECOS Project. http ://www.pecos-project.org, 1999.

[Per 90] J.P. Perez « *Systèmes temps reel, methods de specification et de* conception » Dunod, Paris, 1990.

[Pet 99] P. Pettersson "*Modeling and Analysis of Real-Time Systems Using Timed Automata: Theory and Practice*" Ph.D. Thesis Uppsala University , 1999.

[Pnu 81] A. Pnueli "*The Temporal Semantics of Concurrent Programs*" Theoretical Computer Science, 13, 45–60, Elsevier Science, 1981.

[PST 97] C.A. Philips, C. Stein, E. Torng, J. Wein "*Optimal time-critical scheduling via resource augmentation*" 29th ACM Symposium on Theory of Computing, 140-149, 1997.

[QoS 04] Object Management Group "*UML profil for modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms*" ptc/04-09-01, 2004.

[QUE 97] QUEST: http://www.cs.uni-essen.de/Fachgebiete/SysMod/Forschung/QUEST/, 1997

[RBP 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen "*Object Oriented Modelling and Design*" Prentice-Hall, 1991.

[Rha] I-Logix, "Rhapsody in C++" reference guide 1999.

[Ric 03] P. Richard « *Analyse du temps de réponse des systèmes temps reel* » Ecole d'été temps réel , ETR'03, 241-262, Toulouse, 2003.

[RK 76] F. de Remer and H. Kron."*Programming-in-the-large versus programming-in-the-small*" IEEE Transactions on Software Engineering, 2(2), 1976.

[RMJ 04] V. Rusu, H. Marchand, T. Jéron "*Automatic verification and conformance testing for validating safey properties of reactive systems*" Ecole d'été temps reel, ETR'05 ,69-77, Nancy, 2005.

[Ros] IBM Rational "*Rose Real-Time*" demo tool.

[RRM 04] Tri-Pacific® "*RAPID RMA™: The Art of Modeling Real-Time Systems*" V5.3.3, www.tripac.com, 2004.

[RSR 01] P. Roop, A. Sowmya, S. Ramesh "*Forced simulation : A technique for automating component reuse in embedded systems*" ACM Transactions on Design Automation of Electronic Systems (TODAES), 6(4), 602 – 628, 2001

[RTD] PragmaDev "*Real-Time Developer Studio*" RTDS G3, www.pragmadev.com.

[SAH 01] J. Stankovic, T. Abdelzaher, Z. He, P. Nagarradi, Z. Yu "*Vest : Virginia embedded systems toolkit*", 390–402, 2001.

[SB 96] M. Spuri, G.C. Butazzo "*scheduling aperiodic tasks in dynamic priority systems*" Real-Time Systems, 10, 179-210, 1996.

[Sch 95] J.-J Schwarz « *Environnement graphique pour la conception des applications multitâches temps réel* » rapport d'habilitation à diriger les recherches, UCB Lyon1, 1995.

[SDL 96] ITU-T "*Recommendation Z.100, Specification and Design Language (SDL)"* 1996.

[SDR 03] SDL-RT V2.1 "*Specification & description language real-time*" www.sdl-rt.org/, 2003.

[SG 94] B. Selic, G. Gullekson, P. Ward "*Real-Time Object-Oriented Modeling*" Wiley. 1994.

[Sha 04] L. Sha & All "*Real-Time Scheduling Theory : a Historical Perspective*" Real-Time Systems, 28(2/3), 101-155, 2004.

[Sha 92] A. Shaw "*Communicating Real-Time State Machines*" EEE Transactions on Software Engineering, 18(9), 805 - 816 , Sept. 1992.

[SLC 91] W.-K. Shih , J. W. S. Liu , J.-Y. Chung, "*Algorithms for scheduling imprecise computations with timing constraints*", SIAM Journal on Computing, 20(3), 537-552, 1991.

[SLS 88] B. Sprunt, J. Lehoczky, L. Sha "*Exploiting Unused periodic Time For Aperiodic Service Using The Extended Priority Exchange Algorithm*" Proceedings of the 9th IEEE Real-Time Systems Symposium, 251-258, 1988.

[SOR93] N. Shankar, S. Owre, J.M. Rushby "*The PVS proof checker: A reference manual*" Technical report, Comp. Sci.,Laboratory, SRI International, 1993.

[SPF 98] M. Saksena, A. Ptak, P. Freedman, and P. Rodziewicz. Schedulability "*Analysis for Automated Implementations of Real-Time Object-Oriented Models*" Proceedings, IEEE Real-Time Systems Symposium, 1998.

[SPT 01] G. Sunyé, D. Pollet, Y. Le Traon, J.-M. Jézéquel "*Refactoring UML Models*" Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, LCNS, 2185 , 134 – 148, 2001.

[SPT 03] Object Management Group "*UML® Profile for Schedulability, Performance, and Time*", version 1.0, www.omg.org/cgi-bin/doc?formal/03-09-01, sept 2003.

[SR 04] J. Stankovic, R. Rajkumar "*Real Time Operating Systems*" journal of Real-Time Systems, 28(2/3), 237-253, 2004.

[SRL 90] L. Sha, R. Rajkumar, J. Lehockzy "*Priority Inheritance Protocols: an Approach to Real Time Synchronisation*" IEEE Transaction Computers, 39(9), 1175-1185, 1990.

[SRS 03] C. Schurgers, V. Raghunathan, M. B. Srivastava "*Power Management for Energy-Aware Communication Systems*" ACM Transactions on Embedded Computing Systems, 2(3), 431–447, 2003.

[SS 94] M. Spuri, J.A. Stankovic "*How to Integrate Precedence Constraints and Shared Resources in Real-Time Scheduling*" IEEE Transactions on Computers, 43 (12), 1407-1412, 1994.

[SSD 97] S. Spitz, F. Slomka, M. Dörfel "*SDL* An Annotated Specification Language for Engineering Multimedia Communication Systems*" Workshop on High Speed Networks, Stuttgart, 1997.

[SSL 89] B.Sprunt, L. Sha, J. Lehoczky "*Aperiodic Task Scheduling for Hard Real Time Systems"* Real Time Systems, 1, 27-60, 1989.

[Sta 01] J. Stankovic "*VEST : aToolset for Constructing and Analyzing Component Based Embedded Systems*" LNCS, 2211, 390-402, 2001.

[Sta 88] J.A. Stankovic "*Misconceptions About Real-Time Computing: A Serious Problem For Next Generation Systems* ", IEEE Computer Magazine, 21 (10), 10-19, 1988.

[Str 95] H. Streich, "*TaskPair-scheduling: An Approach for Dynamic Real-time Systems*", Int. J. Mini and Microcomputers, 7(2), 77-83, 1995.

[SZ 92] K. Schwan, H. Zhou "*Dynamic Scheduling of Hard Real Time Tasks and Real Time Threads*" IEEE Transactions on Software Engineering, 18 (8), 738-748, 1992.

[Szy 98] C. Szyperski "*Component Software : Beyond Object-Oriented Programming*" ACM Press and Addison Wesley, 1998.

[Tau] Telelogic « *Telelogic Tau SDL suite* », http://www.telelogic.com

[TC 94] K. Tindell, J. Clark "*Holistic schedulability analysis for distributed hard real-time systems*" Euromicro Journal , 40, 117-134, 1994.

[TFB 96] F.Terrier, G. Fouquier, D. Bras, L. Rioux, P. Vanuxeem, A. Lanusse "*A Real-Time Object Model*" TOOLS EUROPE '96, 127-141, 1996.

[THW 94] K. Tindell, H. Hansson, A. Wellings "*Analysis real-time communications : controller area network (CAN)*" 15th IEEE Real-Time Systems Symposium, 259-265, 1994.

[Tou 02] J.-C. Tournier « *Prototypage à l'aide du langage SDL d'un système de robotique mobile*», rapport de projet de fin d'étude, CITI / ESISAR-Valence, 2002.

[Tou 05] J.-C. Tournier « *Qinna : une architecture à base de composants pour la gestion de la qualité de service dans les systèmes embarqués mobiles* » thèse de l'INSA de Lyon, Juillet 2005.

[Tri 05] Yvon Trinquet "*Les systèmes d'exploitation temps reel*" Ecole d'été temps réel , ETR'05, 123-134, Nancy, Sept. 2005.

[TT 92] K. Takashio, M. Tokoro "*DROL: an object-oriented programming language for distributed real-time systems*" *OOPSLA '92, ACM Sigplan*,276-294, 1992.

[TTT 03] TTTech Computertechnik AG. "*Time-Triggered Protocol TTP/C, High-Level Specification* ", Time-Triggered Technology Vienna, Austria, 2003.

[UML2] Object Management Group "*Unified Modeling Language Specification*" v.2.0, June 2003.

[UVH 01] D. Urting, S. Van Baelen, T. Holvoet, Y. Berbers "*Embedded Software Development : Components and Contracts*" International Conference Parallel and Distributed Computing and Systems, IASTED, 685–690, 2001.

[VAM 96] F. Vernadat, P. Azéma, F. Michel "*Covering Step Graph*" Conference on Application and Theory of Petri Nets, LNCS 1091, 516-535, Osaka, Springer-Verlag, 1996.

[VWJ 96] K. Verschaeve, B. Wydaeghe, V. Jonckers, L. Cuypers "*Translating OMT* to SDL, Coupling Object-Oriented Analysis and Design with Formal Description Techniques*" IFIP conference Method Engineering 96, Atlanta, août 1996.

[Wam 03] D.Wampler "*The role of aspect-oriented programming in omg's model-driven architecture*" Aspect Programming Inc, 2003.

[WF 00] S. Wang, G. Faerber "*A new algorithm to prove the schedulability of real-time systems*" Control engineering practice, 8, 689-695, 2000.

[WH 01] S. Wang, W. Halang "*A heuristic allocation method for multiprocessor systems*" conference on new technologies for computer control, 283-288,Hon Kong (China), Nov. 2001.

[WH 03] L. Waszniowski, and Z. Hanzálek, "*Analysis of Real Time Operating System Based Applic*ations" First International Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS 2003, LNCS Springer, 2003.

[WK98] J. Warmer, A. Kleppe "T*he Object Constraint Language : Precise Modeling with UML*" Addison-Wesley, 1998.

[WT 04] S. Wang, G. Tsai "*Specification and timing analysis of real-time systems*" Real-Time Systems, 28(1), 69-90, 2004.

[XP 90] I. Xu , D. Parnas "*Scheduling Processes with Release Times, Deadlines, Precedence and Exclusion Relations*", IEEE Transactions on Software Engineering, 16(3), 360-369, 1990.

[YL 01] S.N. Yeung, J.P. Lehoczy "*End-to-end delay analysis for Real-Time Networks*" IEEE Real-Time Systems Symposium, 299-309, 2001.

[WA 05] T. Watteyne, I. Augé-Blum "*Proposition of a Hard Real-Time MAC Protocol for Wireless Sensor Networks"*13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, *mascots*, 533-536, sept. 2005.