Vérification des applications temps réel

Françoise Simonot-Lion, Ye-Qiong Song, Bernard Berthomieu, François Vernadat

Vérification des applications temps réel

Françoise Simonot-Lion, YeQiong Song,
Bernard Berthomieu, François Vernadat

Mots-clés : vérification, temps réel, garantie déterministe, garantie probabiliste

Résumé : ce chapitre présente les moyens usuellement disponibles pour vérifier qu'un système temps réel respecte les propriétés qui lui sont imposées. Ceci recouvre des techniques de vérification qui reposent sur des modèles mathématiquement analysables de manière exhaustive ou partielle. Les garanties exigées quant au respect des contraintes pouvant être de nature déterministe ou probabiliste, nous proposons des techniques pour ces deux types d'objectifs.

1 Introduction

Les propriétés des systèmes temps réel sont imposées par les missions qui leur sont demandées et par la dynamique des systèmes physiques sous leur contrôle (que ce soit dans le contexte d'une ligne de production ou d'un système embarqué). Nous nous intéressons, dans ce chapitre, aux méthodes qui garantissent qu'un tel système est correct. Ce terme signifie qu'il doit assurer la mission demandée et respecter toutes les propriétés spécifiées. Prouver que ces systèmes respectent les propriétés qui leurs sont imposées passe, entre autres, par deux activités fondamentales.

La première relève de la conception d'une solution qui garantisse les propriétés "fonctionnelles". Il s'agit, dans ce cas, d'analyser les fonctions du système, leurs interactions ainsi que l'architecture logicielle qui les implémente.

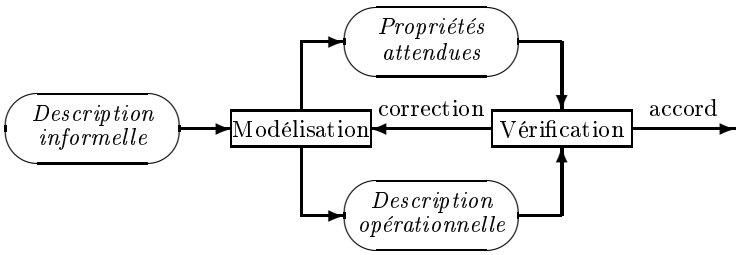

FIG. 1.1 – Vérification d'un système par model-checking

La figure 1.1 présente les principales étapes de la vérification. Ce type d'approches nécessite trois ingrédients :

– une description opérationnelle du système explicitant comment fonctionne le système ;

– un langage de spécification permettant d'exprimer les propriétés du système que l'on souhaite vérifier (cf section 2.3, les logiques temporelles comme outil de spécification) ;

– une procédure de décision permettant de contrôler la conformité entre la description opérationnelle du système et sa spécification, c'est-à-dire une procédure qui permet de vérifier que le système satisfait effectivement les propriétés que l'on attend de lui. Cette phase de vérification du modèle est souvent appelée "contrôle de modèles" ou "*model-checking*" (cf section 3.1, les principes algorithmiques du *model-checking*).

La deuxième activité consiste à prendre en compte l'implantation des logiciels sur une architecture matérielle, à savoir sur un ou plusieurs calculateurs, chacun géré par un système d'exploitation, et communiquant, le cas échéant, via une architecture de communication sous forme de ré-

seaux, munis de protocoles, et interconnectés par des passerelles. Le résultat de cette activité sera nommé par la suite "architecture opérationnelle". Les propriétés qui doivent être vérifiées lors de cette activité sont des propriétés usuellement appelées "non fonctionnelles". Celles-ci dépendent, en particulier, des performances des matériels support (puissance de processeurs, débit de réseaux, etc.) et des temps induits par les stratégies de partage de ressources (protocoles d'accès au médium, ordonnanceurs locaux). La vérification de propriétés non fonctionnelles revient globalement à évaluer deux types de caractéristiques : les dates d'occurrences de certains événements et les besoins en termes de ressources. Les caractéristiques relevant de l'un ou l'autre point peuvent apparaître dans l'expression de contraintes à respecter (contraintes temporelles, contraintes d'occupation mémoire, contraintes de consommation d'énergie, ...) ou, le cas échéant, comme critères à optimiser (minimiser un temps de réponse, une bande passante, l'énergie consommée, ...). De plus, le comportement de tout système temps réel est étroitement dépendant de son environnement. Aussi, les propriétés en général devront-elles être prouvées en prenant en compte l'impact de cet environnement sur le système.

Enfin, on peut aborder le problème de la vérification de propriétés pour des systèmes temps réel de deux manières qui, d'ailleurs peuvent s'avérer complémentaires dans certains cas :

– soit on conçoit un système de telle manière que les propriétés soient obtenues par construction ; l'application de méthodes formelles qui partant de la spécification des propriétés attendues et opérant par raffinement et/ou dérivation construit automatiquement la solution ; dans le domaine du temps réel, on peut citer, notamment, la construction d'un ordonnancement de tâches faisable, la construction d'une messagerie (configuration/ordonnancement de trames transmises sur un réseau) faisable ou la synthèse de contrôleur temps réel ;

– soit, dans le cas de systèmes complexes pour lesquels de telles méthodes ne sont pas applicables au système dans sa globalité ou dont le développement est partagé par plusieurs acteurs, on vérifie à chaque étape de la conception que la solution en cours d'élaboration vérifiera les propriétés ; là encore, suivant la place dans le cycle de développement du système où la vérification est opérée, on distinguera : *la vérification a priori*, c'est-à-dire avant la réalisation du système ; ce type de vérification implique de construire un modèle de ces systèmes qui soit pertinent pour les propriétés à vérifier ; *le test* qui se fait sur tout ou partie du système avant la mise en service de celui-ci ; et enfin, *la vérification* qui peut être faite *en ligne* pour garantir lors de l'exploitation et de la vie du système que certaines propriétés sont toujours préservées.

Dans ce chapitre, nous aborderons principalement le problème de la vérification de propriétés qui soient spécifiques aux systèmes temps réel et ce par construction de modèles de ces systèmes puis méthodes d'études de ces modèles. Les techniques proposées concerneront, alors, majoritairement, la vérification a priori. De plus, deux méthodes peuvent être déployées : l'une fondée sur une analyse mathématique exhaustive, l'autre sur une analyse partielle. L'idéal est d'opérer la vérification par analyse mathématique du modèle du système. Malheureusement, la plupart des techniques de ce type (voir le chapitre ££ Automates et vérification££ de la section 1.9 ££ Algorithmique et programmation££) sont difficilement applicables pour des modèles complexes (la complexité d'un modèle peut venir d'une représentation du système à un fin niveau de granularité ou de la complexité intrinsèque du système lui-même). Dans ce cas, la vérification peut être faite par simulation, c'est-à-dire, exécution du modèle. Si le problème de l'analyse mathématique est celui d'une précision du modèle qui soit pertinente pour la vérification à réaliser, celui de la simulation tient dans la difficulté d'exécuter le modèle sur une exhaustivité des scénarios. Ces deux méthodes sont en fait complémentaires.

2 Propriétés et garanties exigées

Avant de décrire les techniques qu'il est possible de mettre en œuvre pour opérer des vérifications dans les systèmes temps réel, nous introduisons deux notions importantes. La première recouvre l'expression des propriétés dans le contexte temps réel tandis que la deuxième concerne le type de garantie exigée quant au respect de ces propriétés.

2.1 Expression des propriétés

Dans le contexte des systèmes temps réel, une grande partie des propriétés s'exprime sous la forme de contraintes appliquées aux dates d'occurrences d'événements significatifs. Ci-dessous, nous donnons l'expression de quelques contraintes élémentaires.

Contraintes sur des temps de réponse du système Elles sont utilisées pour spécifier la capacité exigée d'un système à répondre aux sollicitations tant internes qu'externes. Ce peut être la promptitude du système à réagir à une situation détectée, l'aptitude de l'architecture logicielle implantant une loi de commande à fournir une consigne à un actionneur dans des temps raisonnables (par exemple, l'actionnement de la cré-

maillère entraînant l'axe de direction d'un véhicule en fonction de l'angle et du couple volant fourni par le conducteur, ...). Concrètement, il s'agit d'exprimer le fait que la durée séparant les occurrences de même rang i de deux événements (dénommés, généralement, stimulus S et réponse R) est inférieure à un délai donné δ_{sup} et supérieure à δ_{inf}. Notons, qu'usuellement cette propriété considère $\delta_{inf} = 0$.

$$\delta_{inf} < date(R_i) - date(S_i) < \delta_{sup}$$

Contraintes sur la régularité d'occurrence d'un événement Dans ce cas, il s'agit de borner la variation de durée qui sépare les occurrences successives, i, $i+1$, ... d'un même événement périodique. Par exemple, l'existence d'une gigue, c'est-à-dire d'une variation de cette durée sur les dates d'acquisition des grandeurs significatives d'un système physique peut entraîner une incohérence pour une loi de commande qui reposerait sur des reconstructions de l'évolution de ces grandeurs par calculs approchés de leur dérivée. Dans un autre domaine, la "non régularité" de diffusion d'une image vidéo peut restituer un film avec une piétre qualité. On considère, pour exprimer une telle contrainte que E est l'événement considéré, T la période spécifiée d'occurrence de cet événement et une valeur ε quantifiant la tolérance sur l'irrégularité d'occurrence de E de la manière suivante :

$$T - \varepsilon < date(E_{i+1}) - date(E_i) < T + \varepsilon$$

Contraintes sur la simultanéité d'occurrences d'événements Ce type de contraintes s'applique, par exemple, à toute application devant diagnostiquer l'état d'un système physique à partir d'un ensemble de mesures sur ce système ou de stimulus venant de ce système ; la qualité du diagnostic est largement dépendante de la durée de l'intervalle pendant lequel les mesures ont été réalisées sur le système physique. Un autre cadre concerne les contraintes de synchronisation entre la production de flux audio et vidéo. Une contrainte de simultanéité s'exprime sur un ensemble d'événements E pour imposer que les occurrences des événements de l'ensemble se produisent dans un intervalle borné par ε ; elle peut se présenter selon deux formes : l'intervalle doit contenir une occurrence au moins de chaque événement (simultanéité faible, équation 1.1) ou l'intervalle doit contenir les occurrences de même rang de tous les événements (simultanéité forte, équation 1.2) :

$$\forall e, f \in E, \exists i, j, \left| date(e^i) - date(f^j) \right| < \varepsilon \quad (1.1)$$

$$\forall e, f \in E, \forall i, \left| date(e^i) - date(f^i) \right| < \varepsilon \quad (1.2)$$

2.2 Types de garanties de propriétés

Les exigences imposées à un système temps réel relèvent de plusieurs types de garanties sur le respect des propriétés spécifiées.

Une garantie, dite déterministe, nécessite que la contrainte soit vérifiée à tout instant. Les techniques qui permettent d'obtenir un tel type de garantie sur un système reposent généralement sur une identification du pire cas et peuvent conduire à un surdimmensionnement du système dès que celui-ci devient complexe.

Une garantie en moyenne ou probabiliste, est un autre moyen d'expression d'une exigence. Il s'agit de spécifier, que sur une suite d'instances (par exemple les occurrences successives des couples d'événements stimulus et réponse), un pourcentage de celles-ci doit respecter la contrainte donnée. On utilise généralement ce type d'expression pour des contraintes de temps dites souples ou faibles. Notons que dans nombre d'applications temps réel, respecter ce type d'exigences ne garantit pas la qualité du système. Par exemple, garantir qu'un temps de réponse est inférieur à une borne dans 90% des cas ne spécifie pas la répartition des 10% de cas où cette contrainte n'est pas vérifiée. Sur mille instances, ce peut être une tous les cent ou les cent premières ; si le système temps réel implante une application de contrôle-commande, l'impact sur l'environnement contrôlé n'est pas le même et le système peut, même, ne plus assurer sa mission de façon sûre.

Afin de palier l'imprécision que peut révéler une garantie en moyenne, il est possible d'exiger des garanties déterministes qui tolèrent le non-respect d'une contrainte par toutes les instances à la condition que la distribution des échecs relève d'un modèle spécifié formellement. On parle alors de garantie déterministe $(m,k) - firm$. Supposons que la propriété s'exprime par une contrainte bornant supérieurement un temps de réponse entre un événement stimulus et un événement réponse ; une garantie $(m,k) - firm$ signifie que, à tout instant, pour les k dernières occurrences successives de l'événement stimulus, m, au moins ont conduit à une occurrence de l'événement réponse respectant la contrainte.

2.3 Formalismes d'expression des propriétés

Après avoir vu ces différents exemples de propriétés, nous introduisons maintenant quelques formalismes, basés sur la logique temporelle, spécifiquement dédiés à l'expression et à la vérification de propriétés des systèmes dont l'état évolue dynamiquement avec le temps. On peut

s'intéresser à des propriétés purement qualitatives telles que l'absence de blocage, la potentialité/inévitabilité d'un événement, etc. On classe souvent les propriétés à satisfaire en deux types complémentaires : les propriétés de "sûreté" exprimant le fait que "Rien de mauvais ne peut arriver" et les propriétés de "vivacité" exprimant le fait que "Quelque chose de bon va arriver". On peut aussi vouloir quantifier la propriété (temps de réponse, temps d'utilisation d'une ressource, etc). Le bref panorama qui suit introduit les logiques temporelles "qualitatives" et une "logique temporelle temporisée" permettant quant à elle d'adresser des propriétés de nature plus quantitative. Pour cette section ainsi que pour la section 3.1, nous invitons le lecteur intéressé à se reporter à (Arnold 1992), (Diaz 2003), (Schnoebelen 1999).

Aspects qualitatifs Les logiques temporelles sont des extensions du calcul propositionnel obtenues en ajoutant de nouveaux connecteurs – des opérateurs temporels – permettant de décrire l'évolution dynamique d'un système. Cette dynamique peut être perçue soit comme un ensemble d'exécutions (approche linéaire), soit comme l'arborescence des exécutions possibles (approche arborescente). A chacune de ces perceptions est associée une logique que nous décrivons maintenant. Par la suite, on considère un ensemble de variables propositionnelles $\mathcal{P}$ permettant de capturer l'état du système.

La logique temporelle linéaire LTL exprime des propriétés sur des séquences d'états que le système atteint au cours d'une de ses exécutions. Une propriété LTL est satisfaite par un système si toute exécution du système la satisfait. Parmi les opérateurs temporels de LTL, on trouve $\Box$ permettant d'exprimer l'invariance et $\Diamond$ permettant d'exprimer la possibilité. De facon informelle, une séquence satisfait $\Box\ \phi$ si la propriété ϕ est vraie sur tous les états de la séquence tandis qu'on notera $\Diamond\ \phi$ pour spécifier que la séquence passe par un état où la propriété ϕ est vraie. L'expressivité de LTL tient à la possibilité d'imbriquer les différents opérateurs temporels, comme le montrent les quelques exemples suivants :

Exclusion mutuelle : $\Box\ \neg(\mathrm{Sc}_1 \wedge \mathrm{Sc}_2)$
"*On n'atteint jamais un état où deux processus sont en section critique.*"

Vivacité : $\Box$ (Requete $\Rightarrow \Diamond$ Reponse)
"*Toute requête sera suivie au bout d'un temps fini par une réponse*"

Equité : ($\Box \Diamond$ Demande) $\Rightarrow$ ($\Box \Diamond$ Accord)
"*Toute requête infiniment posée recevra une infinité d'accords*"

La logique temporelle arborescente CTL exprime des propriétés sur les exécutions possibles d'un système, matérialisées par une arboresence de ses états successifs. Une exécution particulière du système correspond donc à une branche (éventuellement infinie) de cet arbre. CTL utilise par exemple les quatre opérateurs suivants dont la sémantique intuitive est illustrée par la figure 1.2.

AF ϕ *Toujours finalement* ϕ
EF ϕ *Il est possible que finalement* ϕ
AG ϕ *Toujours partout* ϕ
EG ϕ *Il est possible que partout* ϕ

Comme pour LTL, l'expressivité de CTL tient à la possibilité d'imbriquer les différents opérateurs temporels, comme le montrent ces quelques exemples :

Exclusion mutuelle : $AG\ (\neg(\mathrm{Sc}_1 \wedge \mathrm{Sc}_2))$
"*On n'atteint jamais un état où deux processus sont en section critique.*"

Potentialité : AG (Requete $\Rightarrow EF$ Reponse)
"*Toute requête sera potentiellement suivie au bout d'un temps fini par une réponse*"

Inévitabilité : AG (Requete $\Rightarrow AF$ Reponse)
"*Toute requête sera inévitablement suivie au bout d'un temps fini par une réponse*"

CTL et LTL ont des expressivités complémentaires : CTL comme LTL permet d'exprimer des propriétés de sûreté et de vivacité. CTL permet aussi d'exprimer des propriétés de potentialité. Par contre, les propriétés d'équité ne sont pas exprimables en CTL. Dans les deux cas, ces logiques ne permettent d'exprimer que des notions qualitatives : ainsi on peut spécifier qu'*une réponse arrive en un temps fini* mais on ne peut pas spécifier que cette réponse doit arriver en moins de k unités de temps. Pour permettre ce type de spécification, les logiques temporelles classiques (LTL ou CTL) ont été étendues afin de pouvoir "quantifier" le temps : on parle alors de logiques temporelles temporisées. On retrouve des logiques temporisées linéaires et arborescentes. A titre d'exemple nous introduisons TCTL, l'extension temporisée de la logique CTL.

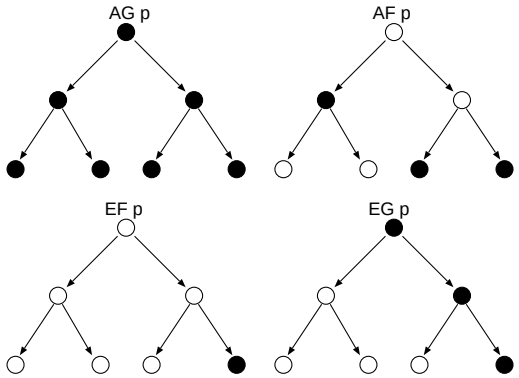

FIG. 1.2 – Exemples de satisfaction des modalités de CTL

Logique temporelle temporisée TCTL TCTL enrichit les opérateurs temporels de CTL en prenant en compte des informations quantitatives sur l'écoulement du temps. Ainsi les formules de TCTL font appel à un symbole relationnel $\sim \in \{<, \leq, =, >, \geq\}$ et un rationnel θ qui va permettre de quantifier l'écoulement du temps. Nous ne rentrerons pas plus en détail dans l'exposé de TCTL et renvoyons à (Schnoebelen, 1999) pour plus d'informations. Nous concluons cette section en donnons quelques exemples de propriétés TCTL.

Séjour borné en section critique :
AG (Sc $\Rightarrow$ $AF_{\leq K} \neg$ Sc)
"*Un processus ne reste pas plus de k unités de temps en section critique*"

Réactivité bornée :
AG (*Probleme* $\Rightarrow \neg\, AG_{\leq 5}$ *Alarme*) :
"*Lorsqu'un problème survient, l'alarme se déclenche et se prolonge au moins 5 unités de temps*"

Inévitabilité bornée :
AG (Requete $\Rightarrow AF_{\leq k}$ Reponse)
"*Toute requête sera inévitablement suivie en moins de k unités de temps par une réponse*"

Latence minimale :
AG (Requete $\wedge \neg\, EF_{\leq k}$ Reponse)
"*Une requête ne peut pas recevoir de réponse en moins de k unités de temps*"

En combinant les deux dernières formules, on peut exprimer des contraintes temporelles appliquées aux dates d'occurrences d'événements significatifs telles que celles présentées à la section précédente. Ainsi une contrainte du type

$$\delta_{inf} < date(R_i) - date(S_i) < \delta_{sup}$$

pourra s'exprimer par la formule suivante :
$[AG\ (\ R_i \wedge \neg\, EF_{\leq \delta_{inf}}\ S_i)] \wedge$
$[AG\ (R_i \Rightarrow\ AF_{\leq \delta_{sup}}\ S_i)]$

3 Vérification à garantie déterministe

L'objectif de cette section est de présenter quelques techniques permettant de vérifier qu'une garantie déterministe de propriété est assurée par un système. En particulier, nous proposons dans un premier temps des techniques qui s'appliquent de manière générique à des modèles exprimés dans un formalisme donné. Il s'agit notamment des méthodes d'analyse de modèles (ou *model-checking*) de systèmes discrets temporisés. Dans un deuxième temps, nous prenons en compte les objets classiquement manipulés au sein d'un système temps réel, à savoir les ressources, les tâches et les messages/paquets, et nous introduisons quelques méthodes propres à la vérification de systèmes décrits sous forme de ce type d'objets.

3.1 Analyse de modèles – *Model-checking*

Nous donnons maintenant les grands principes du model-checking de formules de logiques temporelles. Le lecteur intéressé se reportera avec profit à (Arnold, 1992), (Diaz, 2003), (Schnoebelen, 1999).

Une propriété exprimée en LTL (cf section 2.3) est satisfaite par un système si toute exécution de celui-ci la satisfait. Pour s'assurer de cette propriéte, on vérifie en fait qu'aucune exécution du système ne satisfait la négation de la formule à vérifier. Pour ce faire, on associe à la négation de la propriété un automate de Bucchi qui la caractérise et on effectue un produit synchrone entre le comportement du système et cet automate. On s'assure ensuite que les conditions d'acceptation données par l'automate de Bucchi ne sont pas satisfaites. Dans la négative, on dispose d'une séquence satisfaisant la négation de la propriété que l'on cherchait à vérifier, donc d'un contre-exemple. On peut procéder à la volée –

construire en même temps le produit synchrone et le graphe de comportement – et stopper cette construction dès qu'un contre-exemple a été détecté. Lorsque la propriété à vérifier est satisfaite, il faut bien sûr construire complètement le produit synchrone.

La vérification d'une propriété φ de CTL (cf section 2.3) consiste à calculer sa "théorie" notée $TH(\varphi)$: i.e., l'ensemble des états satisfaisant la formule à vérifier. Une propriété φ est satisfaite par un système si tous les états de celui-ci la vérifient : $TH(\varphi) = S$. Opérationnellement, le contrôle de modèles revient donc à définir une application $TH : CTL \mapsto \mathcal{P}(S)$ où $TH(f) = \{s \in S : M, s \models f\}$.
Dans le cadre propositionnel, l'application TH se définit par induction sur la structure de la formule f : $TH(p) = \{s \in S : p \in \mathcal{P}\ et\ s \models p\}$, $TH(\neg F) = S \setminus TH(F)$ et $TH(F \wedge G) = TH(F) \bigcap TH(G)$.
Pour les modalités de CTL (EF, AF, ...), le problème revient à trouver des "applications" X et Y de telle sorte que : $TH(EF\ G) = X(TH(G))$ et $TH(AF\ G) = Y(TH(G))$. A titre d'exemple, nous présentons la caractérisation opérationnelle de l'opérateur EF. Celle-ci est obtenue comme la limite d'une suite monotone à valeurs dans $\mathcal{P}(S)$. Cette suite est obtenue à l'aide du transformateur de prédicats Pre défini ainsi :

$Pre(U) = \{t \in S : t \rightarrow s \text{ et } s \in U\}$ et
$TH(EF\ G) = \bigcup_{n \geq 0} X_n$ où
$X_0 = TH(F)$ et $X_k = Pre(X_{k-1}) \bigcup X_{k-1}$

Pre est une fonction croissante, de sorte que la suite (X_n) est elle-même croissante dans un treillis complet (à valeurs dans un ensemble de parties : $\mathcal{P}(S)$) et donc convergente.

Limites opérationnelles du model-checking
Il faut noter que l'analyse de modèles n'agit pas directement sur la description formelle du système (automates, réseaux de Petri, ...) mais sur le graphe de comportement qui lui est associé. Un premier problème réside dans la décidabilité de la finitude du graphe de comportement. Karp et Miller (Karp, 1969) ont donné une réponse positive lorsque la description du système est donnée sous forme de réseaux de Petri de base ; le résultat précédent a été étendu par (Finkel, 2001) pour les systèmes de transitions bien formés, mais le problème est indécidable dans le cas général (Brand, 1983). Lorsque le graphe est fini, un deuxième problème est de lutter contre l'explosion combinatoire. De nombreux travaux ont été menés pour faire face à ce problème. On peut citer les techniques de compression de graphes par les BDD (*Binary Decision Diagram*) (Burch, 1990) qui gérent l'explosion en permettant le stockage de graphes de grande taille et les méthodes (voir par exemple (Diaz, 2003)) qui évitent cette explosion en construisant un graphe de comportement réduit : symétries, contrôle de modèles structurel, ou les techniques "ordre partiel", qui tentent d'éviter la part d'explosion combinatoire due à la représentation du parallélisme par l'entrelacement d'actions.

Cas des descriptions temporisées
Construire le graphe de comportement d'une description temporisée – qu'il s'agisse de réseaux de Petri ou d'automates – est en général impossible : les transitions pouvant être tirées à tout instant dans leur domaine de tir, les états admettent en général une infinité de successeurs. Pour se ramener à un graphe de comportement fini, des techniques d'abstractions sont proposées permettant d'obtenir une représentation finie du comportement du système équivalent – pour une classe de propriétés données – au graphe infini qu'il symbolise.

Contrairement aux automates classiques, de part la nature des horloges, le modèle des automates temporisés est un modèle infini. Le concept de région (Alur, 1990) définit un modèle fini sémantiquement équivalent sur lequel pourront donc être appliquées les techniques classiques de vérification de formules. Plus précisément, le modèle des régions définit une relation d'équivalence sur un ensemble fini d'horloges telle que : (1) le nombre de classes d'équivalence est fini, (2) les valuations d'horloges d'une même classe satisfont les mêmes formules de la logique TCTL. Dans un automate de régions, le nombre de régions croit exponentiellement avec le nombre d'horloges et est proportionnel à la plus grande constante manipulée dans les contraintes. Les automates de zones ont pour but de réduire cette complexité : une zone est un regroupement de une ou plusieurs régions convexes. Des algorithmes de construction à la volée de l'automate de zone ainsi que des structures de données dédiées permettent de construire et de décider des propriétés sur l'automate de zone de manière efficace.

Dans le cas des réseaux de Petri temporels, on peut citer deux abstractions utiles, obtenues par groupement de certains ensembles d'états en *classes d'états*. La première, introduite dans (Berthomieu, 1983) permet l'analyse d'accessibilité des réseaux temporels et la vérification de leurs propriétés exprimables dans les logiques temporelles à temps linéaire (cf section 2.3). La seconde (Berthomieu, 2003) permet l'analyse de vivacité des réseaux temporels et la vérification de leurs propriétés exprimables en logiques temporelles à temps arborescent (cf section 2.3). Pour ces deux abs-

tractions, la finitude du graphe de classes est indécidable dans le cas général, toutefois ces graphes sont finis lorsque le réseau atemporel sous-jacent est borné.

3.2 Vérification par analyse de trajectoire majorante du flux d'arrivée

Cette section propose deux techniques qui s'appliquent à un modèle de systèmes temps réel exprimé sous forme de ressources (processeurs, mémoires, réseaux) et d'activités occupant ou partageant ces ressources (tâches, messages/paquets). Ces techniques reposent sur une connaissance de l'arrivée de ces activités dans le système au cours de la vie de celui-ci (on parle de flux d'activités) et donc de la demande de ressources liée à ces flux. Dans le contexte de la vérification à garantie déterministe, une condition nécessaire d'obtention de résultats est d'avoir une majoration des flux d'arrivée d'activités ; ceci implique notamment des hypothèses déterministes sur l'environnement et les perturbations pouvant affecter le système.

Ci-dessous, nous nous intéressons à une propriété s'exprimant par une contrainte de temps de réponse, les événements stimulus et réponse étant respectivement l'arrivée d'une activité et la fin de cette activité (activation d'une tâche / fin d'exécution de la tâche, demande d'émission d'un message / fin de transmission du message) et nous montrons comment évaluer quel sera le temps de réponse au cours de la vie du système, tout d'abord par l'analyse du pire temps de réponse (théorie de l'ordonnancement), puis par la méthode de "calcul de réseau" (*network calculus*).

Evaluation analytique du pire temps de réponse Dans la communauté de l'ordonnancement temps réel, il est d'usage d'évaluer le temps de réponse en considérant le pire cas afin de déterminer l'ordonnançabilité d'un ensemble de tâches périodiques/sporadiques. Pour un ensemble de N tâches (sources) périodiques ou sporadiques, chaque tâche (ou source) τ_i $(i = 1, 2, ..., N)$ est caractérisée par :

– une période (ou l'intervalle minimale d'interarrivée si l'arrivée de la tâche est sporadique) T_i avec laquelle la source génère une demande de travail (appelée par la suite une instance) ;

– une durée d'occupation de la ressource (ou serveur) C_i ; elle correspond à une demande de traitement de l'instance dans le serveur ;

– éventuellement le déphasage maximal de l'arrivée d'une instance par rapport au début prévu de sa période ; ce délai est appelé gigue maximale J_i (quand la source τ_i est strictement périodique, $J_i = 0$) ;

– ainsi que la borne supérieure sur le temps de réponse de cette tâche, ou contrainte d'échéance, δ_i.

Selon la politique d'ordonnancement du serveur, le pire temps de réponse d'une tâche (source) peut être obtenu en appliquant des résultats connus issus de la théorie de l'ordonnancement (Liu, 2000). A titre d'exemple, nous donnons les formules de calcul du pire temps de réponse *pour une politique d'ordonnancement à priorité fixe et sans préemption* (le cas non préemptif recouvre également l'ordonnancement de messages/paquets dans un réseau).

On note par i la priorité d'une source τ_i avec τ_i plus prioritaire que τ_j si $i < j$. Le pire cas pour une source périodique de priorité i correspond à la situation où au moment de la génération d'une instance, il y a une instance de priorité inférieure qui vient de commencer son traitement dans le serveur et où toutes les instances plus prioritaires sont aussi prêtes à être soumises au serveur.

Dans ce pire cas, le temps de réponse de la source de priorité i noté R_i est donné par :

$$R_i = C_i + J_i + I_i;\ R_i \leq T_i \tag{1.3}$$

où I_i est le temps d'interférence durant lequel le serveur est occupé par le traitement des instances générées par les sources plus prioritaires que τ_i et par une instance d'une source moins prioritaire générée juste avant l'arrivée de l'instance de priorité i.

I_i est donné par l'équation récurrente suivante :

$$I_i^{n+1} = \max_{i+1 \leq j \leq N}(C_j) + \sum_{j=1}^{i-1}\left(\left\lfloor \frac{I_i^n + J_j}{T_j} \right\rfloor + 1\right)C_j \tag{1.4}$$

où $max_{i+1 \leq j \leq N}(C_j)$ est le facteur de blocage dû à la non préemption. Cette équation récurrente peut être calculée à partir de $I_i^0 = 0$ et l'itération s'arrête quand $I_i^{n+1} = I_i^n$. La solution existe quand la charge normalisée du serveur générée par les sources, dont les priorités sont supérieures ou égales à i, est inférieure ou égale à 1 (i.e., $\sum_{j \leq i} \frac{C_j}{T_j} \leq 1$).

Evaluation de temps de réponse par la technique de "network calculus" Fournir une garantie de temps de réponse pour des messages ou paquets peut reposer sur la technique précédente. Néanmoins, d'une part, celle-ci repose sur de fortes hypothèses de périodicité et, d'autre part, elle est utilisable uniquement dans le cadre

d'une vérification a priori et peut difficilement, en raison du coût de calcul, être utilisée en ligne.

Dans le contexte de la communication temps réel, on peut prendre en compte le fait que les réseaux implantent des limiteurs du trafic en entrée ; c'est sous ces hypothèses qu'un modèle de trafic, dit $(\sigma, \rho) - borné$, a été proposé. Ce modèle est très utilisé dans la communauté de l'Internet pour le contrôle de la qualité de service car il est, de plus, simple à implémenter dans des mécanismes de contrôle d'admission pour estimer en ligne des temps de réponse. Cette approche considère le système comme un ensemble de N sources ayant des fonctions cumulatives d'arrivées correspondant, pour chaque source τ_i à une demande de quantité de ressource (ou demande de travail) $F_i(t)$ caractérisée par :

$$F_i(t) - F_i(s) \leq \sigma_i + \rho_i(t-s); \forall 0 \leq s \leq t \quad (1.5)$$

où σ_i représente la taille de la rafale maximale et ρ_i le débit moyen à long terme de la $i^{\text{ème}}$ source. La quantité du travail apportée par un paquet généré par la source τ_i c'est-à-dire le nombre de bits à traiter par le serveur, est définie par W_i bits. On considère que le serveur a un débit c (en bit/s).

Considérons maintenant un ensemble de N sources $(\sigma_i, \rho_i) - bornées$ $(i = 1, 2, ..., N)$ partageant un serveur non préemptif de capacité de traitement c bit/s et dans lequel les paquets sont servis selon la politique à priorité fixe. Rappelons qu'un paquet de la source τ_i est plus prioritaire qu'un paquet de τ_j si et seulement si $i < j$. La borne du temps de réponse D_i d'un paquet de la $i^{\text{ème}}$ source (c'est-à-dire celle de priorité i) est obtenue, selon la technique de « network calculus » (Le Boudec, 2002) et est définie par :

$$D_i = \frac{\sum_{j=1}^{i} \sigma_j + \max_{i+1 \leq j \leq N}(W_j)}{c - \sum_{j=1}^{i-1} \rho_j} \quad (1.6)$$

Pour que D_i soit borné, il faut que $c \geq \sum_{j=1}^{i} \rho_j$.

4 Vérification probabiliste

Dans cette section, nous proposons des méthodes permettant de vérifier que des exigences en termes de garanties probabilistes sont assurées par un système temps réel. De la même manière que précédemment, nous fournirons dans un premier temps des techniques d'analyse de modèles reposant sur un formalisme générique de modélisation de systèmes discrets temporisés et stochastiques, puis, dans un deuxième temps, nous considérerons des modèles de systèmes exprimés en termes d'activités (tâches, messages/paquets) et de ressources (processeurs, réseaux, mémoires). L'objectif général de ces méthodes est d'estimer la probabilité qu'une propriété soit respectée. Dans ce cas, on prend en compte des environnements et des perturbations aléatoires et, donc, des modèles d'arrivées de tâches et de messages/paquets également aléatoires.

4.1 Vérification par analyse de réseaux de files d'attente

Un système temps réel ou un réseau peut être modélisé sous la forme de réseaux de files d'attente. Ce formalisme permet de modéliser des *ressources* de capacité donnée (par exemple, le débit d'un réseau) et des demandes d'accès à cette ressource générées par des activités (par exemple, des messages/paquets) appelées *clients*. Les clients sont servis selon une politique d'accès à la ressource (politique d'ordonancement). L'élément de base d'un tel modèle est la station. Celle-ci est constituée d'un ou plusieurs serveurs, qui mettent en œuvre la politique d'accès à la ressource, et d'une file d'attente où sont classés les clients en attente d'accès à cette ressource. Le modèle du système dans ce formalisme comprend, pour chaque station : la spécification du flux d'arrivée des clients, généralement sous la forme d'un processus stochastique et le temps de service des clients dans le serveur, exprimé classiquement à l'aide d'une loi probabiliste. Parmi de tels modèles, certains sont analysables mathématiquement (Allen, 1990), comme c'est le cas pour les files d'attente dites M/G/1, qui représentent un système à demandes d'accès multiples à un serveur unique. Dans un tel modèle, il est donc possible de représenter la partie d'un système qui réalise l'accès à un bus de communication ou l'accès à un processeur. Nous considérons, donc, une file M/G/1 pour laquelle le flux d'arrivée des demandes est Markovien, le temps de service de chaque instance suit une distribution de probabilité générale et le nombre de serveur est égal à 1, pour évaluer les temps de réponse de messages/paquets partageant un bus.

Dans le cas où la politique de gestion d'accès à la ressource ne repose pas sur la priorité des clients, il est possible d'obtenir la distribution de probabilité des temps d'attente de ces clients ; ceci permet alors de vérifier si la probabilité de respect d'une propriété exprimée par une contrainte de temps de réponse est supérieure à un seuil donné.

Par contre, lorsque la politique d'accès repose sur les priorités, ce qui est le cas le plus fréquent dans les systèmes temps réel, il est impossible d'obtenir analytiquement, à l'aide du forma-

lisme des files d'attente , la distribution des temps d'attente des clients. Il est néanmoins possible de construire un modèle du système, selon ce formalisme, avec lequel les seuls résultats qu'il est possible de déduire sont les moments, à savoir la moyenne, la variance, ... Supposons que chaque activité ou source de messages τ_i génère un flux de messages de priorité i qui suit une loi de Poisson avec un taux d'arrivée λ_i (cette hypothèse est classique en l'absence de connaissance plus précise sur l'application). Chaque instance de message de priorité i a une durée de transmission constante C_i (temps de service) sur le bus. Ce système de communication peut alors être modélisé par une file M/G/1 avec une politique de service par priorité et sans préemption du serveur ; plus précisémment, ce système relève alors d'une sous-classe du modèle M/G/1, notée M/D/1 (D comme déterministe), pour laquelle il est possible de calculer le temps de réponse moyen (Allen, 1990).

On note :

- $\lambda = \lambda_1 + \lambda_2 + ... + \lambda_N$ avec N le nombre de sources dans le système,
- $a_j = \sum_{i=1}^{j} \lambda_j C_j$,
- $E[C^2] = \frac{\lambda_1}{\lambda} E[C_1^2] + ... + \frac{\lambda_N}{\lambda} E[C_N^2]$.

Comme signalé ci-dessus, seuls les moments peuvent être évalués ; en particulier, le temps moyen de réponse d'un message de priorité i est donné par :

$$E[Q_i] = E[q_i] + C_i = \frac{\lambda E[C^2]}{2(1 - a_{i-1})(1 - a_i)} + C_i \tag{1.7}$$

avec $a_0 = 0$.

Ce temps moyen de réponse est composé d'une part du temps de traitement du message C_i et d'autre part du temps moyen d'attente d'un message dans la file $E[q_i]$. On voit bien que $E[q_i]$ dépend non seulement des messages de priorité supérieure à i mais aussi de ceux de priorité inférieure à i à cause de la non-préemption. Notons que ce temps moyen ne donne qu'une indication et que, ni la moyenne, ni la variance ne peuvent apporter de garantie en termes de probabilité de respect d'une propriété sur un temps de réponse. Pour obtenir de telles garanties, on peut, soit utiliser le formalisme des réseaux de files d'attente ou de machines à états temporisées et stochastiques et appliquer une approche de simulation (Allen 1990) soit mettre en œuvre des techniques spécifiques aux types d'objets et de systèmes impliqués ; une telle technique est proposée au paragraphe suivant.

4.2 Estimation de probabilité de non-respect de propriété

Dans le cas où une politique d'accès à une ressource repose sur la notion de priorité, d'une part et, dès qu'on intégre la modélisation de perturbations aléatoires de l'environnement susceptibles d'altérer le service des clients, d'autre part, il est indispensable de développer des techniques spécifiques pour évaluer des distributions des temps d'attente pour l'accès à la ressource. Notons que l'altération du service d'une instance générée par une source (par exemple, un message/paquet généré par un ordre d'émission sur un réseau) peut avoir des conséquences différentes selon que le système est guidé par le temps (*time-driven*) ou par les événements (*event-driven*). Dans le premier cas, les instances sont générées périodiquement et une altération du service de l'instance conduit à la perte de celle-ci. Cette perte peut, suivant l'état du système, être tolérée. Néanmoins, il faut dans ce cas évaluer le nombre maximal d'instances consécutives perdues, qui n'altère pas le comportement global du système. Dans le cas où le système est guidé par les événements, la perte d'une instance, si elle est détectée, provoque un traitement supplémentaire (par exemple, réactivation de la tâche ou retransmission du message), surchargeant ainsi le système. Dans ces deux cas, il est nécessaire d'évaluer par des techniques spécifiques les garanties probabilistes de respect des propriétés à savoir, par exemple, probabilité de dépasser le nombre maximal de pertes successives d'instances, pour des systèmes guidés par le temps, ou probabilité de dépasser un temps de réponse donné pour chaque instance, pour des systèmes guidés par les événements.

Dans ce paragraphe, nous introduisons une méthode analytique spécifique pour évaluer la probabilité du non-respect de la contrainte sur un temps de réponse dans le pire cas en présence des erreurs aléatoires pour un système guidé par les événements et composé de N sources τ_i, partageant un même serveur avec priorité et sans préemption. On suppose qu'une erreur arrivant lors du service d'une instance est toujours détectée par le système et, alors, provoque une nouvelle génération de la même instance. Le temps de réponse d'une instance de priorité i dans un système avec perte est défini comme l'intervalle entre la date de sa génération initiale et la date à laquelle cette instance est effectivement servie ; on note par k le nombre total de générations de cette instance pour aboutir à un service correct. Pour chaque instance, nous obtenons alors le pire temps de réponse de cette instance en modifiant le calcul de la période d'interférence (équation 1.4), obtenu en section 3.2) :

$$I_i^{n+1}(k) = B_i + \sum_{j=1}^{i-1}(\left\lfloor \frac{I_i^n + J_j}{T_j} \right\rfloor + 1)C_j + E_i(k) \tag{1.8}$$

où

$$B_i = \max_{i+1 \leq j \leq N}(C_j)$$

La fonction $E_i(k)$ dépend du nombre d'erreurs k survenues durant l'intervalle $[0, t]$ et du temps SC_i nécessaire avant une nouvelle génération de l'instance de τ_i, soit :

$$E_i(k) = k \times SC_i \tag{1.9}$$

Il est évident que le temps de réponse augmente quand le nombre d'erreurs, k, sur la même instance augmente. Pour chaque source τ_i, il existe une valeur maximale de k, notée K_i^{max}, au-delà de laquelle la contrainte de temps de réponse des instances générées par la source n'est plus respectée. Il s'agit de la plus grande valeur de k telle que :

$$R_i^{max} \leq \delta_i$$

$$R_i^{max} = C_i + J_i + I_i(k) \tag{1.10}$$

Le calcul des temps de k est déterministe. Considérons, à présent, un modèle probabiliste d'occurrence des erreurs, la probabilité du non-respect de la contrainte sur le temps de réponse peut alors être évaluée. Soit $N_{err}(t)$ la variable aléatoire qui représente le nombre de fois où une instance est erronée pendant $[0, t]$, la probabilité p_i du non-respect pour une activité τ_i de priorité i donnée, est évaluée pour $t = R_i^{max}$ et $k = K_i^{max}$ et est donnée par :

$$p_i = 1 - P[N_{err}(R_i^{max}) \leq K_i^{max}] \tag{1.11}$$

Nous appelons cette probabilité *la probabilité du non-respect dans le pire cas* car le temps de réponse calculé est celui du pire cas. L'évaluation va dépendre du modèle probabiliste de $N_{err}(t)$ selon lequel une erreur survient.

5 Exemple

Pour illustrer les apports de l'ensemble de techniques ci-dessus et leur complémentarité, nous considérons une application embarquée distribuée autour du réseau CAN (*Controller Area Network*) (Paret, 1996) dans l'automobile. Le protocole d'accès au médium de CAN suit le principe de bus priorisé.

L'exemple choisi, "réseau inter - système", provient d'une étude de cas proposée par PSA Peugeot-Citroën. Elle met en œuvre une architecture matérielle (des calculateurs connectés sur un réseau CAN dont le débit est 250 Kbit/s). Dans ce contexte, les calculateurs sont désignés ECU (*Electronic Control Unit*). L'abstraction que nous avons de l'architecture opérationnelle de ce système est limitée au réseau et aux messages échangés entre calculateurs sous des hypothèses d'envoi de messages de la part des tâches localisées sur chaque ECU. Dans le contexte de l'électronique embarquée dans les automobiles, cette partie du système s'appelle une messagerie. Les calculateurs connectés sur CAN sont :

– CM : il supporte les tâches implantant les fonctions de contrôle du moteur ;

– ABS/CDS : sur ce calculateur s'exécutent les tâches concernant le contrôle de l'ABS (*Anti Blocking System* ou système anti-blocage des freins) et le contrôle de stabilité ;

– BSI : (Boîtier de Servitude Intelligent) il s'agit du centre du système embarqué ; il sert de passerelle entre différents réseaux présents dans l'architecture et transmet certains ordres du conducteur aux fonctions chargées de les mettre en œuvre sur d'autres calculateurs ;

– SUS : les tâches implantées sur ce calculateur sont en charge du contrôle de la suspension dynamique ;

– BVA : les tâches implantées sur ce calculateur concernent la gestion de la Boîte de Vitesse Automatique ;

– CAV/CdP : la mesure de l'angle volant (Capteur d'Angle Volant) et un correcteur de phare constituent les deux fonctionnalités supportées par ce calculateur.

La spécification de la messagerie, c'est-à-dire de l'ensemble des messages qui transitent sur le réseau CAN est donné dans le tableau 1.1. Cette messagerie est entièrement définie sous des hypothèses de périodicité des demandes d'émission des messages de la part des tâches localisées sur les calculateurs cités ci-dessus. Les messages sont classés par priorité décroissante (M_i est plus prioritaire que M_j si $i < j$) ; pour chaque message, on donne sa source d'émission, ECU_i, c'est-à-dire le calculateur qui supporte la tâche émettant ce message, s_i la taille, en octets, des données utiles qu'il contient (rappelons que, lors de la transmission d'une information "applicative", des champs supplémentaires sont ajoutées à cette information pour en constituer une "trame" ; dans le cas du réseau CAN, la taille réelle de la trame est majorée par $\lfloor \frac{34+8\times s_i}{4} \rfloor + 47 + 8 \times s_i$ bits), T_i la période, en ms, d'émission du message.

L'avant-dernière colonne du tableau 1.1 donne C_i la durée de transmission, en ms, de la trame contenant le message M_i, à partir de sa taille réelle et de τ_{bit}, la durée de transmission d'un bit (soit, $\tau_{bit} = \frac{1}{250} ms$ pour un débit du réseau de 250 Kbits/s) : $C_i = (\lfloor \frac{34+8\times s_i}{4} \rfloor + 47 + 8 \times s_i)\tau_{bit}$. Enfin,

dans la dernière colonne, δ_i représente la borne imposée sur le temps de réponse de chaque instance du message M_i. Nous considérons que la propriété à vérifier s'exprime par une contrainte sur le temps de réponse de chaque message : la durée entre la date d'émission de toute instance d'un message M_i et la fin de sa transmission sur le réseau doit être inférieure à δ_i. Nous appliquons dans un premier temps les techniques vues ci-dessus, pour obtenir une garantie déterministe de cette propriété, puis, dans un deuxième temps, nous regardons comment obtenir des garanties probabilistes.

5.1 Garantie déterministe

Dans cette section, on s'attache à vérifier que la propriété est respectée par toute instance de chaque message. Pour celà, il est possible d'appliquer directement les formules 1.3 et 1.4 pour obtenir le pire temps de réponse par la méthode vue au paragraphe 3.2. Les résultats, obtenus pour une gigue maximale J_i nulle, figurent dans la deuxième colonne, R_i, du tableau 1.2.

Pour appliquer la technique vue en 3.2, il est nécessaire, au préalable, de traduire le modèle initial (tableau 1.1) en un modèle de trafic $(\sigma, \rho) - borné$, soit :

– ρ_i, valeur du débit moyen à long terme pour une source τ_i

$$\rho_i = \frac{W_i}{T_i} \tag{1.12}$$

Notons que pour évaluer W_i, il suffit de constater que le temps de traitement C_i induit par toute instance de τ_i est dû à une charge W_i pour un serveur de débit c ($C_i = W_i/c$).

– σ_i, la taille maximale de la rafale ; celle-ci est définie comme étant la plus petite valeur de σ_i qui satisfait la contrainte 1.5. En général la rafale existe quand le trafic périodique est perturbé par une variation de gigue. La taille de la rafale dépend alors fortement des valeurs de la gigue. En considérant le pire cas qui correspond à une arrivée générée effectivement J_i unités de temps avant l'instant d'activation prévu, il a été montré dans (Koubâa, 2004) que la taille optimale de la rafale (σ_i minimale qui majore la courbe d'arrivée) est donnée par :

$$\sigma_i = \frac{W_i}{T_i}(T_i + J_i) \tag{1.13}$$

Dans l'exemple considéré, par application des formules 1.12, 1.13 et 1.6, on obtient une borne supérieure pour le temps de réponse de chaque message. Ce résultat est donné dans la troisième colonne, D_i du tableau 1.2.

Message	ECU_i	s_i	T_i	C_i	δ_i
M_1	CM	8	10	0,54	2
M_2	CAV/CdP	3	14	0,34	2
M_3	CM	3	20	0,34	2
M_4	BVA	2	15	0,30	2,5
M_5	ABS/CDS	5	20	0,42	2,5
M_6	ABS/CDS	5	40	0,42	10
M_7	ABS/CDS	4	15	0,38	10
M_8	BSI	5	50	0,42	10
M_9	SUS	4	20	0,38	10
M_{10}	CM	7	100	0,50	10
M_{11}	BVA	5	50	0,42	10
M_{12}	ABS/CDS	1	100	0,2	100

TAB. 1.1 – Spécification de la messagerie (temps en ms)

Nous pouvons remarquer que les deux techniques appliquées ci-dessus permettent, chacune, d'obtenir une borne sur le temps de réponse et, ainsi de vérifier de manière déterministe qu'une contrainte de temps de réponse est respectée. L'évaluation analytique de pire temps de réponse fournit des bornes plus petites ou égales à celles fournies par l'approche du "network calculus" et, ainsi, les résultats obtenus par la dernière sont plus pessimistes. En effet, dans l'exemple présenté ici, il n'est pas possible d'obtenir de garantie déterministe de la propriété sur le temps de réponse du message M_5 en appliquant la technique du "network calculus" alors qu'il est possible de la garantir par l'approche fournie en section 3.2. En revanche, il faut noter que la technique de "network calculus" » peut prendre en compte un modèle de trafic plus général (borné supérieurement), in-

Message	R_i	D_i	$E[Q_i]$	$E[q_i]$	K_i^{max}	R_i^{max}	p_i
M_1	1,04	1,04	0,587	0,047	1	1,67	0,01
M_2	1,38	1,46	0,391	0,051	0	1,38	0,1
M_3	1,72	1,87	0,394	0,054	0	1,72	0,1
M_4	2,02	2,23	0,356	0,056	0	2,02	0,1
M_5	2,44	2,76	0,479	0,059	0	2,44	0,1
M_6	2,86	3,31	0,481	0,061	11	9,81	10^{-12}
M_7	3,24	3,80	0,443	0,063	10	9,56	10^{-11}
M_8	3,66	4,42	0,486	0,066	10	9,98	10^{-11}
M_9	4,04	4,93	0,448	0,068	9	9,73	10^{-10}
M_{10}	4,46	5,57	0,570	0,070	8	9,52	10^{-9}
M_{11}	4,72	5,93	0,491	0,071	8	9,78	10^{-9}
M_{12}	4,72	6,33	0,332	0,072	122	99,38	0

TAB. 1.2 – Résultats obtenus par application des techniques présentées dans les sections 3.2 (colonne R_i), 3.2 (colonne D_i) , 4.1 (colonnes $E[Q_i]$ et $E[q_i]$) et 4.2 (colonnes K_i^{max}, R_i^{max}, p_i)

cluant le trafic périodique/sporadique et être utilisée pour intégrer des vérifications en ligne (par exemple, dans des mécanismes de contrôle d'admission dans les architectures de qualité de service).

5.2 Garantie probabiliste

En modélisant ce système sous la forme d'un réseau de files d'attente et en appliquant la formule 1.7, pour $\lambda_i = 1/T_i$, nous pouvons évaluer les valeurs de $E[q_i]$ et $E[Q_i]$ pour chacun des messages. En ce qui concerne le temps moyen de réponse, comme la charge du réseau n'est pas importante ($\sum_{j=1}^{N} \frac{C_j}{T_j} \approx 0,22$), nous pouvons constater que la moyenne du temps d'attente est très faible, la majeure partie du temps de réponse étant due à la durée de transmission. Ce résultat ne fournit aucune garantie probabiliste sur la faisabilité de cette messagerie ; par contre, il apporte une aide au concepteur de la messagerie pour paramétrer son application (choix de la priorité des messages au vu des temps moyens de réponse obtenus, dimensionnement du débit du réseau, ...). Pour compléter ces évaluations et, en particulier, obtenir des résultats en termes de garantie probabiliste, en prenant en compte les erreurs, il faut recourir à d'autres moyens.

Aussi, dans ce qui suit, nous cherchons à évaluer, pour chaque message M_i, sa probabilité de non-respect de l'échéance pour un modèle d'erreurs donné. Dans la réalité, la plupart des bus CAN embarqués dans l'automobile sont soumis à des erreurs de transmission car les supports de transmission de type "paires torsadées non blindées", en particulier, sont sensibles à des perturbations dues aux champs électromagnétiques. Dans CAN, chaque trame erronée est retransmise automatiquement augmentant ainsi le temps de réponse global du message. Le pire temps supplémentaire induit par k retransmissions peut être estimé par :

$$E_i(k) = k \times (23\tau_{bit} + \max_{j \leq i}(C_j)) \qquad (1.14)$$

Le temps supplémentaire induit par une erreur de transmission est composé d'une part de la trame de signalisation d'erreurs ($23\tau_{bit}$ dans le pire cas) et d'autre part de la retransmission de la trame corrompue. Dans le pire cas, toutes les erreurs se produisent au dernier bit de la plus longue trame susceptible d'être retransmise. Compte tenu du protocole d'accès qui repose sur la priorité, seule une trame plus prioritaire ou égale à la priorité i peut être retransmise.

En appliquant les formules 1.8, 1.14 et 1.10, on obtient K_i^{max} correspondant au plus grand R_i^{max}tel que $R_i^{max} \leq \delta_i$ (voir tableau 1.2). La probabilité p_i de non-respect de la propriété a été calculée sous l'hypothèse que chaque transmission a une probabilité d'erreur $p = 0,1$ (dernière colonne du tableau 1.2). La probabilité d'avoir exactement k erreurs (forcément consécutives dans notre cas) suit une distribution géométrique. Selon la formule 1.11, on a :

$$p_i = 1 - \sum_{k=0}^{K_i^{max}} P[N_{err}(R_i^{max}) = k]$$

$$p_i = 1 - \sum_{k=0}^{K_i^{max}} p^k(1-p) \qquad (1.15)$$

La probabilité de non-respect est fonction du nombre maximal de retransmissions autorisé. Ainsi nous pouvons constater que des messages prioritaires mais ayant des échéances serrées ont

des probabilités de non-respect plus importantes. Cette information permet au concepteur d'applications de bien dimensionner le système s'il exige une certaine robustesse vis à vis du respect des contraintes sur des messages de priorité élevée.

6 Conclusions

La vérification d'applications est une activité complexe qui est souvent imposée par des réglementations internationales, nationales ou internes à une entreprise. En effet, ces applications relèvent de systèmes dits critiques dont la mise en service passe par une garantie déterministe ou probabiliste de leur bon fonctionnement. Cette garantie s'obtient de plus en plus tout au long de leur développement par analyse de modèles du système. Ceci amène deux problèmes principaux. Le premier est celui de la modélisation, c'est-à-dire de la spécification d'une abstraction du système telle qu'elle puisse être analysée et que cette analyse fournisse un verdict pertinent pour les garanties attendues. Le deuxième problème est celui de l'exploitation des modèles soit par son analyse mathématique, soit par simulation. Résoudre ces deux problèmes repose sur les formalismes qui supportent la représentation des systèmes à événements discrets temporisés et éventuellement probabilistes. Si de nombreuses avancées ont été obtenues ces dernières années sur ce sujet (développement de nouveaux formalismes, maîtrise de la complexité de l'analyse) et si on commence à voir quelques outils industriels applicant ces résultats, il n'en demeure pas moins que la généralisation de l'application des techniques de vérification reste à pénétrer tous les secteurs industriels concernés par la conception d'applications temps réel. Dans ce chapitre, nous avons présenté la problématique de la vérification dans le contexte du temps réel. Puis, nous avons identifié quelques formalismes et techniques d'analyse qui ont atteint un degré significatif de maturité et qui, pour certaines, sont déjà appliquées dans l'industrie. Enfin, grâce à l'exemple concret d'un système embarqué dans une automobile, nous avons montré comment certaines des techniques présentées pouvaient s'appliquer, quelles abstractions du système devaient être construites et quels résultats pratiques on pouvait en attendre.

Index

Bibliographie

A.O. Allen (1990), *Probability, statistics and queueing theory with computer science applications*, Academic Press, Inc.

R. Alur, C. Courcoubetis, D. Dill (1990), *Model-Checking for real-time systems*, Proc of 5th IEEE Symp. on Logic in Computer Science

A. Arnold (1992), *Systèmes de transitions finis et sémantique des processus communicants*, Masson Paris.

B. Berthomieu and M. Menasche (1983), *An Enumerative Approach for Analyzing Time Petri Nets*, IFIP Congress 1983.

B. Berthomieu, F. Vernadat (2003), *State class constructions for branching analysis of Time Petri nets*, In Proceedings of TACAS 2003. Springer Verlag, LNCS 2619, ISBN 3-540-00898-5

D. Brand, P. Zafiropolo (1983), *On Communicating Finite-State Machines*, J.A.C.M, Vol. n^o 2

J.R Burch, E.M Clark, K. L McMillan, DL Dill and J.L Hwang (1990), *Symbolic model-checking : 10^{20} states and beyond*, Proc of 5th IEEE Symp. on Logic in Computer Science, Avril 83

M. Diaz (2003), *Vérification et mise en oeuvre des réseaux de Petri*, Hermes Science, ISBN 2-7462-0445-2

A. Finkel, Ph. Schnoebelen (2001), *Well-structured transition systems everywhere !*, Theoretical Computer Science 256(1-2) : 63-92.

R. Karp, R. Miller (1969), *Parallel Program Shemata*, Journal of Computer and System Science, Vol n^0 3,4

A. Koubâa and Y. Q. Song (2004), *Evaluation and improvement of response time bounds for real-time applications under non pre-emptive fixed priority scheduling*, International Journal of Production Research, Vol.42, NO.14, pp 2899-2913, July 2004, Taylor & Francis Ltd.

J.Y. Le Boudec and P. Thiran (2002), *Network Calculus : A Theory of Deterministic Queueing Systems For The Internet*, Editions Springer Verlag New-York, Inc.

J.W.S. Liu (2000), *Real-Time Systems*, Upper Saddle River, New Jersey, USA, Editions Prentice-Hall, Inc.

D. Paret (1996), *Le bus CAN (controller Area Network)*, Dunod.

Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie et A. Petit (1999), *Vérification de logiciels : techniques et outils du model-checking*, Vuibert.