Génération automatique de modèles d'application Temps-Réel

Fabrice Jumel

▶ **To cite this version:**

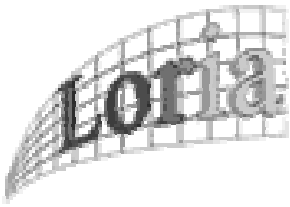
Fabrice Jumel. Génération automatique de modèles d'application Temps-Réel. [Stage] 99-R-238 || jumel99a, 1999, 50 p. inria-00107814

HAL Id: inria-00107814
https://inria.hal.science/inria-00107814

Submitted on 19 Oct 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Génération automatique de modèles d'application Temps-Réel

étude réalisée au sein de l'équipe TRIO
(Temps Réel et InterOpérabilité) du LORIA
(Laboratoire Lorrain de Recherche en Informatique et ses Applications)

Mémoire de DEA d'i nformatique

présenté et soutenu le 17 septembre 1999

par

Fabrice JUMEL

Composition du jury :

Jean-Paul Haton	Professeur de l'université Nancy I
Michaël Rusinowitch	Directeur de recherche INRIA
Didier Galmiche	Maître de conférence Nancy I
Jeanine Souquieres	Professeur de l'université Nancy 1
Françoise Simonot	Maître de conférence INPL
Jean-pierre Thomesse	Professeur de l'INPL

Remerciements

Je tiens à remercier toutes les personnes qui m'ont permis de mener « à bien » ce stage .

Tout d'abord ma tutrice Françoise Simonot, pour sa disponibilité(en particulier dans les moments difficiles) et pour ses nombreux conseils ;

Les différents membres de l'équipe TRIO pour leur aide et les bons moments passés au cours de cette période . En particulier Gerald et Nicolas qui eurent à me supporter pendant de longs mois.

Résumé

Le but du projet est de mettre en place une méthodologie et des outils permettant de valider les propriétés temporelles d'applications distribuées complexes (hétérogénéité des matériels supports, approche événementielle) spécifiées dans un langage de description d'architecture (ici Ardeco propre au consortium Covadis [Thomas97]). Ce langage permet de décrire une architecture fonctionnelle sous forme d'une architecture de composants.

L'architecture fonctionnelle permet de décrire les différentes fonctions des composants et leur interactions logiques. L'architecture matérielle constitue l'informatique support (calculateurs et systèmes exécutifs, réseaux et protocoles, topologie de connexion). L'architecture opérationnelle est le résultat de la projection d'une architecture fonctionnelle sur une architecture matérielle.

Nous sommes partis des travaux initiés au sein du consortium COVADIS et les avons étendus par des propositions de mécanismes supplémentaires.Il a fallu également définir les mécanismes de projection de l'architecture fonctionnelle sur l'architecture matérielle, puis proposer les modèles de l'architecture matérielle (ordonnanceurs locaux, protocoles de communication) et les interactions de ces modèles avec ceux de l'architecture fonctionnelle.

SOMMAIRE

1 PROBLÉMATIQUE 6
1.1 PRÉSENTATION DU LANGAGE ARDECO. 6
1.2 SIMULATION D'ARCHITECTURE OPÉRATIONNELLE - OBJECTIFS 7
2 PRINCIPES DE CONSTRUCTION DE MODÈLES D'AO 9
2.1 MODÉLISATION D'UN COMPOSANT FONCTIONNEL 9
2.1.1 Quelques exemples d'automates 9
2.1.2 Instanciation des automates au cours de la simulation 11
2.1.3 Principe de traduction Ardeco vers OPNET 12
2.2 PRISE EN COMPTE DE L'AM ET DE LA DISTRIBUTION 13
2.2.1 Instruction « traiter_pendant » 13
2.2.2 Les connexions entre composants 14
3 MODÈLE DES COMPOSANTS DE L'AM 16
3.1 IDENTIFICATION DES COMPOSANTS DE L'ARCHITECTURE MATÉRIELLE 16
3.2 PRISE EN COMPTE DES SURCHARGES DUES À L'EXÉCUTIF 16
3.3 MODÉLISATION D'UN ORDONNANCEUR DE TÂCHE 17
3.3.1 Le Modèle JSD d'OPNET 17
Le modèle proposé 19
3.3.3 Tests des modèles d'ordonnanceur 21
3.3.4 Modélisation des systèmes de communications 26
4 SPÉCIFICATION D'UNE ARCHITECTURE MATÉRIELLE ET D'UNE DISTRIBUTION 35
4.1 LE LANGAGE DE DESCRIPTION D'ARCHITECTURE MATÉRIELLE 35
4.1.1 Définition des sites 35
4.1.2 Définition du réseau 35
4.2 MODÉLISATION DE L'AO 36
4.2.1 Projection d'un composant sur un site 36
4.2.2 Projection des échanges entre composants sur les systèmes de communications 36
4.3 CALCUL DES TEMPS DE TRAITEMENT 36
5 CONCLUSIONS ET PERSPECTIVES 38
6 BIBLIOGRAPHIE 38
7 ANNEXES 41

INTRODUCTION

Ce stage s'inscrit dans le cadre du projet COVADIS (Conception et Validation d'Applications DIStribuées temps réel) qui rassemble :
- des laboratoires de recherche (LORIA-Nancy, IRCyN-Nantes, I3S-Nice, LAIL-Lille)
- un industriel Thomson-CSF (Laboratoire Central de Recherche - Marcoussis)

Les travaux s'appliquent aux applications de surveillance et contrôle aérien dans les domaines civil et militaire.
Le travail s'est déroulé dans l'équipe TRIO (Temps Réel et InterOpérabilité) du LORIA. Le rôle de cette équipe, dans COVADIS, est de développer un outil de modélisation et validation d'architecture opérationnelle à partir du langage de description d'architecture fonctionnelle développé dans le projet. Sous le terme « validation », on entend la vérification de propriétés temporelles ; la méthode de vérification choisie, au sein du groupe, repose sur l'évaluation de performances par exécution-simulation de modèles d'architecture opérationnelle. Le moteur de simulation est celui de l'outil OPNET[1] et les modèles à construire le sont dans le formalisme des automates temporisés.

L'objectif de mon stage est double :
- d'une part, spécifier les différents éléments de l'architecture fonctionnelle qui doivent être pris en compte pour concevoir et valider l'architecture opérationnelle (projection de l'architecture fonctionnelle sur l'architecture informatique support, appelée architecture matérielle)
- et d'autre part, construire les modèles de ces architectures matérielles afin de pouvoir les utiliser dans la simulation de l'architecture opérationnelle.

La partie pratique de ce stage (Modélisation de modèles d'ordonnanceur) a été faite dans le cadre d'un stage ingénieur ENSEM.

Le document est structuré ainsi :

Après une présentation de la problématique du projet Covadis, la méthode de modélisation de l'architecture opérationnelle retenue est présentée. On trouve dans une troisième partie les modèles d'architecture matérielle qui sont proposés. On définit enfin dans une quatrième partie une spécification des architectures matérielles mais aussi de la distribution (nécessaire à la création d'une architecture opérationnelle).
En annexe figure en particulier un état de l'art sur les ADL et la prise en compte du temps réel.

[1] OPNET est un atelier développé par la société Mil3 Inc.

1 Problématique

Ce stage s'inscrit dans le cadre du projet COVADIS (Conception et Validation d'Applications DIStribuées temps réel) qui rassemble :
- des laboratoires de recherche (LORIA-Nancy, IRCyN-Nantes, I3S-Nice, LAIL-Lille)
- un industriel Thomson-CSF (Laboratoire Central de Recherche - Marcoussis)

Les travaux s'appliquent aux applications de surveillance et contrôle aérien dans les domaines civil et militaire.
Le travail s'est déroulé dans l'équipe TRIO (Temps Réel et InterOpérabilité) du LORIA. Le rôle de cette équipe, dans COVADIS, est de développer un outil de modélisation et validation d'architecture opérationnelle à partir du langage de description d'architecture fonctionnelle développé dans le projet. Sous le terme « validation », on entend la vérification de propriétés temporelles ; la méthode de vérification choisie, au sein du groupe, repose sur l'évaluation de performances par exécution-simulation de modèles d'architecture opérationnelle. Le moteur de simulation est celui de l'outil OPNET[2] et les modèles à construire le sont dans le formalisme des automates temporisés.

L'objectif de mon stage est double :
- d'une part, spécifier les différents éléments de l'architecture fonctionnelle qui doivent être pris en compte pour concevoir et valider l'architecture opérationnelle (projection de l'architecture fonctionnelle sur l'architecture informatique support, appelée architecture matérielle)
- et d'autre part, construire les modèles de ces architectures matérielles afin de pouvoir les utiliser dans la simulation de l'architecture opérationnelle.

Ce stage s'est poursuivi par un stage de DEA d'informatique de Nancy.

1.1 Présentation du langage Ardeco.

Ardeco est un langage de description d'architecture (ADL) au sens classique du terme et enrichi par la possibilité de décrire des architectures opérationnelles (prise en compte du support exécutif). Il offre trois concepts de base : la description de composants génériques, la description d'une architecture d'instances de ces composants (architecture logique) et la description d'une distribution de cette architecture sur un support informatique.

– *Description de composants* :

Un composant est une boîte noire dont les effets sur l'extérieur sont perçus au travers de « points d'accès en sortie » ; à ceux-ci se rattachent une notion d'événement (émission, par le composant d'une information sur ce point d'accès) et de donnée (valeur de l'information). Le composant réagit à plusieurs types de stimulus : une sollicitation venant de l'extérieur sur un « point d'accès en entrée », l'échéance d'une période, le changement de valeur d'une donnée interne. La spécification de ces réactions se fait, respectivement, au travers des définitions de déclencheurs « sur … », « chaque … », « quand … ».
Lors d'un stimulus sur un de ces déclencheurs, les actions réalisées par le composant sont définies comme des « activités », lesquelles ont pour rôle d'exécuter un programme (« code déclenché »). Ce programme est un modèle de ce qu'exécutera l'application. Des gardes peuvent être associées au stimulus ; elles portent sur des valeurs de données, sur des contraintes de synchronisation entre activités ou s'expriment par des chiens de garde.
Les instructions décrivant un programme en Ardeco sont des transformations de données, des attentes, éventuellement bornées par le temps, d'événements, des émissions sur point d'accès en sortie, des levées d'exceptions et des changements de versions. Toutes ces opérations sont supposées se dérouler en temps nul. Pour représenter ce que deviendra le composant au sein d'une architecture opérationnelle, une instruction « traiter pendant dt » a été introduite . Elle permet de spécifier le temps nécessaire pour réaliser les transformations de données qui la précède dans le texte du programme. Des détails supplémentaires sur ce langage sont disponibles dans [Thomas 97b] et [Thomas 98].

[2] OPNET est un atelier développé par la société Mil3 Inc.

Un composant type est défini par :

- Une identification
- Une interface décrivant la partie visible du composant
  - Types génériques utilisés
  - Liste des points d'accès en sortie
  - Liste des points d'accès en entrée
  - Liste des versions disponibles (traitement des modes de marche) et, pour chacune d'elles, les relations entre points d'entrée et de sortie (causalité stimulus / réponse)
- Un comportement
  - Liste de données internes
  - Description des codes déclenchés lors des sollicitations externes (« sur point d'accès en entrée ») ou internes (« chaque période », « quand condition »)
  - Liste des propriétés exigées du composant au sein d'une architecture opérationnelle
  - Liste des gardes non temporelles sur les réactions.

– Description d'architecture de composants (architecture fonctionnelle)

Une autre partie du langage spécifie les interactions entre les composants à l'aide de combinateurs (« 1 – 1 », « 1 - & - n », « n - & - 1 », « n - | - 1 », où « | » et « & » ont respectivement le sens du ET et du OU logique). Ceux-ci relient les points d'accès en sortie d'un composant aux points d'accès en entrée d'autres composants. Ils s'accompagnent de fonctions d'implémentation du ET et du OU, le cas échéant ainsi que de fonctions d'adaptation de types (marshall / unmarshall). Dans l'étude préliminaire qui est l'objectif de ce stage, le seul combinateur traduit est de type « 1 – 1 » et on fait abstraction des adaptations de types. On pourra se référer aux documents Covadis [Thomas 97a], [Thomas 97b], [Thomas 98], [Thomas 99a] pour plus de renseignements.

– Description d'architecture opérationnelle

Il s'agit de fournir au concepteur un moyen de décrire d'une part l'architecture informatique qui supportera l'architecture de composants (architecture matérielle) et, d'autre part, la projection de cette dernière sur l'architecture matérielle.
Mon stage doit apporter des éléments pour la définition du langage de cette description

1.2 Simulation d'architecture opérationnelle - objectifs

Le but de la simulation est de donner une aide aux concepteurs d'architecture opérationnelles, en lui offrant la possibilité de valider a priori, c'est-à-dire avant tout investissement en matériel et avant l'implémentation, et d'optimiser les choix en matière d'AO. Plus particulièrement, le but de la simulation est alors double :

– permettre la validation des propriétés temporelles de l'architecture opérationnelle

– fournir une aide à son dimensionnement en utilisant la simulation pour rechercher les valeurs optimales des paramètres.

Il est en effet possible de définir plusieurs architectures matérielles et de jouer sur le placement des tâches et leurs configurations ainsi que leur mode de gestion pour obtenir les « meilleures solutions ». Compte tenu de la complexité des architectures à modéliser, les méthodes exhaustives d'analyse de modèles sont impossibles à appliquer dans la pratique (explosion combinatoire des graphes d'atteignabilité). Aussi, la méthode de validation choisie est la simulation et repose donc sur des études du type (modélisation/simulation/analyse de performances).

L'outil utilisé pour mener à bien cette validation est OPNET. OPNET a été développé en particulier pour permettre de valider le fonctionnement d'architectures distribués où la notion de réseau est

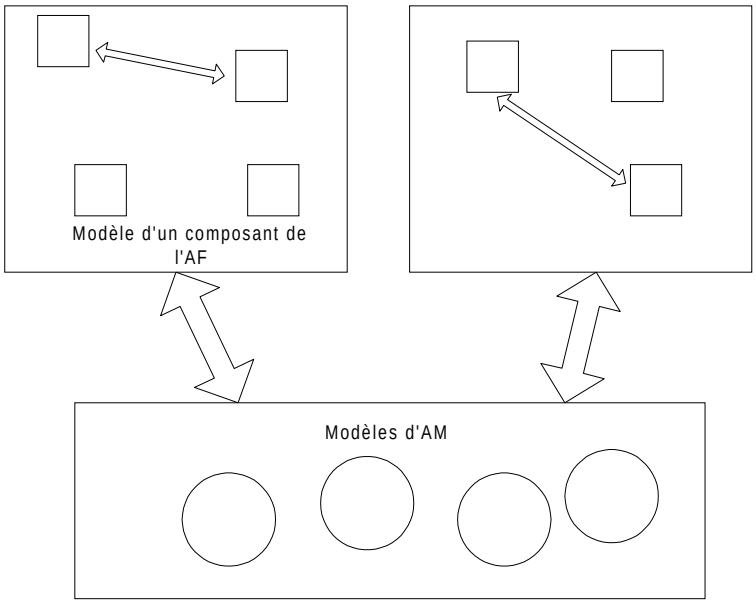
fondamentale. Il permet de modéliser les systèmes suivant une structure hiérarchique qui repose sur 3 niveaux de modélisation :

– - le niveau réseau (le modèle de la topologie de l'architecture matérielle)
– - le niveau nœud (une image des machines connectées sur un réseau)
– - le niveau « processus » (les activités supportées par un nœud)

OPNET est cependant tout a fait adapté pour simuler tout type d'architecture logique. Ainsi au niveau de la description des modules (process), on utilise un formalisme reposant sur la notion d'automates temporisés synchronisés qui permettent de modéliser tout système à événements discret.
Par ailleurs, il permet de décrire des modèles de manière modulaire et de spécifier leurs interactions par la notion de synchronisation d'automates (événements, informations associées à l'événement). Dans le cadre de notre étude ce point est fondamental car, au cours de la simulation, nous aurons à faire cohabiter et communiquer deux types de modèles. En effet plutôt que d'essayer de modéliser directement l'architecture opérationnelle, il a été choisi de modéliser parallèlement l'architecture fonctionnelle et l'architecture matérielle support. Cette séparation permet de modifier rapidement les caractéristiques de l'architecture matérielle et préserve tous les avantages de l'architecture fonctionnelle en terme de lisibilité et de modularité. Cette approche nécessite donc d'une part une modélisation de l'architecture fonctionnelle (travaux de Fce Simonot), la modélisation de l'architecture matérielle mais surtout la formalisation des interactions entre les modèles d'architecture fonctionnelle et d'architecture matérielle.

1.2.1 Cahier des charges du stage

Mon travail a donc consisté en partie à compléter et réaliser sous Opnet les modèles d'architecture fonctionnelle spécifiés par Françoise Simonot et à spécifier et réaliser les modèles d'architecture matérielle et les interactions avec les précédents.
Pour mener à bien les premiers essais un certain nombres d'hypothèse ont du être considérés. Toutes les possibilités d'Ardeco n'ont pas été prise en compte :
En particulier :
Les déclencheurs quand n'ont pas été considérés car il nécessite la gestion du devenir de toutes les variables du système.
La gestion des combinateurs aux niveaux des connexions entres les composants n'a pas été prise en compte



Modélisation de l'AO

2 Principes de construction de modèles d'AO

2.1 Modélisation d'un composant fonctionnel

Les travaux de Fce Simonot [Thomas 99] ont permis d'extraire de la notion de composants telle qu'elle est définie en Ardeco huit automates qui modélisent le comportement de l'architecture fonctionnelle.

On trouve plusieurs automates génériques dont la création est automatique soit, statiquement, en début de simulation, soit, dynamiquement, lors de la création d'activités.

- AG : un automate de gestion de l'activité globale du composant (c'est à dire des déclencheurs) : son rôle est d'assurer la création des activités du composant (Sur, Chaque, Quand).
- AP : un automate de gestion de la prise en compte (une fois que l'activité est crée)
- AA : un automate de gestion de l'activité (du code déclenché) qui est crée une fois q'un composant est effectivement pris en compte.
- AEXCPT : un automate de gestion des exceptions au niveau du composant
- ACEXCPT : un automate de gestion des codes d'exceptions au niveau des activités
- ACEXP : un automate de gestion des expirations
- AWD : un automate de gestion des chiens de garde (nécessaire à la gestion des expirations)
- AEVT : un gestionnaire d'événements qui permet aux différents automates de s'abonner à l'attente de tel ou tel événement.

Les deux derniers ont pour rôle d'offrir les mécanismes nécessaires à la gestion de la synchronisation des premiers.
Il y aura, lors de la simulation un exemplaire des automates AG , AEXCPT , AWD, AEVT par composant (création statique) et un exemplaire de chacun des autres lors de la création d'activités (création dynamique).
Nous avons vérifié la cohérence de ces automates et nous avons proposé des mécanismes pour les gérer au cours de la simulation, permettre leur synchronisation et assurer l'échange nécessaire des données.

2.1.1 Quelques exemples d'automates

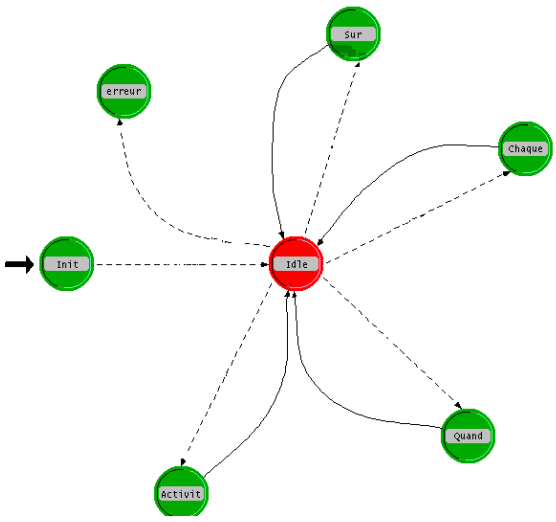


figure 0 automate de gestion des déclencheurs

Cette automate gère la création de toutes les activités du composants de façon périodique ou en réponse à des sollicitations. En particulier le déclencheur sur , relié à un point d'entrée entraîne la création d'une activité.

Automate de contrôle de la prise en compte du composant (les transitions sont étiquetées par des événements ; ceux-ci ne figurent pas sur l'automate dans un souci de lisibilité)

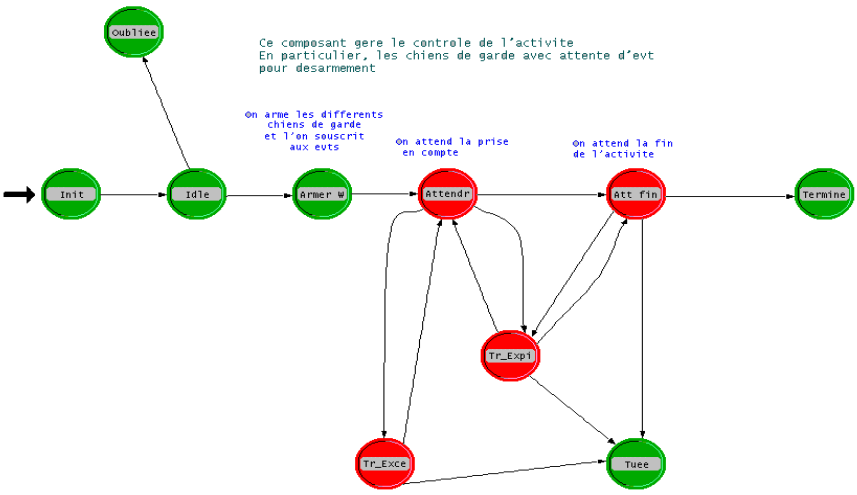


figure 1 Automate de contrôle de la prise en compte

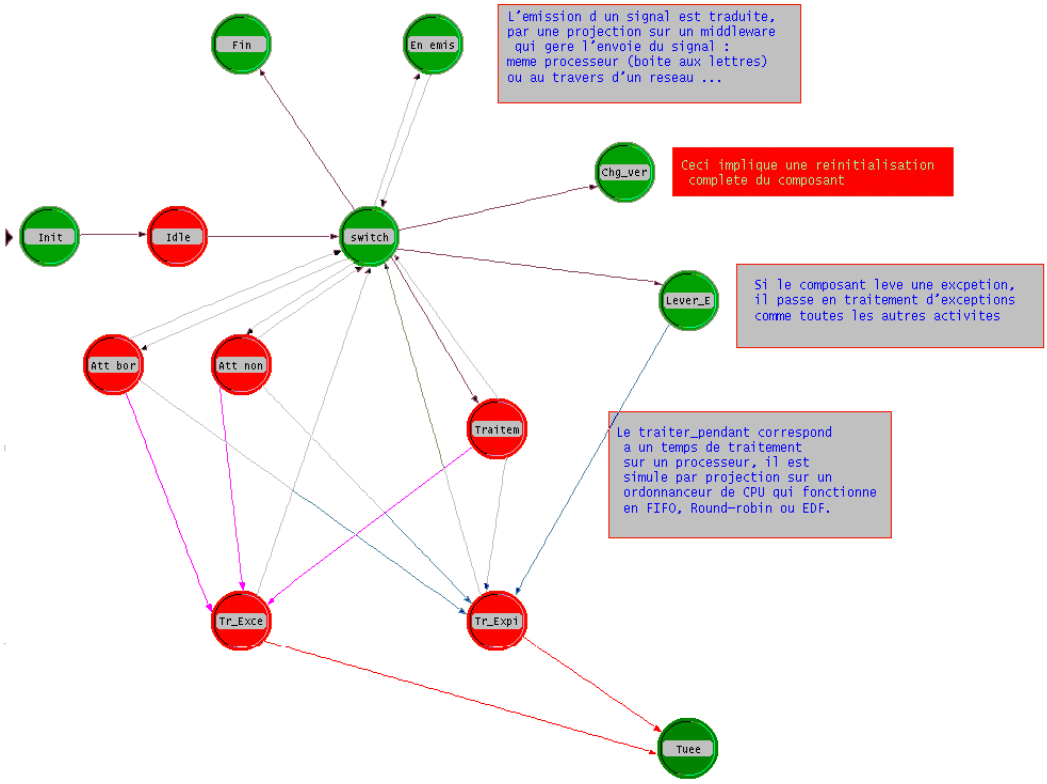


figure 2 Automate de gestion du code déclenché d'une activité

2.1.2 Instanciation des automates au cours de la simulation

Au cours de la simulation, certains automates sont crées à l'initialisation du modèle alors que d'autres sont créés dynamiquement au cours de la simulation en fonction des événements.

A l'initialisation de la simulation, les automates de gestion globale du composant sont instanciés. Il s'agit de l'automate de gestion des déclencheurs, de l'automate gestion des exceptions, de l'automate de gestion des chiens de garde et de l'automate de gestion des événements.
Les autres automates sont instanciés dynamiquement (créés et initialisés en temps simulé nul) et plusieurs versions d'un même automates peuvent coexister ;elles font référence à des activités différentes.

Exemple de déroulement d'une simulation (on ne considère ici que l'architecture fonctionnelle)

Initialisation du modèle d'architecture.
Cette étape consiste à instancier les automates de gestion du composant. Il est nécessaire de récupérer les informations pertinentes pour ceci, à partir de la spécification Ardeco de l'architecture fonctionnelle. On considère à ce niveau que toutes les informations nécessaires sont disponibles dans une structure de données en forme d'arbre respectant la représentation modulaire d'Ardeco (le « parser » associé sera développé plus loin).

Une fois que ces automates sont instanciés le composant est prêt à réagir aux différentes sollicitations et la simulation peut commencer (progression de l'horloge de la simulation).

Les autres étapes, sont mises en œuvre dynamiquement et dépendent à la fois des sollicitations qui sont faites sur le composant et de sa description fonctionnelle.

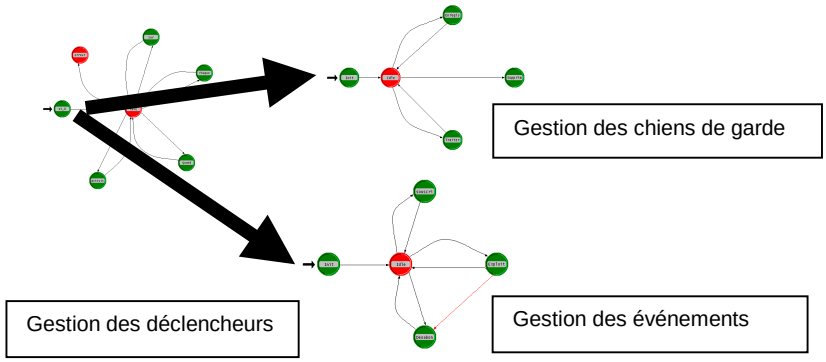


figure 3 Instanciation des automates de Gestion du composant

Exemple de traitement d'un événement au cours de la simulation

- Arrivée d'une sollicitation
Considérons par exemple que le composant possède un déclencheur « sur a » associé à une activité et qu'au cours de son activité il lève une exception. Une sollication arrive sur l'entrée a de l'activité. Cette dernière est prise en compte par le gestionnaire d'événements. Une transition est tirée dans l'automate de gestion du composant (gestion des déclencheurs). L'automate de gestion du composant se réfère à sa structure de données (image de la description en Ardeco) et instancie dans les cas où cela est possible[3] un automate de contrôle de la prise en compte de l'activité. A sa création, cet automate va chercher dans l'image de la description en Ardeco les informations dont il a besoin, en particulier les conditions de retards et les différents délais qu'il peut avoir à gérer. Si nécessaire, sa souscription à un événement ou à un chien de garde est signalé par le tir d'une transition de l'automate de gestion de l'évènement (par exemple, dans le cas où un envoi asynchrone est gardé).

[3] Pour plus de précision sur les conditions d'acceptation de sollicitations, voir [Thomas 99].

Si rien ne s'oppose à la prise en compte de l'activité, l'automate de gestion de la prise en compte instancie un automate de gestion du code déclenché de l'automate. A sa création cette automate récupère dans l'image de la description Ardeco, la suite d'instructions du code déclenché qu'il doit gérer.

- Traitement de l'instruction de levée d'une exception
L'automate de gestion du code déclenché déroule les différentes instructions. Une de ces instructions « lever_exception ». Cette procédure particulière touche le composant dans sa globalité. Toutes les activités créées et non terminées passent en mode d'exception. Ainsi chaque activité instancie un automate de gestion d'exception qui exécute le code qui lui est associé (qui peut être différent pour chaque activité). De même, les activités qui ne sont pas encore prises en compte passent aussi en mode de traitement d'exception. Un automate particulier de gestion de l'exception est activé pour ce composant ; il a pour rôle de synchroniser les traitements d'exception de toutes les activités ; pour ce faire, il vérifie que toutes les activités ont fini leur traitement d'exception avant de faire quitter le mode exception à l'ensemble des activités.

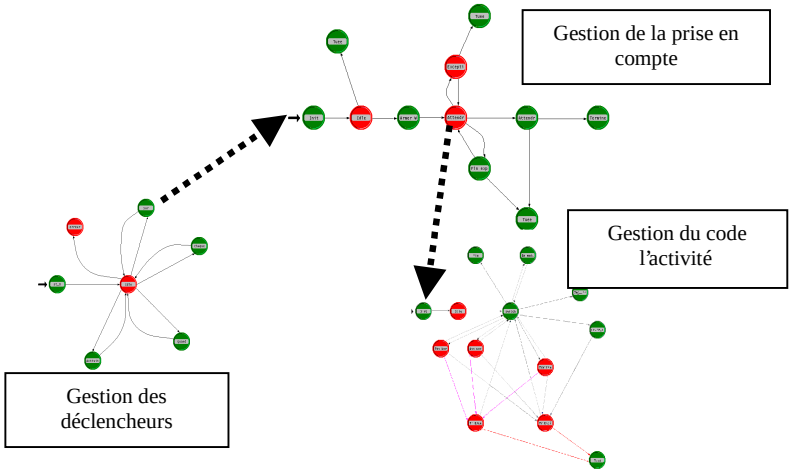


figure 4 Instanciation dynamique des modèles d'activité

2.1.3 Principe de traduction Ardeco vers OPNET

La description en Ardeco possède une sémantique suffisamment précise pour permettre la génération automatique des modèles de l'architecture fonctionnelle (« parsing »). Dans le cadre de la construction du modèle OPNET à simuler, toutes les instructions du langage ne sont pas utiles ; on ne doit exploiter que les informations nécessaires (description des déclencheurs, code des activités …).
La récupération de ses informations nécessitent la mise en place d'un traducteur ou « parser » exploitant la grammaire d'Ardeco. Nous avons généré les analyseurs correspondants grâce à l'utilisation de Lex et Yacc. On peut ainsi récupérer les arbres syntaxiques. Nous avons ensuite spécifié comment utiliser ces arbres pour générer les modèles nécessaires sous Opnet.
Deux mécanismes sont exploitables :
-Utiliser l'arbre syntaxique pour générer directement le code de la simulation en Ardeco en utilisant les possibilités EMA d'OPNET (External model access)
-Utiliser l'arbre syntaxique correspondant au langage d'expression du code des activités lors de la simulation (c'est-à-dire, construire un interprète de code déclenché) à l'intérieur des modèles de la simulation comme une structure de données dans laquelle les différents automates vont chercher les informations dont ils ont besoin.

La solution qui est proposée est d'appliquer une technique mixte, à savoir :
- la génération automatique d'une partie des automates par « parsing » partiel de la description architecturale et de la spécification externe des composants et utilisation des EMA

- la génération des interprètes de codes déclenchés se fera suivant la deuxième (les transactions s'accompagnent alors de la structure de données décrivant le code).

2.2 Prise en compte de l'AM et de la distribution

Nous avons détaillé précédemment comment nous pouvions modéliser le comportement de l'architecture fonctionnelle et exploiter ces modèles au cours de la simulation. Dans ce paragraphe, nous étudions comment prendre en compte les interactions avec une architecture matérielle.
L'analyse du langage Ardeco montre l'existence de deux motifs significatifs. Il s'agit des instructions « traiter_pendant » au niveau d'un composant et des connexions entre composants (point d'accès en sortie vers point d'accès en entrée) au niveau de l'architecture de composants. Nous allons voir où interviennent ces motifs dans les modèles à construire.

2.2.1 Instruction « traiter_pendant »

Les temps de traitement (de type calcul, traitement de l'information…) sont spécifiés en Ardeco à l'aide d'instructions « traiter_pendant ». Ces instructions apparaissent à la suite de la séquence de code décrivant ces traitements (dans le code de l'activité, dans la code lié à une exception ou à une expiration).

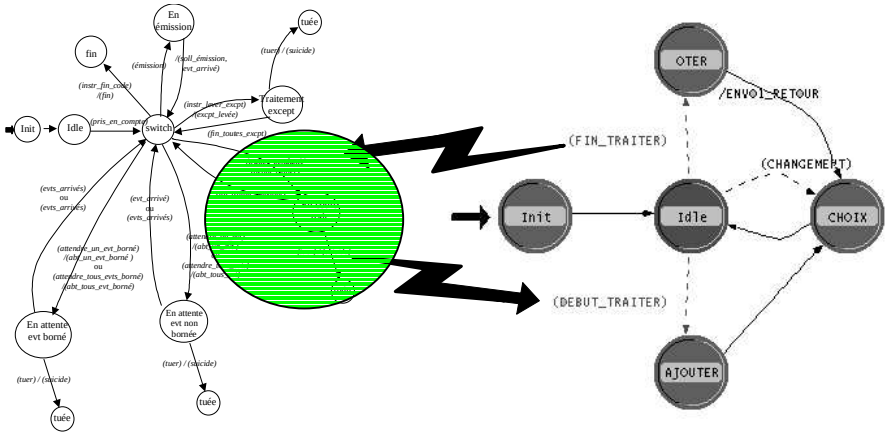
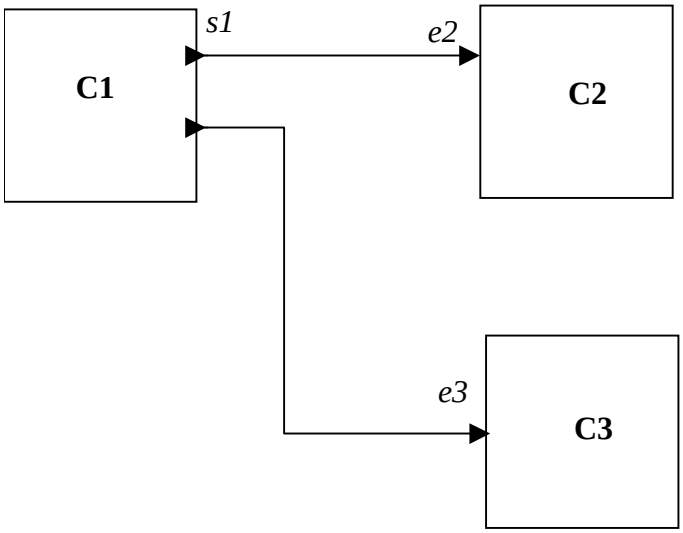


figure 5 Projection de temps de traitement à un ordonnanceur de tâches

Le temps passé en paramètre représente le temps processeur nécessaire pour effectuer le traitement. Il dépend de l'architecture matérielle et du placement (ou projection) du composant concerné sur celle-ci.. La prise en compte de cette instruction et de son paramètre est fondamentale car elle permet , par simulation, d'évaluer les dates de fin de calcul et ainsi d'avoir les éléments pour la vérification des propriétés temporelles du système. Il est donc indispensable de modéliser les mécanismes qui gère l'attribution de la ressource processeur aux différentes activités des composants de l'architecture fonctionnelle. Sur un ordinateur, ces mécanismes sont à la charge du système d'exploitation et plus précisément d'une unité spécifique de ce dernier : l'ordonnanceur. On peut considérer qu'un ordonnanceur de tâches est un gestionnaire d'accès à une ressource non partageable simultanément (le processeur). L'atelier OPNET propose des modules (JSD pour Job Service Discipline) qui permettent de simuler la gestion d'une ressource partageable sequentiellement (section critique) . Nous avons étudié ces modèles pour évaluer l'opportunité de les utiliser.

2.2.2 Les connexions entre composants

Les différents composants sont reliés entre eux par l'intermédiaire de points d'accès en entrée et en sortie . Ces points d'accès permettent la communication entre les différents composants de l'architecture. Le langage Ardeco définit un cadre de communication logique avec une forte notion de typage et l'existence de combinateurs. Notre but dans le cadre du projet est de considérer les retards induits par les communications entre les composants. Une étude menée en collaboration avec l'IRCyN c'est occupé de mettre en place des modèles d'une forme de middleware entre les composants qui prennent en compte les traitements des combinateurs et du typage. Nous ferons ici abstractions de ces surcharges pour nous concentrer sur les retards liées à l'utilisation d'un médium de communication.



C1, C2, C3 : composants

s1 : point d'accès en sortie

e2, e3 : points d'accès en entrée

figure 6 Modélisation des communications spécifiées dans les modèles de l'AF

L'impact de cette communication apparaît naturellement au niveau de 2 automates :

- Au niveau de l'automate qui gère le code déclenché du composant « émetteur » : une des instructions modélisées par cet automate est celle d'« envoi asynchrone », qui correspond bien sûr à l'envoi d'une donnée sur un point d'accès en sortie du composant.
- Au niveau de l'automate de gestion des déclencheurs : voir sur la figure suivante.

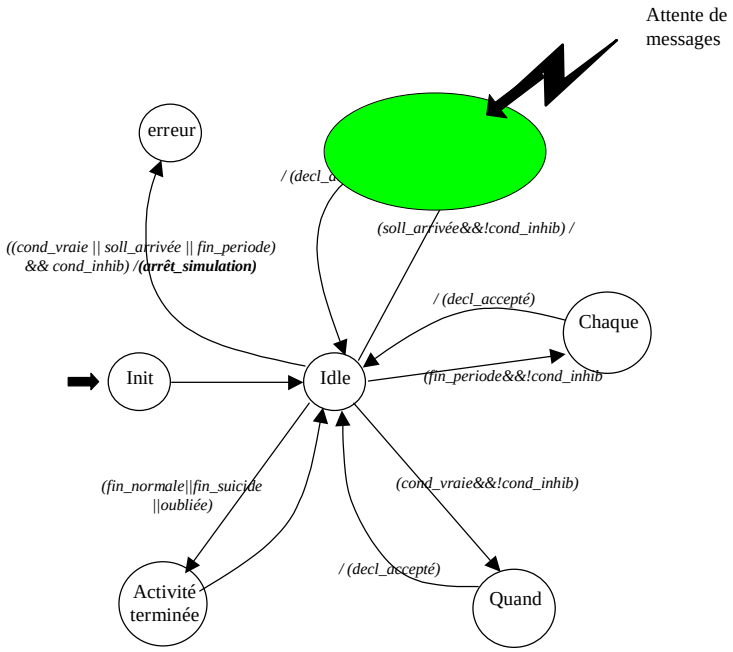


figure 7 projection de l'arrivée d'un message sur l'AG

Si les deux composants sont distants (sur deux calculateurs connectés sur le même réseau), il faut prendre en compte le coût de la communication et, pour ceci disposer d'un modèle décrivant le protocole et les performances du réseau utilisé. Si les deux composants sont locaux, il faut évaluer le temps mis par les services exécutifs pour assurer ce service. Lors de ce stage, nous n'avons pris en compte que le premier cas et nous avons étudié comment mettre en place un modèle de réseau de communication ainsi que l'interaction de ce modèle avec les automates de l'architecture fonctionnelle.

En raison des possibilités de construction hiérarchique des modèles, OPNET est très utilisé dans le domaine de la modélisation et de la validation des réseaux (il permet en effet de simuler l'ensemble des couches selon le modèle OSI). De très nombreux modèles de support de communications sont proposés dans les bibliothèques fournies avec le logiciel Opnet, j'ai évalué les possibilités d'interfacer ses modèles avec nos modèles d'architecture fonctionnelle. Cette étude est faite dans la partie modèle de communications du chapitre 2.

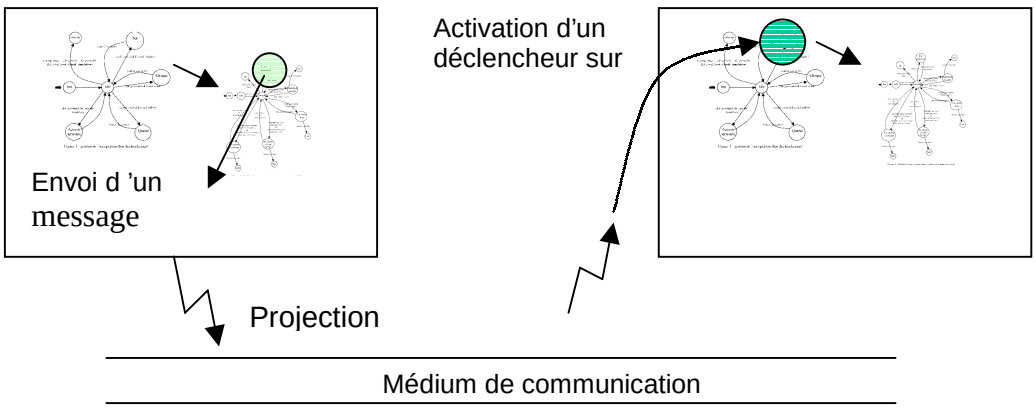


figure 8 projection bout en bout de la communication

3 Modèle des composants de l'AM

3.1 Identification des composants de l'architecture matérielle

Nous avons déjà pu voir qu'il est nécessaire de prendre en compte deux instructions qui apparaissent au niveau de la description de l'architecture fonctionnelle en Ardeco. Il s'agit :

de l'instruction traiter_pendant (qui admet en paramètre un temps de traitement qui correspond au temps de traitement sur une machine qui sert de référence

de l'instruction d'envoi asynchrone. On doit, alors prendre en compte les mécanismes exécutifs qui assurent la propagation de l'événement « émission sur point d'accès en sortie » à l'événement causal « sollicitation sur point d'accès en entrée du composant destinataire. Dans la description en Ardeco, la syntaxe associée aux envois asynchrones correspond au nom du connecteur de sortie . Ainsi un envoi asynchrone sur la sortie b, sera décrit par une instruction « b » qui signifie envoi_asynchrone sur b.

Ces deux instructions constituent les éléments fondamentaux de la projection de l'architecture fonctionnelle vers l'architecture matérielle support pour lesquels les mécanismes exécutifs de l'architecture matérielle doivent être considérés au cours de la simulation. En effet, dans notre cas il ne s'agit pas d'implémenter une architecture opérationnelle (projection de l'architecture de composants décrits en Ardeco sur une architecture matérielle) mais de la simuler pour vérifier ses propriétés temporelles par évaluation de ses performances. C' est pourquoi on ne s'intéresse pas au code en lui même de l'application à tester mais seulement aux éléments qui interviendront aux cours de la simulation. Les traiter_pendant qui apparaissent dans la description correspondent aux temps des traitements d' informations effectués par l'application (remarque : il s'agit d'un temps « hors préemption).

Pour pouvoir simuler l'architecture opérationnelle, il faut donc disposer des modèles de l'architecture matérielle. Pour les traiter_pendants, il faut simuler le serveur de calcul et la stratégie de gestion de ce serveur pour chaque demande de service paramètré par le temps de traitement.
Pour les envois asynchrones, il faut modéliser le mécanisme de communication qui est mis en place entre les différents composants.

3.2 Prise en compte des surcharges dues à l'exécutif

Pour obtenir des informations plus pertinentes au cours de la simulation, il pourrait être intéressant de compléter le modèle de l'architecture matérielle. En particulier, il faut modéliser :

Les surcharges CPU dues à la gestion des objets de l'exécutif [TANNENBAUM98]. En effet, tout exécutif aura crée un certain nombres de tâches (tâches systèmes) dont l'existence influera considérablement sur les temps d'attente des autres tâches (tâche utilisateurs). Une bonne connaissance du taux d'occupation des processeurs par ces tâches systèmes ainsi que d'un modèle de leur comportement (lois d'activation, priorité, …) permettra d'avoir un modèle plus réaliste du comportement des tâches applicatives. En particulier dans l'étude des pires cas, il est alors possible prendre en compte les tâches systèmes de hautes priorités.

Dans le cadre du projet Covadis, on étudie des architectures réparties, pour ce genre d'architecture on dispose en général d'un middleware, qui gère les objets réparties (adresses, services cf [CORBA]) et la synchronisation des horloges ([He 93]), les opérations de gestion d'un tel middleware vont avoir pour conséquence d'alourdir les charges aussi bien au niveau des CPU que des réseaux (qui sont aussi vecteurs de communication des envois asynchrones). Il est également possible d'évaluer cette surcharge et éventuellement de modéliser les tâches et messages assurant ces services.

Pour l'instant ces charges ne sont pas prises en compte mais il serait facilement envisageable de les modéliser et de les inclure dans les mécanismes de simulation d'une architecture.

3.3 Modélisation d'un ordonnanceur de tâche

Le 1er modèle qui a été mis en place correspond aux ordonnanceurs. Il permet de simuler le serveur « processeur » qui gère les différentes files de tâches (systèmes et applicatives) qui sont émis par les clients demandant ses services.

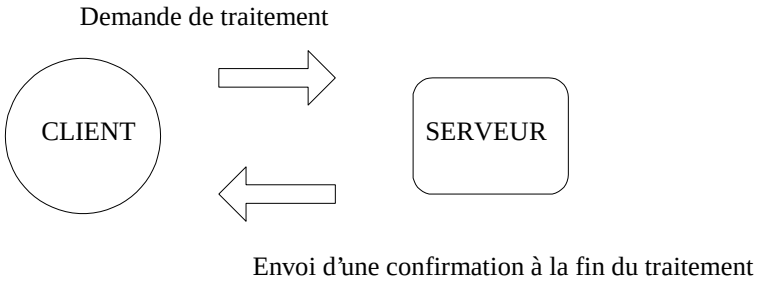


figure 9 Gestion des traitements en Client/Serveur

Nous commencerons par étudier le modèle de processeur des bibliothèques d'OPNET [Garzon95] avant de proposer notre propre modèle.

3.3.1 Le Modèle JSD d'OPNET

Quand un processus a besoin d' une ressource, il émet vers le module JSD un paquet de requête de type JSD_PK qui contient son identité et le temps que prendra la tâche sur cette ressource.
Il attend ensuite un paquet d' acquittement qui correspond à la fin du traitement, il peut alors reprendre son activité. Le temps pendant lequel la ressource est occupée peut être obtenue de 3 manières :

- Directement par le champ svc_time de la requête ;
- Par le champ instructions qui contient le nombre d' instructions nécessaires au traitement. La conversion en temps se fait par l' utilisation de l' attribut processing_rate du processeur (en instructions par seconde).
- Par le champ job_type. Le module processeur utilise alors une table qui lui permet de connaître le nombre d' instructions pour un type de tâche donnée, et de là le temps nécessaire.

On peut ainsi modéliser précisément les temps de traversée de chaque couche. Il faut noter que le temps passé en calcul par le simulateur dans un état ne compte pas du point de vue du déroulement du temps de la simulation.
Le modèle standard fournit de nombreuses statistiques sur l' utilisation du processeur : utilisation instantanée, travail en attente, délai pour chaque tâche...
Description des modèles

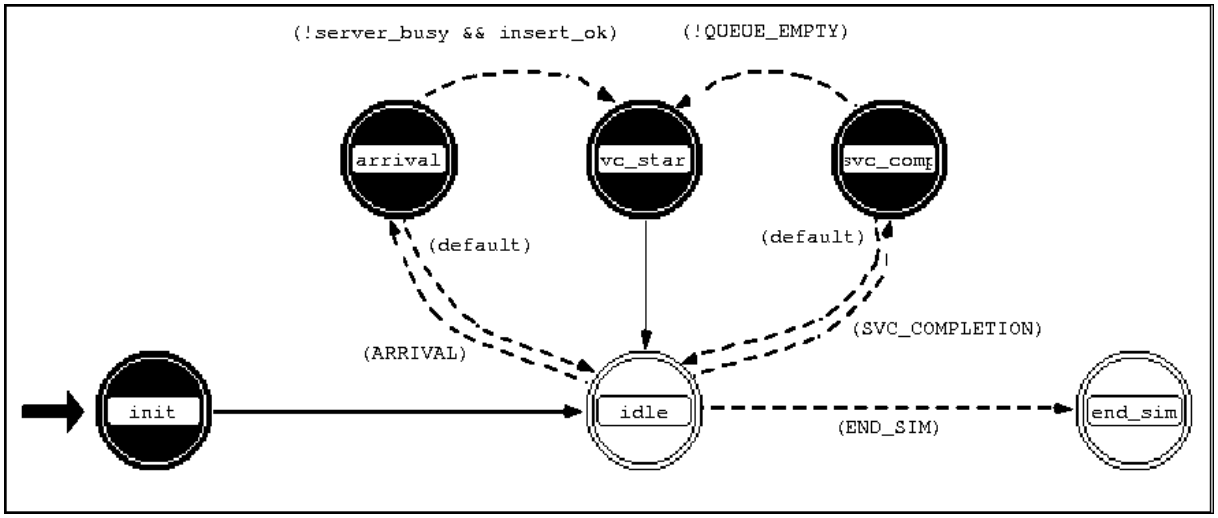


figure 10 Exemple de module processeur : JSD_FIFO

Les différents types de modèles ont les mêmes états, ce n'est que la gestion des files d'attente qui diffère selon le type d'ordonnancement des tâches : simple FIFO, selon la priorité, << le plus court d'abord >>, priorité et mode préemptif...

États :

- INIT :
  - Initialisation des différentes variables, statistiques notamment ;
  - Chargement de la table de description des temps de traitement par type.
- IDLE : Etat d'attente d'un événement. En sortie on met à jour des statistiques sur le temps qu'a passé le processeur dans les différents états *(busy, idle)*.
- END_SIM : Atteint en fin de simulation : Calcul de différentes statistiques scalaires du modèle.
- ARRIVAL : Atteint à l'arrivée d'une requête de demande du processeur.
  - Détermination du temps de traitement à partir des champs de la requête ;
  - Mise en attente dans une file (selon la priorité pour JSD_PRIO). *Remarque: Il y a perte de la requête si la file est pleine (plus de mémoire).*
  - Mise à jour du temps de traitement restant à effectuer dans la file pour statistiques.

  Si le processeur n'est pas occupé, on passe directement à l'état SVC_START (voir schéma).
- SVC_START : Atteint lorsqu'un traitement commence.
  - Programmation d'une interruption sur soi à la date de fin de traitement (On ne défile pas la requête);
  - Mise à jour du flag server_busy à 1, pour empêcher un autre traitement de démarrer.
- SVC_COMP : Atteint sur une interruption programmée par l'état SVC_START.
  - Le paquet correspondant au traitement se terminant est défilé ;
  - Mise à jour des statistiques ;
  - Envoi d'un paquet de fin de traitement au processus qui a effectué la requête (son identité est dans la requête, ainsi qu'un numéro de canal lui permettant d'identifier d'où vient le paquet)
  - Mise à jour du flag server_busy à 0 : une autre requête est alors traitée si la file d'attente n'est pas vide (passage à l'état SVC_START).

Ces modèles permettent de simuler plusieurs stratégies d'accès à une ressource et pourraient être utilisés dans le futur pour gérer des accès disques ou toute autre forme de section critique.
Dans notre cas, et pour simuler tout type d'ordonnanceur, ces modèles se révèlent insuffisants. Il est, en effet, intéressant de pouvoir comparer l'impact de différentes stratégies d'ordonnancement. Or même si certaines stratégies sont modélisables avec les JSD (priorités statiques, par exemple) il est par contre impossible de considérer des stratégies à base de tourniquet (Round-Robin) ou des

mécanismes à priorité dynamique comme la stratégie EDF (Earliest Deadline First). Nous avons donc été amenés à développer notre propre modèle d'ordonnanceur.

3.3.2 Le modèle proposé

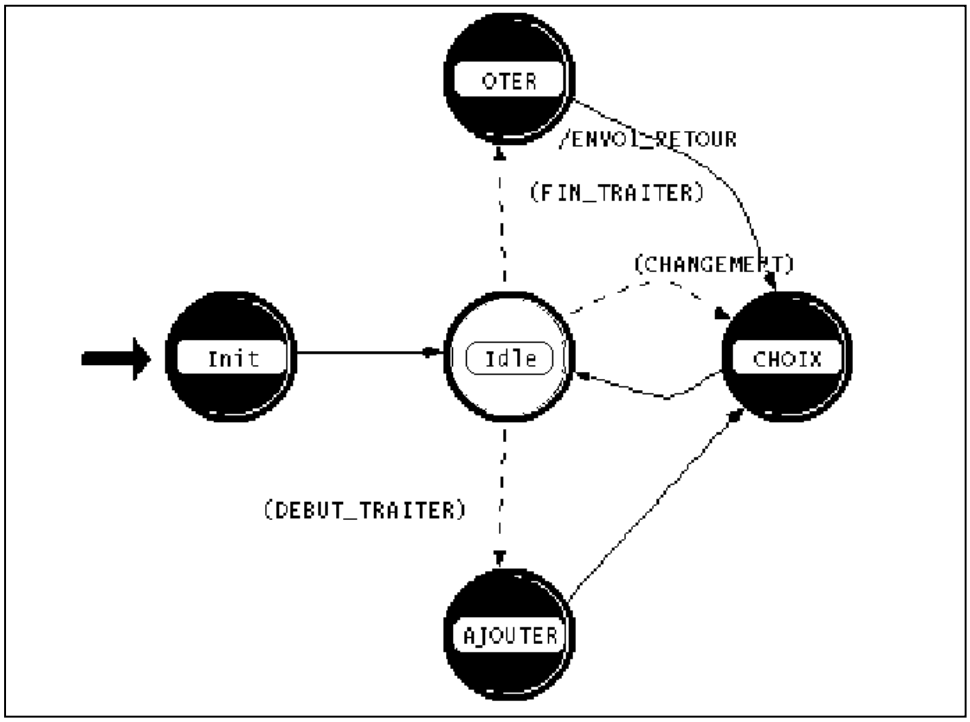


figure 11 Les états de l'ordonnaceur

États :

- INIT :
  - Initialisation des différentes variables, statistiques notamment ;
  - Définition de la stratégie et des paramètres de l'ordonnanceur
- IDLE : Etat d' attente d' un événement, trois transitions sont possibles
  - Debut_traiter correspond à l'arrivée d'une demande de traitement
  - Fin_traiter correspond à la fin d'un traitement
  - Changement permet l'activation de la fonction de choix (nécessaire pour gérer le Round-Robin)

- OTER: Atteint en fin de simulation d'un traitement
  - Mise à jour des données statistiques (temps de réponse,…)
  - Retour à la fonction de choix

- AJOUTER : Atteint à l' arrivée d' une requête de demande du processeur.
  - Détermination du temps de traitement à partir des champs de la requête
  - Prise ne compte des informations nécessaires à l'ordonnanceur (Priorité, échéance …)
  - Mise en attente dans une file (en fonction du critère de l'ordonnanceur)
  - Mise à jour de variables statistiques (date d'arrivée de la requète)

- CHOIX : Cette état correspond à un accès à la fonction de choix de l'ordonnanceur
  - Gère dynamiquement la file des tâches en attente
  - Permet le commutation de tâches (si la stratégie le permet)
  - Gère la mise à jour des données liées à une tâche (mise à jour du temps de traitement restant en cas de changement de contexte)
  - Gère le quantum de temps

Le détail du modèle figure en Annexe

Plusieurs stratégies d'ordonnancement sont modélisées en particulier FIFO, par priorité, round-robin et EDF (Earliest Deadline First) (cf Annexe) La principale difficulté est de formaliser les paramètres de l'architecture fonctionnelle nécessaire à la modélisation de l'architecture opérationnelle. Par exemple pour pouvoir appliquer une stratégie de type EDF, il est nécessaire de connaître les dates d'échéances admissibles des différentes tâches, pour un ordonnancement par priorité, il faut définir la priorité des tâches, tandis que pour appliquer une stratégie RM (Rate Monotonic), il faut connaître la périodicité (ou pseudo périodicité) des tâches.

Au cours de la simulation l'ordonnanceur ainsi mis en place sert de serveur de requêtes pour des demandes CPU (a priori des traiter_pendant définis dans le langage Ardeco). L'envoi des demandes se fait par l'émission d'un événement déclenchant le passage d'une transition. Cette événement est accompagné d'une information et correspond sous Opnet à l'envoie d'un message (Packet)

La structure de l'information contient les éléments suivants :

Une désignation du client (source): Il s'agit d'une désignation de l'émetteur de la requête qui pourra être utilisé directement pour envoyer la confirmation. Le codage des sources est un mécanisme propre à Opnet.
L'identificateur : Chaque requête possède une identification qui lui est propre. Une source n'est en attente à un moment donné que d'une seule confirmation. Mais cette identificateur permet plus facilement de suivre le devenir des traitements sur les traces générées par la simulation. L'identificateur est un entier qui code à la fois la source et un numéro de traitement qui s'incrémente à chaque nouvelle demande (de la même source).
La durée : Il s'agit du temps de traitement que l'on demande à l'ordonnanceur d'exécuter. Il s'agit en particulier de la durée des traiter_pendant définie dans le langage Ardeco.

D'autres champs sont optionnels (en fonction du choix des algorithmes d'ordonnancement)
L'échéance : Dans le cas de l'utilisation d'une stratégie de type EDF, il est nécessaire de fournir à l'ordonnanceur les échéances. Le champs échéance doit alors contenir un réel qui correspond à l'échéance relative (c'est à dire une durée correspondant à l'échéance maximale au moment de l'envoi de la requête).
La priorité : Dans le cas de politique à priorité statique (Rate Monotonic…), il est nécessaire d'indiquer à l'ordonnanceur la priorité de la requète.
La préemption : Dans le cas ou l'ordonnanceur autorise la préemption, il peut cependant être intéressant de l'empecher pour certaines tâches. Ce champ permet donc de définir si l'on autorise la préemption de la tâche.

Le codage d'une demande de traitement est donc le suivant :

Une telle structure est envoyé avec chaque demande de traitement qui arrive à un ordonnanceur.

```
typedef struct
{
double  SOURCE   ;
int     IDENTIFICATEUR  ;
double          DUREE    ;
double  ECHEANCE ;
double  PRIORITE ;
```

```
int             PREEMPTIF;
} TACHE_ENVOIE;
```

3.3.3 Tests des modèles d'ordonnanceur

Pour tester l'ordonnanceur des générateurs de tests ont été mis en place pour envoyer différents types de requêtes : périodique, sporadique, ou suivant une loi de probabilité prédéfinie. Le temps de traitement associé à ses requêtes peut être constant ou suivre lui aussi une loi de probabilité. D'après le type de stratégie utilisé, les paramètres optionnels des requêtes sont utilisés (comme l'échéance pour la stratégie EDF). Pour vérifier le bon fonctionnement de l'ordonnanceur et surtout pour pouvoir obtenir des informations utiles au cours de la simulation, un certain nombre d'informations sont évaluées Ainsi il est possible de suivre le nombre de tâches en attente de traitement, le nombre de changements de contexte au cours de la simulation ou au cours de la durée d'un traitement. Il est aussi possible de connaître l'occupation CPU au cours du temps et bien sûr le temps de réponse à une demande de traitement.

a) Mise en place d'un outil de génération de tests

Dans cette exemple indépendant du contexte COVADIS, on modélise trois sources, correspondant à g0,g1,g2 générant des demandes de services à l'ordonnanceur

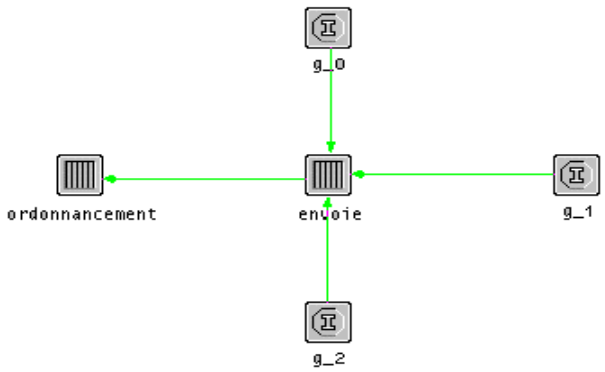


figure 12 Organisation du système de test

Il est possible pour chaque générateur d'indiquer les lois d 'émission des requêtes et de durée des services demandés

b) Exemples de mesures effectués lors d'une simulation

Au cours de la simulation il est possible de suivre un certain nombre de propriété.

Evaluation de l'occupation du processeur

figure 13 Suivi de l'état du processeur au cours du temps

Cette indicateur permet de suivre l'état du CPU au cours du temps (libre ou occupé). Dans la figure 13, on observe l'occupation CPU induite par l'existence de deux sources périodiques, de période différentes mais de durée de service identique.

Evaluation du temps d'attente des taches
Ce temps est calculé comme étant : (date de fin de traitement - date d'arrivée - temps de traitement)

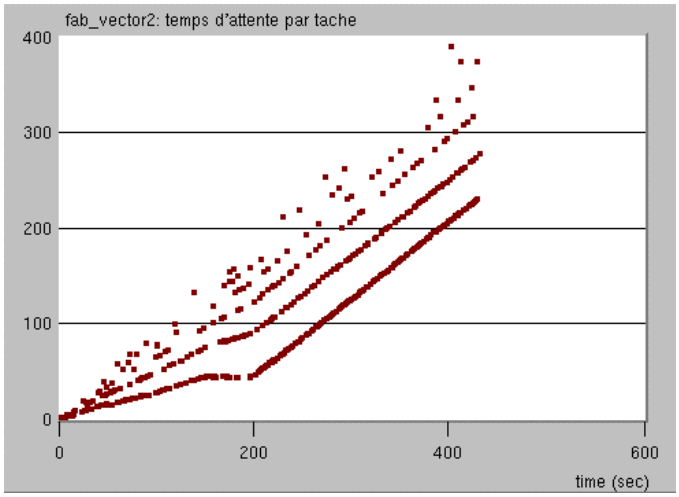


figure 14 Etude des temps d'attente sur un exemple

Cette indicateur permet de suivre les temps d'attentes des différentes tâches. Sur le graphique une tâche est repérée en abscisse par sa date de fin de traitement . Dans l'exemple figure 3, on peut observer les temps d'attente d'une série de tâche crées en rafale entre les instants t=0 et t=180, et dont la durée de service est aléatoire

Suivi des Taches actives

Il est possible de suivre au cours du temps les taches qui occupent le CPU

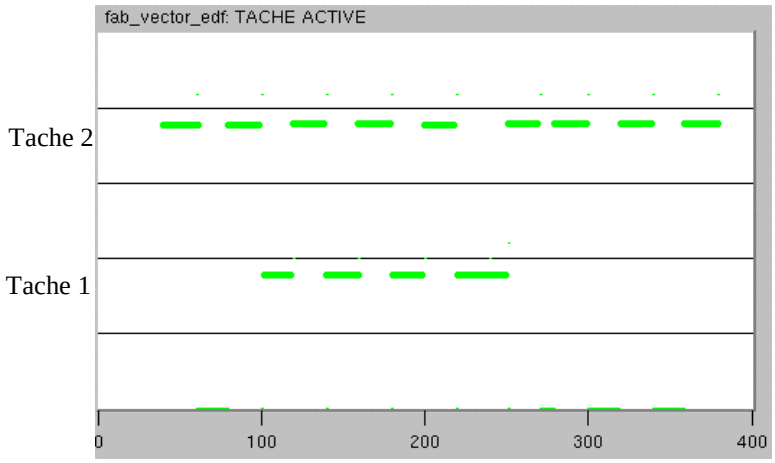


figure 15 Suivi des tâches

Suivi du nombre de tâches en attente

Un indicateur permet de suivre le nombre de tâches en attente de traitement (que l'on appelle tâche prête, par opposition à active et finie) il est possible de suivre indépendamment le nombre de tâche prêtes en provenance d'une source donnée.

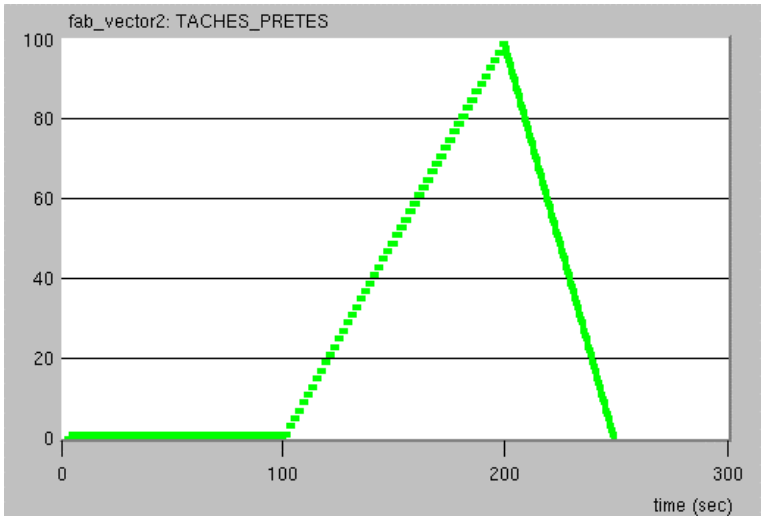


figure 16 Suivie du nombre de tâches en attente

Dans l'exemple de la figure 17, on constate l'augmentation du nombre de tâches en attente de traitement pour une rafale de demandes entre les instants t=100 et t=200 (pour une demande de durée de service constante).

D'autres indicateurs peuvent être mis très rapidement en place , ce qui permet de suivre tout les paramètres qui pourraient se révéler utiles dans l'étude et la validation d'un système ayant à utiliser un tel ordonnanceur.

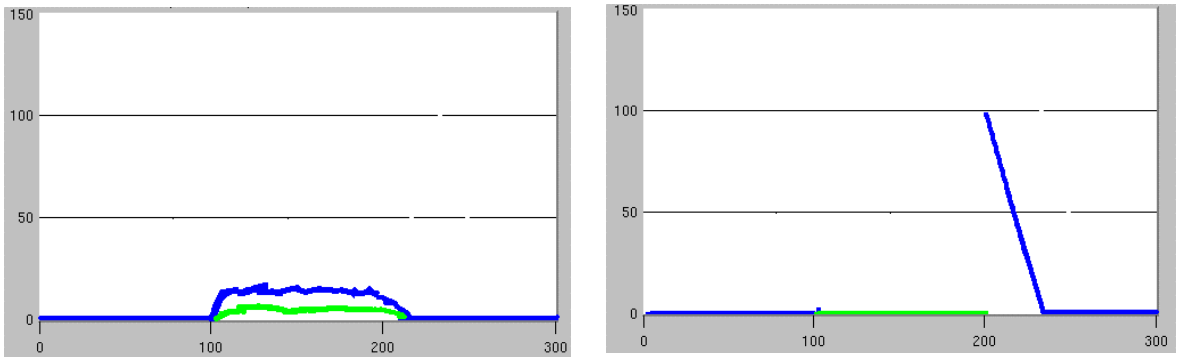
c) Utilisation des résultats de simulation pour la validation et le dimensionnement

Il est possible de suivre la réponse du système pour des stimuli identiques en considérant des stratégies différentes ou en jouant sur les paramètres de ces dernières. Notre outil de simulation nous permet alors de visualiser directement l'impact de ces changements. Ce qui permet d'une part de

valider le fonctionnement attendu du système mais aussi de dimensionner ces composants (capacité mémoire …)

Impact du choix de la stratégie

Dans cette exemple on considère deux sources, la première source émet des demandes en continu et la seconde uniquement entre les instants t0=100 et t1=200. Le but est de comparer l'impact d'une staégie FIFO et d'une stratégie FPP (c'est à dire à priorité fixe où la source 1 possède une priorité plus faible). L'indicateur considéré ici est le temps d'attente. (Remarque : les temps de traitements ont été choisis pour que la source 2 puisse occuper toutes les ressources CPU entre t0 et t1)



FIFO FPP

figure 17 Comparaison des temps de réponse entre une stratégie FIFO et FPP

On remarque que le fait de fixer les priorités permet de mettre à 0 le temps d'attente des requêtes de la source 2 mais qu'en contrepartie le temps d'attentes des requêtes de la source 1 qui ne dépasse pas 20 s en FIFO dépasse dans ce cas 100s

La simulation des stratégies peut permettre de valider le fonctionnement du système, dans le cas précédant même si les taches de la source 2 sont plus importantes pour le système, la stratégie en FIFO permet de limiter l'attente des requêtes de la source 2 (inférieur à 10s) sans pour autant entraîner des retards sur les requêtes de la source 1 qui peuvent être inacceptable

Affinage des paramètres

La simulation peut aussi permettre d'affiner les paramètres d'une stratégie. L'exemple suivant présente les temps d'attentes liées à une rafale de requête en considérant une stratégie de type Round Robin et en proposant 2 paramètres différents pour le quantum de temps.

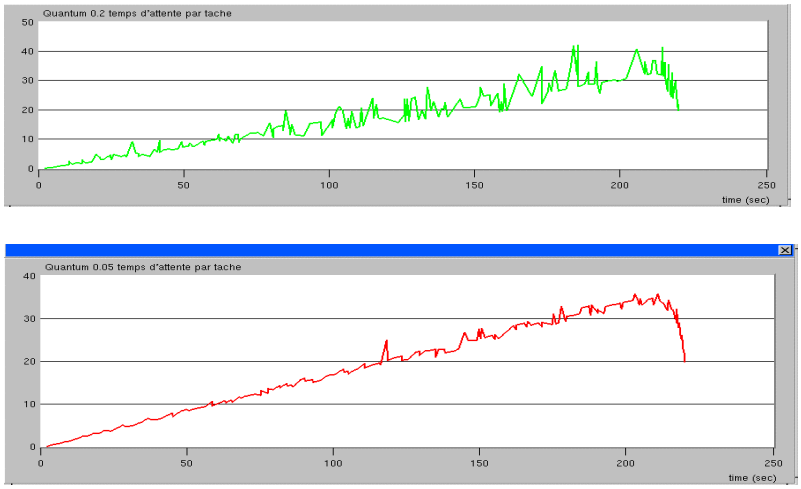


figure 18 Comparaison des temps de réponse pour des quantums de temps différents

On voit par exemple sur cette exemple que le gain au niveau des temps d'attente pour une diminution du quantum de temps de 0,2 à 0,5 s est très faible en moyenne, mais que l'on diminue très sensiblement l'écart type. Le concepteur de l'application peut donc décider de l'intérêt d'une telle modification en fonction de ces impératifs.

Aide au dimensionnent

Le choix d'une stratégie peut influer sur d'autres caractéristiques du système ainsi le nombre de tâches en attente varie entre deux stratégie différente. La connaissance de ces variations permet d'adapter la taille des mémoires (en particulier celle des buffers associés aux tâches en attente)

Sur l'exemple suivant on peut constater les variations induites pour le traitement de la même rafale de requète pour une stratégie FIFO et Round-Robin.

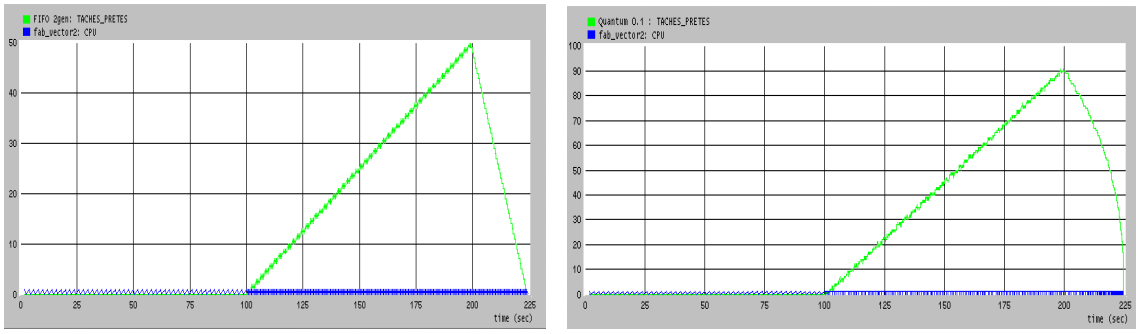


figure 19 Comparaison du nombre de tâches en attente entre une stratégie FIFO et Round-Robin

On constate que le nombre de tâche en attente peut être double à un instant donnée dans le cas de la gestion en Round-Robin et qu'il sera alors nécessaire d'avoir correctement paramètrer le buffer associé.

Ces différents exemples nous montre l'intérêt de modéliser et de simuler l'unité de traitement. Le concepteur de l'application peut valider a priori les différents paramètres de son architecture et optimiser la réponse de son système par les bons choix de stratégies et de paramètres.

3.3.4 Modélisation des systèmes de communications

On s'intéresse à la réutilisation des modèles de réseau existant sous Opnet en vue de les utiliser dans notre simulation d'AO.

Etude d'un modèle de réseau sous OPNET : Le modèle TCP/IP

Description du protocole IP

Si on décompose le protocole Internet en couches comme les modèles OSI, IP assure la couche réseau, juste au-dessus des couches matérielles, comme ethernet. Elle assure un service de transport non fiable en mode non connecté. L' information transportée est structurée en *datagrammes*, dont la longueur dépend des couches inférieures. Ses rôles principaux sont :

- Encapsulation/décapsulation des données vers les couches basses/hautes. Gestion de la fragmentation/défragmentation selon la longueur maximale (MTU, *Maximum Transmission Unit*) du paquet pour un canal donné.
- Routage des paquets selon des tables statiques ou dynamiques (Notion de route par défaut).

Organisation des modules IP dans OPNET
La bibliothèque OPNET de modèles IP se compose de 3 processus distincts organisés selon la figure 20.

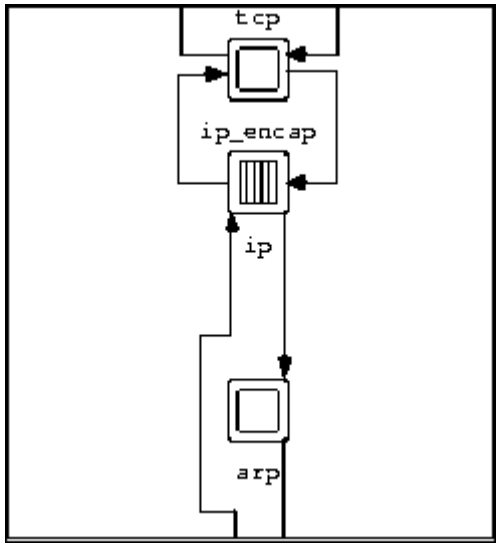


figure 21: Organisation des modules IP

- IP_ENCAP : interface pour les couches supérieures ;
- IP_RTE : effectue le travail de routage et de fragmentation en datagrammes ;
- IP_ARP : Conversion des adresses IP en adresses ethernet (conversion en dur par une table de conversion).

L' idée principale de ce découpage est de fournir un module central banalisé qui gère de la même manière les paquets venant des couches supérieures ou inférieures.

IP_ENCAP
Rôle : Recevoir les paquets de données des couches supérieures et les ICI associés contenant les adresses, créer avec cela un datagramme IP sans le fragmenter, et le passer à IP_RTE qui effectuera le reste du travail. Dans l' autre sens, le travail effectué est exactement symétrique.

États :

- ENCAP : effectue le travail ci-dessus ;

- DECAP : Opération dans l' autre sens : Récupération des infos dans l' en-tête IP, création de l' ICI et envoi des données vers la couche supérieure.

Remarque : IP_ENCAP n' a que deux canaux de communication bi-directionnels : un vers la couche transport (TCP), l' autre vers le module IP_RTE.

IP_RTE
Rôle : Processus central du modèle de la couche IP. Fragmentation, routage des paquets selon une table de routage statique venant d' un fichier. Possibilité de spécifier les routes selon le numéro de réseau... On peut aussi paramétrer une route par défaut.

IMPORTANT : Ce modèle intègre un temps de traitement pour chaque paquet reçu. Il faudra donc modifier son comportement pour qu' il accède au processeur comme les autre modules. Principe : A la réception d' un paquet, on programme une interruption sur soi pour la date de fin de traitement, et on se met en attente de celle-ci. Les paquets arrivant entretemps sont stockés dans une file de type FIFO.
Le traitement effectif du paquet ne se fait qu' à la réception de l' interruption. Ces calculs d' OPNET ne modifient en rien le cours de la simulation, car ils ont une duré nulle dans l' échelle du temps de simulation.

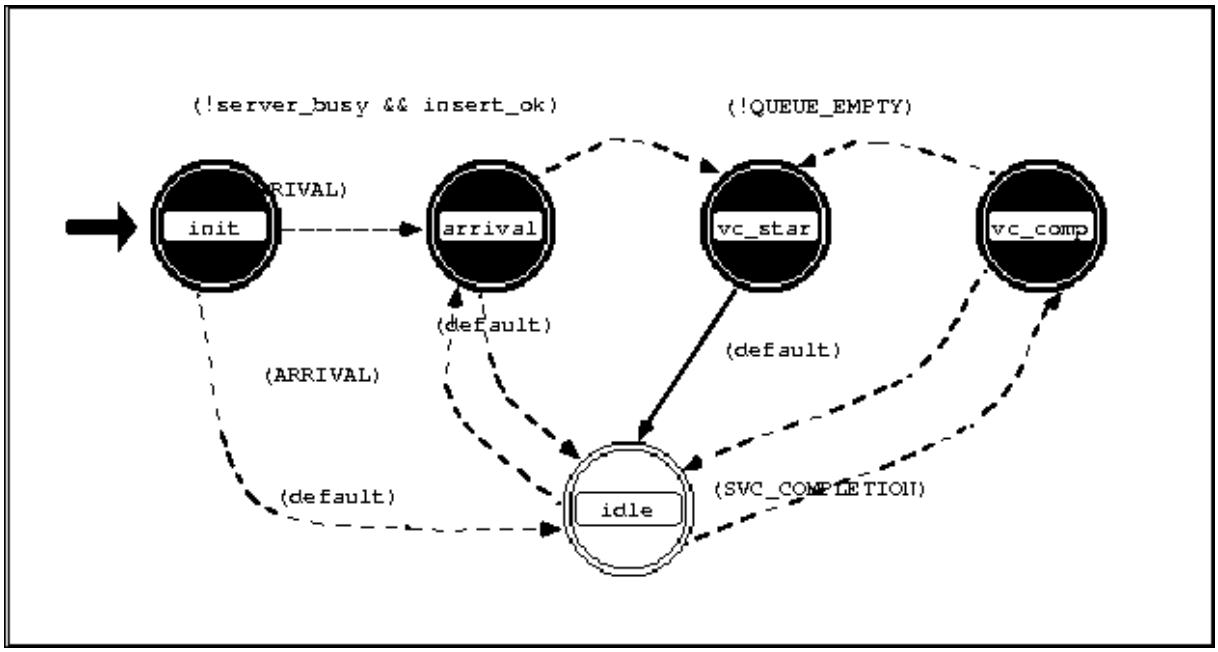


figure 22 Diagramme de transition du modèle IP_RTE

États :

- INIT :
  - Initialisation des variables à partir des attributs du modèle (Adresses du noeud qui le contient...) ;
  - Chargement de la table de routage à partir du fichier dont le nom est spécifié par l' attribut <<route_file>> ;
  - Initialisation de la structure qui contiendra les datagrammes fragmentés attendant d' être <<recollés>>.
  - Création d' un ICI de type IP_ARP_REQ pour communiquer avec le module IP_ARP.
- ARRIVAL : atteint lors d' une interruption d' arrivée de paquet IP ; après ARRIVAL si le processeur n' est pas déjà occupé, à la fin du traitement d' un paquet si la file n' est pas vide.

- Gestion du champ TTL (*Time To Live*) du paquet ;
- Mise du paquet dans la file des paquets en attente de traitement ;
- Positionnement du flag insert_ok pour prévenir au cas où la file est pleine.

- SVC_START : atteint au début du traitement d' un paquet.
  - Programmation de l' interruption qui indiquera la fin du traitement. Calculé grâce à l' attribut service_rate (donné en paquets traités par seconde).
  - Positionnement du flag server_busy à 1 pour empêcher le démarrage d' un autre traitement.
- SVC_COMPLETION : atteint à la réception de l' interruption programmée à l' état précédent.
  - Choix de la route du paquet. La syntaxe de la table de routage permet de traiter comme les autres les paquets destinés au nœud qui contient ce module ou qui sont envoyés par la couche supérieure.
  - Gestion des fragments : si le paquet est destiné au nœud, on gère sa fragmentation éventuelle grâce à un système de listes des fragments déjà reçus, utilisant le système de numérotation des fragments conforme au protocole IP. S' il est destiné à un autre nœud, on utilise la MTU, *Maximum Transmission Unit* de la route à suivre lue dans la table de routage. Il y a fragmentation si la taille totale du paquet - en-tête comprise - est supérieure au MTU considéré.
  - Envoi du ou des paquets par le *stream* indiqué par la table de routage. Un paquet ICI est aussi envoyé pour indiquer au module ethernet l' adresse IP et le réseau où envoyer le paquet.

IP_ARP

Rôle : Effectuer la conversion entre adresse logique - IP - et physique - Ethernet -. La conversion se fait en dur grâce à une table statique venant d' un fichier.
Si la conversion n' est pas possible, le paquet est détruit avec émission d' un avertissement.
Description du protocole TCP
Le protocole TCP se trouve dans le modèle en couches entre IP et les applications. C' est la couche transport du modèle OSI. Il permet d' instaurer un canal de communication fiable full-duplex entre deux applications distantes. Ce protocole est orienté connexion, en ce sens qu' il particularise les deux acteurs de la communication : d' un côté le serveur, qui ouvre une connexion passive, de l' autre le client qui effectue vers le serveur un requête de connexion. Une fois la communication établie, les 2 acteurs jouent dependant le même rôle.
On appelle souvent une connexion TCP une socket. Elle a le comportement vu de l' utilisateur d' un pipe Unix, i.e. un *STREAM* (flux) dans lequel l' utilisateur écrit et lit un flot de caractères comme dans un fichier.
Une connexion est identifiée de chaque côté par un numéro de port, qui permet notamment d' identifier le type de service du serveur (Mail, transfert de fichier...).
TCP garantit notamment :

- la reprise sur erreur ;
- l' ordre d' envoi des données ;
- le mode de transfert full-duplex.

Les modules OPNET du protocole TCP

Le modèle TCP fourni par OPNET est constitué de deux modules distincts :

- TCP_MANAGER : Module principal de la couche, c' est avec lui que communique les applications. Il gère l' établissement des connexions, la réception des paquets...
- TCP_CONN : Module gérant une connexion (socket). C' est un processus créé dynamiquement par TCP_MANAGER, qui lui passe ensuite le contrôle pour traiter les commandes correspondant à la socket qu' il gère.

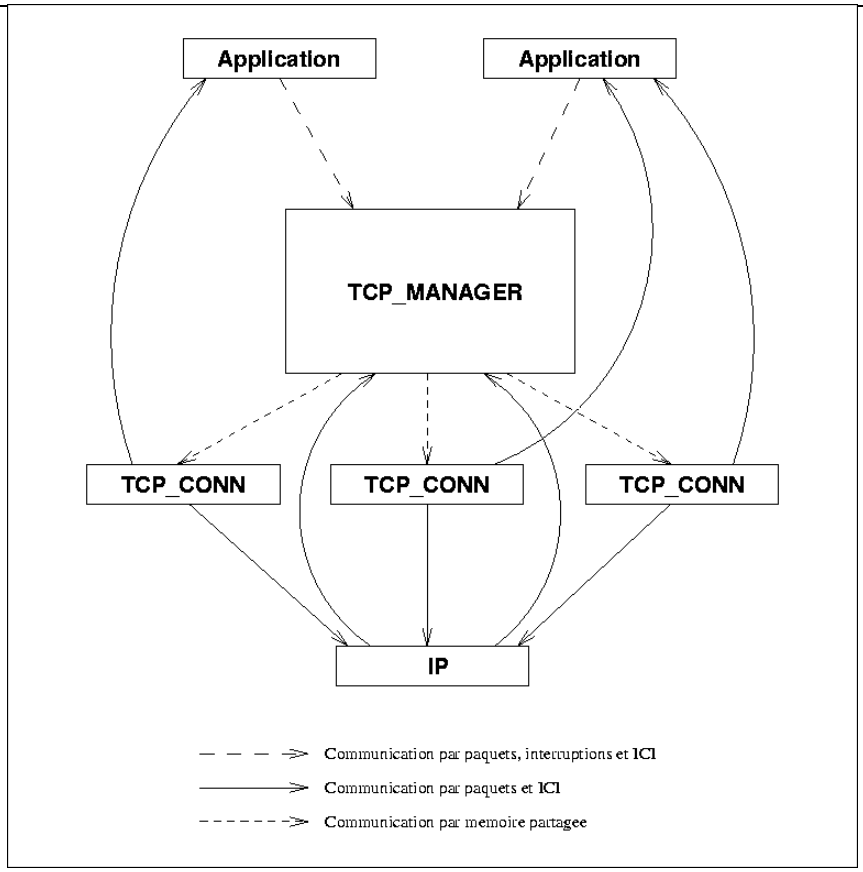


figure 23 Diagramme de connexion de la couche TCP

Implémentation de TCP sous OPNET

Conformément à la norme, OPNET implémente les fonctionnalités suivantes du protocole :

Garantie de la sûreté de la connexion par un système d' acquittement et de retransmission des données en cas de perte de paquets dans les couches basses.

Les délais de retransmission sont recalculés dynamiquement à partir des temps moyens de parcours des paquets pour une connexion donnée

Contrôle du flux de données par scrutation chez le nœud distant de la taille restante dans le tampon de réception : À chaque fois qu' un paquet est envoyé, l' en-tête TCP contient la place restante dans le tampon de réception, permettant au module distant de différer l' envoi d' un paquet trop long. Au bout d' un certain temps (Persistence Timeout), celui-ci envoie un paquet de un octet de long, qui a toutes les chances de tenir dans le tampon de réception, et qui lui permet de connaitre par le paquet d' acquittement la nouvelle taille de la fenêtre de réception.

Interface avec la couche application :

Le protocole TCP fonctionne du point de vue des couches supérieures par un ensemble de commandes, implémentées dans OPNET par des interruptions vers le module TCP_MANAGER. Celles-ci sont soit déclenchées par l' envoi d' un paquet *(Stream Interrupt)* soit explicitement envoyées *(Remote Interrupt)*. Elles sont accompagnées d' un ICI permettant de passer des paramètres. les réponses de la couche TCP vers les applications se font exactement de la même manière. Les commandes sont les suivantes :

- OPEN : Pour ouvrir une connexion. Il y a deux types d' ouverture :
  1. OPEN_ACTIVE, pour laquelle un paquet de demande de connexion est envoyé à la socket distante. L' ICI doit contenir les informations nécessaires à l' établissement de la connexion : Adresses de réseau et de nœud, numéro de port.

2. OPEN_PASSIVE, la socket locale attend alors que ce soit le nœud distant qui demande l' ouverture de la connexion.

- SEND : Pour envoyer un paquet (correspond à une << stream interrupt >>)
- RECEIVE : interruption sur le module TCP_MANAGER pour demander à recevoir des paquets, dont le nombre est spécifié dans l' ICI.
- CLOSE : Demande de fermeture de la connexion. Les paquets restant dans le tampon d' émission sont envoyés.
- ABORT : Fermeture rapide, les paquets restants ne sont pas envoyés.
-

Les modules TCP_MANAGER et TCP_CONN

Le modèle TCP d' OPNET repose sur le module TCP_MANAGER, qui gère l' interface avec les autres couches et un ensemble de connexions (sockets), chacune étant gérée par un module TCP_CONN. Lorsque le TCP_MANAGER reçoit des commandes des applications ou un paquet de la couche inférieure, il effectue les traitements nécessaires (création d' un processus TCP_CONN lors de l' établissement d' une connexion...) puis passe la main au processus qui gère la connexion qui est concernée par l' événement reçu.
C' est donc un processus de modèle TCP_CONN qui effectue le travail particulier pour chaque connexion : envoi ou attente d' acquittement,encapsulation des données en paquets TCP... La communication entre le processus principal et les processus TCP_CONN se fait par un bloc de << mémoire partagée >>, que le TCP_MANAGER passe en argument au moment de l' activation d' un processus TCP_CONN. Ce bloc contient notamment le type de commande à effectuer, le paquet reçu de la couche supérieure...

TCP_MANAGER

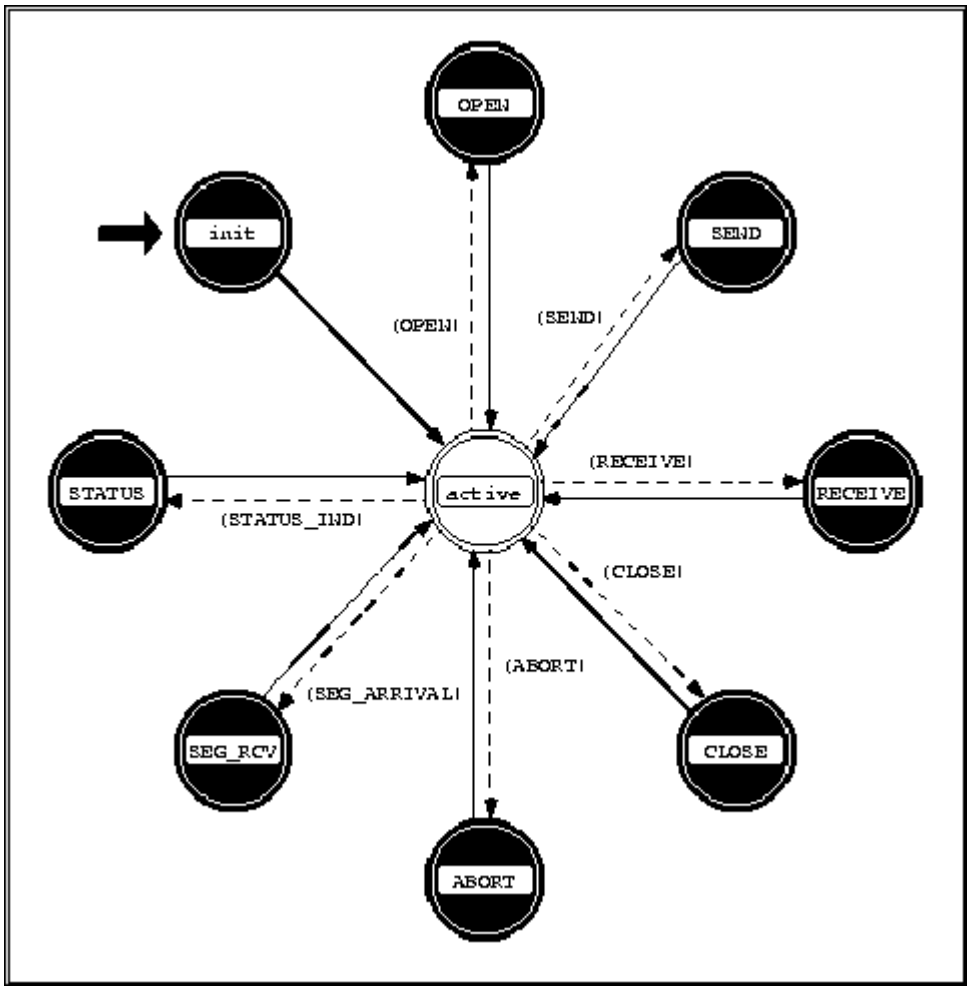


figure 24 Diagramme d' état de TCP_MANAGER

Comme on peut le voir sur le schéma 25, ce processus possède un état pour chaque commande TCP, plus un pour gérer la réception des paquets (SEG_REC).
Dans chacun de ces états, TCP_MANAGER retrouve dans la liste des processus TCP_CONN qu' il a créés celui qui gère la connexion en question, renseigne les champs utiles du bloc de mémoire qu' il partage avec lui puis lui passe le contrôle.

TCP_CONN

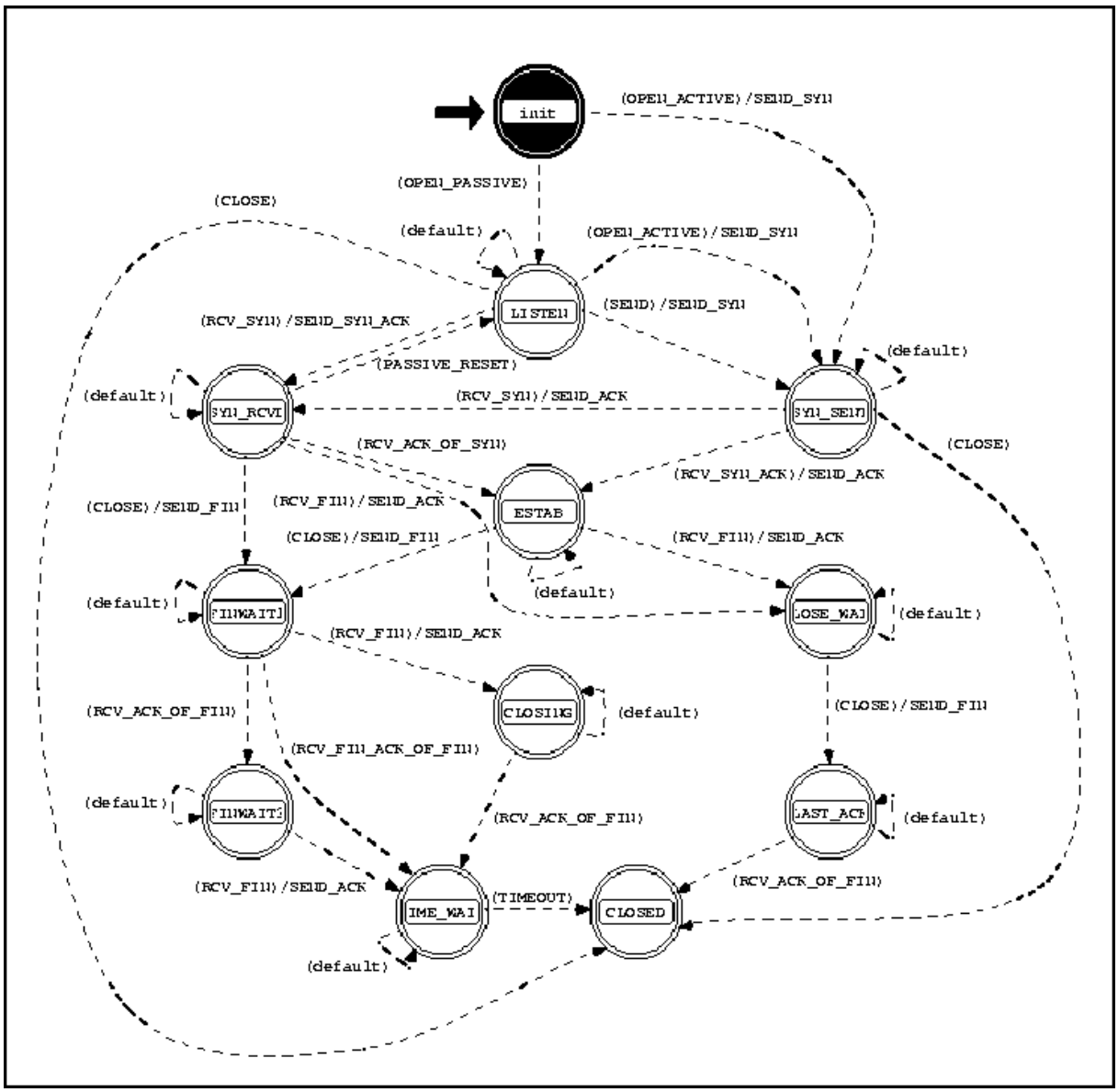


figure 25 Diagramme d' état de TCP_CONN

C' est la partie principale du module TCP. Sa relative complexité vient du fait de la gestion des différents acquittements et timers que demandent les impératifs de fiabilité de la connexion. Comme on le voit sur le schéma 26, la réception d' une commande, qui correspond à un changement d' état, entraîne pendant la transition un envoi d' acquittement.
Exemple : pour la transition LISTEN SYN_RCVD, qui correspond à la réception d' un paquetSYN (requête d' établissement de connexion), on envoie à la transition un acquittement qui confirme la

réception de ce paquet (Macro SEND_SYN_ACK après le / sur la description de transition (RCV_SYN)/SEND_SYN_ACK).

États :

- init : État de départ. Toutes les initialisations sont effectuées. Si le processus a été invoqué par une commande OPEN_PASSIVE, le processus entre dans l'état LISTEN, si c'était une commande OPENE_ACTIVE, il envoie un paquet SYN de demande d'établissement de connexion et entre dans l'état SYN_SENT.
- LISTEN : Le processus attend une requête SYN. Si une est reçue, il passe dans l'état SYN_RCVD en envoyant un acquittement. Si une commande SEND ou OPEN_ACTIVE est reçue, une requête SYN est envoyé : cela signifie qu'un processus veut envoyer des données ou ouvrir une connexion active en utilisant le port local que gère ce processus. Il passe alors dans l'état SYN_SENT.
- SYN_SENT : Le processus attend un paquet SYN. Si celui qui arrive est l'acquittement de celui qui a mis le processus dans cet état, il passe dans l'état ESTAB en avertissant l'application que la connexion est établie. Sinon, il envoie l'acquittement correspondant et entre dans l'état SYN_RCVD.
- SYN_RCVD : Le processus attend l'acquittement de sa propre requête de connexion. S'il arrive, il passe dans l'état ESTAB et avertit l'application que la connexion est ouverte (par un ICI). Si une commande CLOSE arrive, une requête de fin de connexion (FIN) est envoyée et il entre dans l'état FIN_WAIT1.
- FIN_WAIT1 : Le processus attend une requête de fin de connexion (FIN) ou l'acquittement du paquet FIN envoyé précédemment. Si l'acquittement est reçu, le processus passe dans l'état TIME_WAIT. S'il reçoit une requête FIN qui n'est pas l'acquittement escompté, il rentre dans l'état CLOSING en envoyant un acquittement pour cett requête. Si le processus reçoit un paquet qui n'est pas de type FIN mais qui acquitte sa requête FIN précédente, il rentre alors dans l'état FIN_WAIT2.
- FIN_WAIT2 : Le processus attend une requête FIN, après quoi il rentre dans l'état CLOSE_WAIT.
- CLOSE_WAIT : Le processus atend une commande CLOSE. Si elle est reçue, il envoie une requête FIN et entre dans l'état LAST_ACK.
- LAST_ACK : Le processus attend l'acquittement de la requête FIN envoyée plus tôt. À son arrivée, il passe dans l'état CLOSED.
- TIME_WAIT : Le processus attend suffisamment longtemps pour que le dernier paquet d'acquittement de fin de connexion soit arrivé. La durée d'attente est de deux fois la durée de vie maximale d'un paquet de la couche physique sous-jacente. Une fois cette durée écoulée, Le processus passe dans l'état CLOSED.
- CLOSED : Le processus TCP_CONN se termine, en envoyant un ICI pour prévenir le TCP_MANAGER et l'application de la fin de la connexion.

Interfaçage du modèle TCP/IP d'Opnet avec notre modèle d'architecture fonctionnelle.

Notre but est d'utiliser ces models pour simuler les propriétés temporelles de l'architecture opérationnelles dus à l'existence de communications entre les différents composants. Les temps générés par le modèle TCP/IP sont multiples ; on trouve des temps de propagation, des temps d'émission et de réception (liés à des traitements ou à des recherches de chemin de routage) ou encore des temps liés à des mécanismes de réémission sur erreurs. Dans le cas des temps d'émission et de réception, il s'agit de temps de traitement processeur que l'on doit déléguer à un JSD et que l'on pourrait faire traiter à notre propre ordonnanceur pour prendre en compte les surcharges du processeur liés à la gestion et à l'utilisation du réseau.

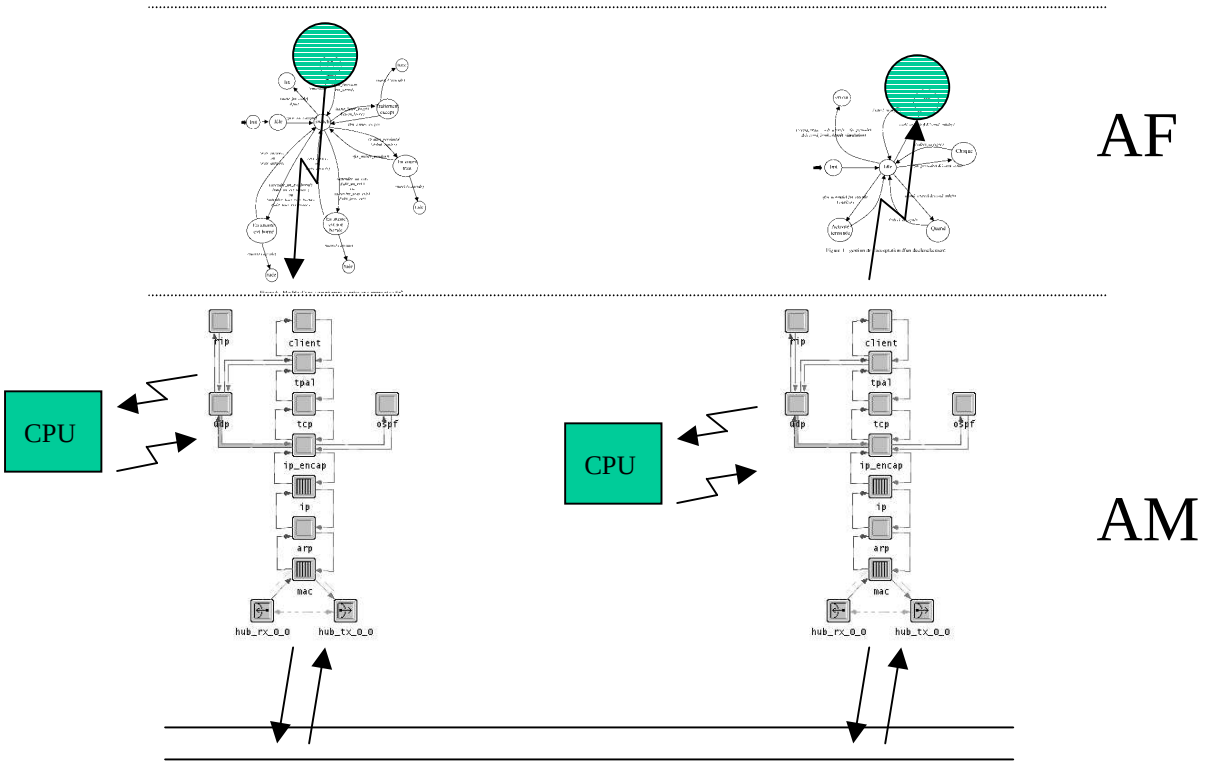


figure 26 Utilisation du modèle TCP/IP dans notre simulation

L'utilisation du modèle TCP/IP dans le cadre de notre simulation se révèle possible mais le nombre très élevé des paramètres à identifier vis à vis de l'architecture opérationnelle empêche son utilisation sans considérer un travail de projection du composant très complexe. De plus comme les messages envoyés par nos composants sont à priori de petites tailles tout les mécanismes d'encapsualtion/désencapsulation de TCP/IP ne nous sont d'aucune utilité et compliquent considérablement le modèle.

En conclusion, le modèle TCP/IP présente peu d'intérêt dans le cadre de notre simulation car son utilisation se révèle trop complexe et le modèle est trop riche vis à vis de nos besoins.

Notre modèle de communication

L'envoi de messages sur un support (médium de communication) s'apparente à l'utilisation d'une section critique. L'utilisation des JSD d'opnet (cf le modèle JSD d'opnet) est tout à fait approprié. En effet est possible d'envoyer au JSD, une requète contenant la source, le destinataire et la taille du message à envoyer, celui ci pourra alors à l'aide d'une d'une table de conversion en interne obtenir le temps de traitement équivalent et le simuler. Il faut cependant lui fournir la table de conversion qui pourra prendre si nécessaire en compte les délais de propagation (à priori négligeable dans les systèmes embarqués) mais surtout proposer un délai de transmission (fonction de la taille du message, du type de réseau et de son débit).
Il reste encore à pouvoir interfacer ce modèle avec notre modèle d'architecture fonctionnelle., Il faut pour cela utiliser un automate supplémentaire qui gère le transfert logique de la communication. Effet le JSD renvoie forcement la confirmation du traitement (qui correspond à la simulation du temps du traitement) à l'auteur de la demande. Dans notre cas, la demande est émise par l'émetteur mais doit être reçu par le récepteur. C'est pourquoi on utilise un automate qui se charge d'envoyer un le message au récepteur une fois que le JSD a simulé le temps de traitement et lui a envoyé la confirmation. Cette automate est crée dynamiquement pour gérer un envoi asynchrone donnée)

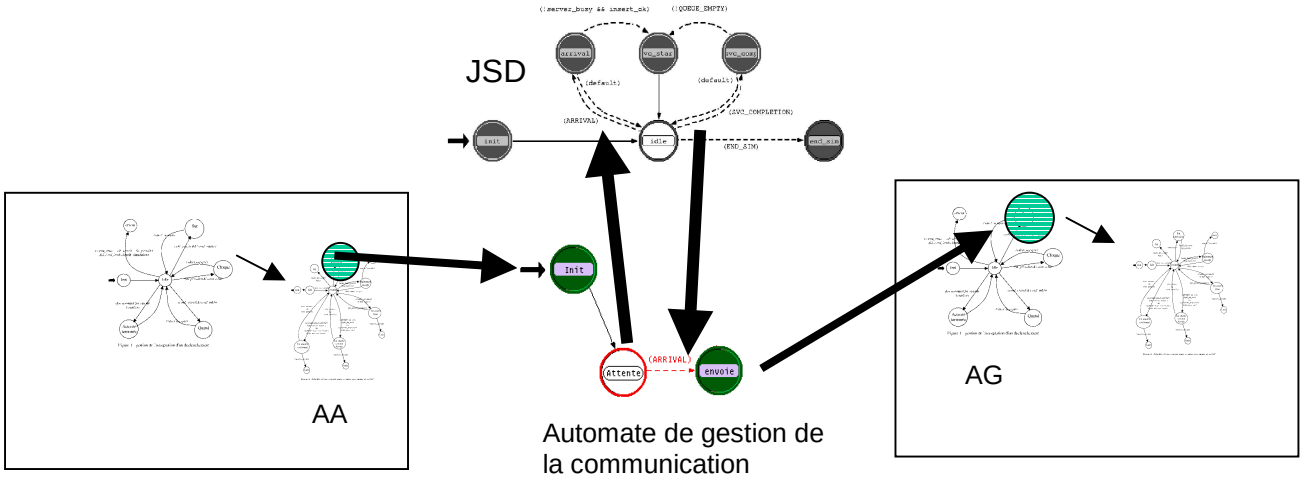


figure 27 Projection des communications JSD

Nous proposons un modèle de communication qui permet de simuler la gestion du médium de communication en permettant un certain nombre de stratégie d'accès à ce dernier (la gestion de la file d'attente du JSD permet en effet une gestion par priorité). Il est ainsi possible de simuler les communications en prenant en compte les caractéristiques propres du réseau considérées (débit, mode de gestion du MAC …). Le modèle proposé contrairement au modèle des bibliothèques d'opnet ne permet cependant pas de prendre finement en compte les surcharges liés à l'utilisation du réseau ou encore de générer des erreurs de transmission (et donc des mécanismes de réemission). Si la simulation de tel phénomène était cependant nécessaire cela se révélerait modélisable en ne modifiant que l'automate de gestion de la communication. Ce modèle est en cours d'implémentation.
Les informations suivies au cours de la simulation seront principalement les temps de transfert bout en bout des messages mais aussi les temps d'attentes et le nombre de messages en attentes de transmission à un moment donné.

4 Spécification d'une architecture matérielle et d'une distribution

Pour pouvoir fixer les propriétés de l'architecture matérielle, il faut nécessairement disposer d'un langage de description de cette dernière. De plus, comme un certain nombre de paramètres doivent être fixés au moment de la projection, il faut aussi disposer d'un langage qui donne les différents paramètres de la projection. Il est donc nécessaire d'enrichir le langage Ardeco avec un langage de description d'architecture matérielle.

4.1 Le langage de description d'architecture matérielle

4.1.1 Définition des sites

La notion élémentaire est celle du site. Un site est un calculateur, d'architecture monoprocesseur, multiprocesseur sur bus ou autres (automates programmables …). Dans le cadre de notre étude on ne s'intéressera qu'à des sites correspondant à des monoprocesseurs mais la prise en compte d'autres types d'architectures ne pose aucun problème.

Un site possède un certain nombre de caractéristiques, les informations considérées comme nécessaires pour être utilisés dans notre simulation sont les suivantes :
Le type d'ordonnanceur utilisé
La capacité du processeur (nombre de tâches pouvant être prises en compte à un moment donné)
La puissance (Facteur multiplicatif par rapport à la définition du traiter_pendant sur une machine de référence).

Un site est muni d'un système d'exploitation (ensemble de tâches gérant les ressources matérielles du site pour les besoins de l'application. Chaque utilisation de l'exécutif crée une surcharge (overhead) sur un site (occupation du processeur par les tâches systèmes, occupation mémoire …). L'ANNEXE 3 donne des moyens de la quantifier.

4.1.2 Définition du réseau

Le deuxième élément de l'architecture matérielle que nous avons modéliser est l'architecture de communication. Il est possible d'envisager deux méthodes pour décrire le choix du réseau.

Désignation directe du modèle

Une fois que le modèle d'un type de réseau a été créé (par exemple réseau CAN), il suffit de faire référence directement à ce modèle en désignant son type et les paramètres propres aux modèles (pour CAN ce pourraient être le débit.). Ces renseignements seront utilisés pour créer la table de conversion du JSD associé à un médium de communication

Définition des propriétés du modèle de réseau

Il pourrait être tout aussi intéressant de pouvoir décrire les caractéristiques directement au niveau du langage: Débit, nombre de sites possibles, propriétés topologiques, vitesse de transmissions, modèle des pertes …). Il peut aussi être intéressant de prendre en compte la surcharge de la CPU des machines connectées pour les envois/réception de messages. Cette méthode peut paraître intéressante mais dans le cadre de notre étude, un des élément les plus importants du réseau, dont va dériver les propriétés temporelles est la gestion du MAC. Or la description du fonctionnement du MAC est souvent trop complexe pour pouvoir être décrite simplement. L'utilisation d'une méthode de

description des caractéristiques n'est donc possible que dans les cas où la modélisation du MAC est très simple.

Problème lié à l'utilisation du réseau

La projection sur le réseau apparaît principalement lors des envois asynchrones de messages définis dans Ardeco. Il est donc nécessaire de fixer une taille à ces messages pour pouvoir obtenir un temps de transfert au cours de la simulation.

4.2 Modélisation de l'AO

Maintenant que nous disposons d'un moyen de décrire l'architecture matérielle, encore faut-il disposer d'un formalisme de description de la projection de l'AF sur l'AM l'architecture fonctionnelle. Pour cela il est nécessaire de pouvoir définir la situation topologique de certains composants et de pouvoir fixer les paramètres des fonctions de placement des composants sur les sites.

4.2.1 Projection d'un composant sur un site

Il faut avant tout définir le nombre de sites et fixer chaque composant sur un site. De nombreux travaux concernant les algorithmes de placement ont été mené.[Beauvais 99] on considérera dans notre cas que ces fonctions de placement existe et que la répartition des composants sur les différents sites est établie.

Chaque site est définit dans le langage de description d'architecture matérielle exposé au chapitre 2 (en particulier on définit la stratégie de l'ordonnanceur). Il reste encore à pouvoir fournir à l'ordonnanceur les informations relatives aux tâches qu'il gère et donc en particulier leurs priorités (pour les stratégies à priorité statique qui sont les plus couramment utilisées)
Par exemple, pour fixer les priorités, on peut envisager deux méthodes ; on peut donner une priorité à chaque composant et donner cette priorité à toutes les tâches issues du même composant ou définir une priorité pour chaque activité (chaque dt …, sur a …, sur b)

4.2.2 Projection des échanges entre composants sur les systèmes de communications

Pour relier les sites au réseau de communication, il suffit simplement de sélectionner un ensemble de site et d'indiquer qu'ils appartiennent à un réseau dont la description est expliquée dans la partie architecture matérielle. Bien entendu un site peut être relié à plusieurs réseaux, il faut cependant faire correspondre chaque entrée ou sortie du composant à un nœud du réseau. La correspondance entre les entrés ou sorties d'un composant et les nœuds du réseau est implicite par la spécification de connexion des composants au sein de l'architecture fonctionnelle. Ainsi au cours d'un envoi asynchrone, l'envoi du message se fera sur le médium de communication désiré. Le choix d'un réseau de communication peut nécessiter de donner des caractéristiques supplémentaires au message. Ainsi dans le cas du réseau CAN, il est nécessaire de fixer la priorité d'un message. Il faut donc comme dans le cas de l'ordonnanceur de disposer de règle pour fixer ses priorités ou de donner à l'utilisateur la possibilité (ou l'obligation) de le faire. Un JSD particulier (cf Notre modèle de communication) est associé à chaque médium de communication et sera crée à l'initialisation du système

4.3 Evaluation des temps de traitement

Le langage Ardeco distingue le traitement des informations et le temps d'occupation du processeur nécessaire pour faire ces traitements.
Ainsi une spécification d'activité en Ardeco est construite comme dans l' exemple qui suit :

méthode1
méthode2
méthode3
traiter_pendant t où t est le temps nécessaire au processeur pour exécuter méthode 1,2,3

Ce temps de traitement t qui intervient dans la simulation que nous proposons de mettre en place dans ce projet. Cependant il n'est pas possible de fixer ce temps de traitement avant la phase de l'architecture opérationnelle. Comme cela a déjà été mentionné dans le rapport, de nombreux problèmes doivent être pris en compte. D' une part des problèmes liés au déroulement du programme, d' autre part des problèmes liées à son exécution sur la machine de traitement (calcul sur multiprocesseur ou coprocesseur, existence de pipeline, technique de compilation, optimisation ...), il reste alors encore à prendre en compte l' influence possible de la mémoire cache, de phénomène de pagging ou encore d' accès mémoire de type DMA. Un certain nombre de recherche on déjà été poursuivis dans le domaine, en particulier par des groupes qui comme Covadis s' intéresse à la mise en place de projet de conception d' architecture temps réel proposant des fonctions de validation .

L' étude du temps de traitement (sans préemption) consiste en la recherche du pire temps d'exécution (WCTE Worst Case Time Execution).
En général on ne peut pas calculer le pire temps d'exécution et l'on se limite donc a en chercher une borne supérieure. La première propriété que l'on cherche à vérifier et celle de sécurité, le temps calculé doit effectivement être une borne du pire cas. On cherche aussi à obtenir un temps le plus proche possible du pire cas, on parle alors de propriété de précision. Dans le cadre d'une validation, il est nécessaire de posséder la propriété de sécurité, la précision du calcul est cependant utile pour permettre de dimensionner au mieux et donc au moins cher l'architecture matérielle.
L'Annexe 1 présente des moyens d'évaluer ce temps

5 Conclusions et Perspectives

Le travail qui a été effectué dans le cadre de ce stage a permis de montrer les possibilités de modélisation d'architectures à l'aide du logiciel Opnet. Certains de ces modèles sont totalement opérationnels (ordonnanceurs de CPU) ,les autres étant en phase finale d'implémentation (modèles de réseau).
La mise en place de modèles de l'architecture fonctionnelle, des modèles d'architecture opérationnelle et des projections nécessaires entre ces modèles a été défini. Il reste maintenant à finaliser l'implémentation de ces modèles et à générer automatiquement (cf Principe de traduction Ardeco vers OPNET) les modèles de l'architecture opérationnelle d'un système décrit en Ardeco.
Il sera alors possible de vérifier a priori les caractéristiques attendus de notre système à partir de la définition de son architecture fonctionnele et du choix de son architecture matérielle de projection.

Comme attendu, les différentes réflexions liés à cette phase de modélisation de l'architecture opérationnelle ont montré la nécessité d'enrichir le langage Ardeco. D'une part la projection nécessite l'existence d'un langage de description de l'architecture matériel mais aussi des règles nécessaires à la projection (choix des priorités …).
Il est aussi apparu que la modélisation des temps de traitements à l'aide d'une unique instruction traiter_pendant se révélaient insuffisante en particulier pour prendre en compte le temps lié aux accès disques ou à d'autres types d'accès à des mémoires de masse.
Il se révélerait tout aussi pertinent de formaliser les propriétés attendues du système en particulier une notion d'échéance. Non seulement pour pouvoir les valider au cours de la simulation mais surtout pour pouvoir les utiliser pour appliquer des stratégies qui les respecteront.

6 Bibliographie

Rapports de contrats COVADIS

[Thomas 97a] Thomas L., Jourdan J., Simonot-Lion F., Bayart M., Choukair C., Peraldi M.-A., André C., Deplanche A.-M., Trinquet Y., rapport intermédiaire à 6 mois du contrat COVADIS 96448 DRET-DGA/ 96C0076 DGRT-MENESR, juillet 1997
[Thomas 97b] Thomas L., Mattioli J., Jeannet B., Jourdan J., Simonot-Lion F., Toussaint J., Kaiser L., Bayart M., Choukair C., Peraldi M.-A., André C., Deplanche A.-M., Trinquet Y., rapport intermédiaire à 12 mois du contrat COVADIS 96448 DRET-DGA/ 96C0076 DGRT-MENESR, décembre 1997
[Thomas 98] Thomas L., Simonot-Lion F., Toussaint J., Bayart M., Choukair C., Peraldi M.-A., André C., Deplanche A.-M., Trinquet Y., rapport intermédiaire à 18 mois du contrat COVADIS 96448 DRET-DGA/ 96C0076 DGRT-MENESR, juillet 1998.
[Thomas 99a] Thomas L., Simonot-Lion F., Bayart M., Choukair C., Peraldi M.-A., André C., Deplanche A.-M., Trinquet Y., rapport intermédiaire à 2 ans du contrat COVADIS 96448 DRET-DGA/ 96C0076 DGRT-MENESR, février 1999.

[Thomas 99b] Thomas L., Lambolais T., Lesiour R., *Architectural Techniques for the Description and Validation of Distributed Real-Time Systems*, in Proceedings ISORC'99, 3-5 mai 1999, Saint-Malo.

Propriétés temporelles

[Vega 95] Vega Saens L., Thomesse J.-P., Temporal Properties in Distributed Real-Time Applications, in Proceedings of 13th IFAC Workshop on Distributed Computer Control Systems, (réd. A.-K. Sahraoui, J.-A. de la Puente), p.119-124, Toulouse (France), septembre 1995.
[Vega 96] Vega Saens L., *Modèles de coopération et de communication entre processus temps réel répartis*, Doctorat de l'Institut National Polytechnique de Lorraine, septembre 1996

[Toussaint 96] Toussaint J., Véga-Saens L., Simonot-Lion F., *Formal Verification of Time Constrained Communications*, proceedings International Conference on Parallel and Distributed Computing Systems, ISCA'96, p. 138-143, Dijon, septembre 1996.

[Toussaint 97e] Toussaint J., Simonot-Lion F., Thomesse J.-P., *Time Constraint Verification Methods Based on Time Petri Nets*, in Proceedings 6th Workshop on Future Trends in Distributed Computing Systems, FTDCS'97, p. 262-267, Tunis (Tunisie), octobre 1997
[Delfieu 95] Delfieu D., Expression et validation de contraintes temporelles pour la spécification des systèmes réactifs, thèse de doctorat

[Kountouris 98] Kountouris A., Outils pour la validation temporelle et l'Optimisation de Programmes Synchrones, thèse de doctorat.

Placement

[Beauvais 99] Beauvais J.P. Placement de tâches temps-réel dans un système réparti

Simulation

[Courrier 98b] Courrier M., Simonot-Lion F., Song Y.-Q., *Microscopic Modeling of Support System for in-Vehicle Embedded Systems*, in Proceedings IFIP Workshop on Distributed and Parallel Embedded Systems, DIPES'98, Paderborn (Allemagne), octobre 1998.
[Simonot 97c] Simonot-Lion F., Song Y.Q., Raymond J*., Validating real-time applications distributed over CAN : an interoperability verification*, in Proceedings 4th international CAN conference, ICC'97, Berlin (Allemagne), october 1997
[Song 97b] Song Y.Q., Simonot-Lion F., Belissent P., *VACANS - A tool for the validation of CAN-based applications*, in Proceedings of IEEE WFCS' 97, Barcelone (Espagne)ctober 1997.

Architecture

[Simonot 96a] Simonot-Lion F., Bayart M., *Le processus de développement des systèmes automatisés de production : analyse de l'étape de conception*, , in Actes Conférence on Real-Time Systems and Embedded Systems, RTS'96, (réd. Teknéa), Paris, janvier 1996.
[Bayart 95a] Bayart M., Simonot-Lion F., Impact de l'émergence des réseaux de terrain et de l'instrumentation intelligente dans la conception des architectures des systèmes d'automatisation de processus, 132 pages, contrat MESR 92-P-239, février 1995.
[Trinquet 99] Trinquet Y., Les systèmes d'exploitation temps réel

Synchronisation d'horloge

[He 93] He J., *Modélisation de la synchronisation d'horloge dans les systèmes répartis,* thèse de doctorat de l'Institut National Polytechnique de Lorraine, juillet 1993.

Transcription de modèle

[Mrabet 96] Mrabet R., Modèle de simulation QNAP2 pour l'étude de performances de spécifications Estelle hiérarchiques,Colloque Francophone de l'ingénierie des Protocoles, CFIP'96.Annexes
[Valderruten 93] Valderruten Vidal A., Hjiej O.,Benzekri A., Gazal D., Deriving Queuing Networks Performance Models from Annotated LOTOS specifications, 6th International Conference on Modelling Techniques and Tools for Performance Evaluation, Edimburgh University Press, août 1993.

Modélisation

[Bellissent 96] Bellissent P., *Validation de systèmes distribués autour du réseau CAN,* mémoire de DEA d'informatique, septembre 1996, Nancy.
[Courrier 98c] Courrier M., *Modélisation de composants matériels et exécutifs en vue de la validation d'architectures opérationnelles par évaluation de performances*, Diplôme Firtech INPL, novembre 1998.
[Delorme 98] Delorme E., *Modélisation d'un langage de description et de validation d'architectures logicielles avec un langage synchrone : Esterel et un outil d'évaluation de performances : OPNET*, , mémoire de DEA d'informatique, septembre 1998, Nancy.

[Philippe 97b] Philippe C., *Méthodologie de modélisation et d'évaluation d'applications réparties temps réel sûres de fonctionnement* , thèse de doctorat de l'INPL, octobre 1997, Nancy.
[Toussaint 97d] Toussaint J., *Modélisation d'applications temps réel réparties pour la validation de propriétés temporelles – Méthodologie de construction de modèles et algorithmes de validation*, Doctorat de l'Institut National Polytechnique de Lorraine, octobre 1997
[Garzon 97] Garzon Frederic, Modélisation de TCP/IP à l' aide d' OPNET, rapport de DEA

Design

[Mooney 97] Mooney V., Sakomata T., Run-Time Scheduler Synthesis for Hardware-Software Systems and Application to Robot Design .
[Neely 98] Neely W. S.,Mooney V. J., System Level Design for System on a Chip
[Knudsen 98] Knudsen P. V., Madsen J., Communication Estimation for Hardware/Software Codesign
[Abdullah 98] Abdullah F., Mooney V. J., Verilog Simulation of aHardware-Software Implementation of a Robotics Control Algorithm .
[POLIS 99] Polis documentation
[MetaH 98] Metah documentation
[Chinook] Chinook documentation
[Cosyma] Cosyma documentation
]

7 ANNEXES

Calcul du temps d'exécution

Le problème du chemin d' exécution

La principale difficulté dans le calcul du temps d' exécution réside dans la recherche du chemin d' exécution. En effet le temps de traitement étant fonction du nombre d' instructions, il est intéressant de connaître le nombre d' instructions qui seront exécutés par un processus. Pour connaître la suite d' instructions maximales (en tenant en compte du temps d' exécution de chaque instructions) il est intéressant d' étudier les suites maximales d' instructions qui peuvent être exécutées (par l'étude des boucles et des sauts conditionnels en particulier). Quelques remarques essentielles sont à formuler :
1) Plus le programme est important, plus la recherche du chemin d' exécution va se révéler délicat (complexité exponentielle), il est donc nécessaire qu' une application soit divisée en un certain nombre de modules, et de pouvoir ramener l' étude de l' application à l' étude séparée des différents modules. La démarche modulaire est la seule qui permette d' obtenir rapidement des résultats exploitables. Dans le cadre du projet Covadis, la notion de module est développée dès le niveau fonctionnel, on obtiendra donc logiquement en final une application modulaire.
2) L' étude du programme doit se faire au niveau d' un langage évolué et structuré et pas en assembleur, pour permettre de garder le maximum d' expressivité dans le programme
3) Une grande difficulté est liée à l' influence des paramètres d' entrées de la fonction. On ne s' intéresse pas ici aux conséquences de l' entrée sur le résultat final, mais uniquement à ses conséquences sur le chemin d' exécution du programme (nombre de passages dans une boucle ...)

exemple
soit a[i] variable d' entrée

```
for (i=0,i<100,i++)
{temp[i]=a[i]*2 ;
}
```
Dans ce cas, le chemin d' exécution ne dépend pas de l' entrée

par contre dans le cas suivant

```
for (i=0,a[i]<2,i++)
{temp[i]=a[i]*2 ;
}
```
on remarque que le chemin d' exécution dépend fortement des valeurs de la variable d' entrée.

L' étude des chemins d' exécution a fourni un certain nombre de technique pour trouver les résultats attendus. On trouve plusieurs familles de programmes de recherche.
Les plus simples, après une analyse simplifié du programme, cherche l'exécution contenant le nombre d' instructions maximales (en pouvant toutefois donnait un poids relatif a chaque instruction distincte). L'étude des boucles, si le nombre d' itérations n' est pas fixe, nécessite d'être borné par le programmeur, de même les sauts conditionnels doivent être commentés (ce qui nécessite un langage spécifique utilisé comme commentaire qui décrit ce genre de propriété [Puschner]). Ces mécanismes permettent d' obtenir des résultats vérifiant la propriété de sécurité mais la précision obtenu est souvent assez mauvaise.

D' autres techniques considèrent (on verra que pourtant ce n' est pas toujours le cas) la recherche du chemin d' execution comme un probleme de resolution d' équation et d' inéquation linéaire entière (cf timing-tree []) avec maximation d' une fonction de coût (le nombre de cycle machine correspondant à l' exécution du programme).

Ex
```
if a>b then
{
methode1
```

}

est transformé tout d' abord dans un langage qui ne conserve que les propriétés utiles et qui est indépendant du langage de programmation d' origine :
if
  20 (temps de calcul de la condition)
  oh_true 8 (temps du saut si la condition est vraie ou fausse ...)
  oh_false 10
  then 100 (temps de calcul de methode1)

Il faut ensuite générer les équations correspondantes

Z=f0*20+f1*8+f2*10+f3*100 (fonction de coût)
f1<=1 (correspond à oh_true)
f2<=1 (correspond à oh_false
f3=f1 (le then n' est exécuté que si la condition est vrai
f0=f1+f2 (si le programme décrit ici est exécuté une seule des deux conditions 1 ou 2 est vérifié)
f0=1 (le bout de programme considéré ici est exécuté)
Il ne reste plus qu'à donner ce système d'équation a un outil de résolution qui donne la solution optimale soit ici
Z=128 f1=1 f2=0 f3=1
Même si cette méthode peut sembler intéressante, les équations générés ne seront plus linéaires à partir du moment ou le nombre d' itération des boucles n' est pas constante. Il est donc la encore , comme pour la méthode précédente de fixer les bornes de toutes les boucles.

Méthode basée sur un exécution logique

Dans le cas précédent (timing tree) un certain nombre de chemin impossible ne sont pas écartés dans la recherche des chemins les plus long. En effet chaque condition est traité de manière indépendante et le programme n' est pas prévu pour considérer que si la condition a>b est considère comme vrai dans un test , il n' est pas très intéressant de la considérer comme fausse dans un autre test. Pour corriger ces problèmes d' autres études [Ye95] proposent donc de garder en mémoire les différents résultats logiques considérés au cours du programme pour éviter de générer des chemins d' exécutions impossibles. Il s' agit là de méthode classique dans le domaine de la validation de programme (dans des domaines non temps réel), qui doivent permettre d' améliorer la finesse des résultats. La gestion de ces informations supplémentaires se révèlent souvent très lourde et nécessite en plus d' un moteur d'inférence logique une grande quantité de mémoire pour stocker l' ensemble des informations du déroulement du programme.

L' étude des principales techniques de recherche des chemins d' exécutions amènent à plusieurs conclusions. Tout d' abord malgré l' impression d' avoir à traiter un problème simple, on rencontre de nombreuse difficulté (il faut cependant garder à l' esprit que dans un cadre générale, il n' existe pas de solution à notre problème. En effet le fait de savoir si un programme se termine est indécidable et il est donc impossible de résoudre de tel problème sans un certain nombre d' hypothèse.
On comprend donc bien l' intérêt d' une part, de garder un langage le plus expressif possible et de pouvoir d' autre part fixer un certain nombre d' hypothèse dans le langage et en particulier de borner toutes les itérations qui interviennent dans les calculs

Mes conclusions sont donc qu' il serait préférable de disposer d' un langage de programmation le plus expressive possible et qui permettent de donner, le plus de sens possible aux variables générées mais aussi de limiter les boucles au nombre d'itérations variable ou d' obliger le programmeur à fixer les bornes (ou de donner un moyen de les calculer en fonction d' autres variables plus explicites).
D' autre part la programmation modulaire permet de diminuer la complexité du programme (réduction de l'arbre des exécutions possibles) et doit donc être privilégié pour la programmation temps réel. pour ce qui est de prendre en compte les différentes relations logiques, qui entraîne des chemins d' exécutions impossibles. Il pourrait être plus pertinent de donner au programmeur des outils (recherche de relation, aide à la décision) au niveau de son éditeur de programme qui à la création de

chaque nouveau test conditionnelle lui permettra d' indiquer l' existence ou non d' une relation d' impossibilité manifeste entre le résultats de tests différents (mais liés). En effet le programmeur pourra plus facilement juger au moment de la création de son programme de la réel pertinence de prendre en compte telel ou telle incompatibilité.

L' exécution sur la machine de traitement

Nous nous sommes pour l' instant intéressés au chemin d' exécution du programme qui permet de connaître à priori le chemin d' exécution le plus long. La fonction d' évaluation utilisé prend en compte le temps de traitement sur le processeur (nombre de cycle) mais ce temps donné par le constructeur ne suffit plus dans les architectures actuelles (pipelines, RISC...) pour obtenir le temps d' exécution du programme. Pour obtenir des informations précises sur le temps d' exécution d' un programme il est en fait nécessaire de joindre à la recherche du chemin d' exécution un simulateur qui permettra de faire apparaître les gains de temps (pour la plupart complètement déterministe) du à l' utilisation d'une machine à l' architecture moderne. Il existe la encore de nombreuses études qui on étés réalisés et surtout de nombreux simulateurs qui émulent de manière très fine les différentes machine du marché. L' utilisation de ces simulateurs nécessitent l' existence du code objet des applications à tester. Or dans l' étude du chemin d' exécution ont a pu voir l' intérêt de travailler sur un langage de haut niveau, il est donc maintenant nécessaire de le compiler pour simuler l' exécution réel du programme. La dernière difficulté qu' il reste à résoudre pour arriver à ce stade est celle de l' optimisation au cours de la compilation. Des études ont montrés que l' optimisation à la compilation d' un programme peut améliorer sa vitesse d' exécution d' un facteur 3, il est donc très important de pouvoir prendre en compte ce phénomène, tout en évitant bien sûr de perdre le fil du chemin d' exécution trouvé à l' aide du langage de haut niveau. Les études dans le domaine ne sont pas très nombreuses et se révèlent complexe. Pourtant cette phase est essentiel. La méthode actuelle consiste à générer le code objet du programme C est à utilisé ce dernier dans le simulateur. On utilise alors des techniques de synchronisation (callback) qui permettent de créer le code objet nécessaire au simulateur tout en gardant le parallélisme entre le langage de haut niveau (nécessaire dans la recherche de graphe) et l'exécution instruction par instruction sur le simulateur. La synchronisation est nécessaire, en effet pour avoir les résultats les plus fin possible, il est nécessaire de posséder une fonction d' évaluation du coût processeur des instructions qui soit la plus proche possible de l' exécution réel sur la machines. Or dans les architectures modernes, le temps de calcul d' une instruction dépend des instructions qui la précèdent, c' est pourquoi cette technique de simulation est nécessaire. Il serait bien sûr possible de mettre en place un calcul de fonction d' évaluation qui prennent en compte tout les phénomènes possibles sans faire de simulation. Mais cela compliquerait considérablement la création de l'outil de validation car cela ne permettrait pas d' utiliser les simulateurs du marché qui sont parfaitement opérationnelle
L' utilisation des méthodes de recherche de chemins et de simulation de code objet sont donc utilisés conjointement pour obtenir les résultats les plus fins possibles sur les temps de calcul. On trouvera dans les articles suivants [] une description de quelques simulateurs et des différents éléments qui sont pris en compte. On remarquera en particulier que les comparaison entre les temps d' exécution estimés par ces simulateurs et les temps réellement observés sont très proches. Ces techniques permettent donc de prendre en compte les apports des architectures modernes des processeurs.

Les différentes études qui ont étés mentionnés pour prendre en compte l' architecture, ce limite cependant à l' architecture interne du processeur. A partir du moment ou l' on utilise des méthodes d' accès à l' information qui ne sont pas propre au processeur, de nouvelles difficultés de prise en compte apparaissent . En particulier les services de lecture/écriture sur bus, l' utilisation et la gestion de port de communication ou l' existence de DMA complique considérablement l' évaluation des performances et introduisent souvent un indéterminisme dans les temps de traitement (certains de ces indéterminisme reposent sur des paramètres dans le comportement ne peut être considéré que de façon probabiliste à l' échelle de notre étude, on peut signaler en particulier le positionnement de la tête de lecture sur un disque ou la présence dans une mémoire cache des prochaines données à lire).

Prise en compte de phénomène complexe (par exemple les vols de cycle liées à l'utilisation d'accès DMA)

La thèse de Tai-Yi Huang [Huang97] présente une étude relativement complète sur l'étude du pire temps d'exécution en prenant en compte un déroulement en parallèle d'entrée sortie de type DMA. En partant des recherches actuelles sur l'étude des chemins d'exécutions (en considérant par hypothèse que les différentes itérations sont bornées), il intègre les conséquences du vol du cycle, ce qui nécessite une étude bonne compréhension de son fonctionnement. En effet il faut décomposer les différents cycles du CPU en B-cycle ou E-cycle en fonction de l'utilisation ou nom du bus ou cours de l'utilisation d'une instruction. Les règles d'utilisation du bus par la mémoires sont complexes mais totalement déterministes (dans le cas du vol de cycle) et il est donc possible de les prendre finement en compte. Sur l'architecture étudié (680XX), Huang obtient, en tenant compte du phénomène de vol de cycle une augmentation du pire temps d'exécution de l'ordre de 14%. Plus généralement la méthode qu'il propose et qu'il applique au DMA lui permet aussi de prendre en compte les différentes mémoires caches du système et il applique en particulier à la mémoire cache du jeu d'instructions. En effet dans le processeur, avant de pouvoir traiter une instructions, il est nécessaire de la décoder, une mémoire cache de taille variable (il fait l'étude pour 4,8 et 16 instructions) permet donc d'éviter cette phase de décodage et de gagner du temps. L'application de méthodes équivalentes aux autres phénomènes qui peuvent influer sur les temps d'exécution comme les caches sur les données (data cache), les buffers spécialisés pour les E/S (prefetch I/O buffers) ou encore la prise en compte d'architecture plus complexe (pipeline à ordonnancement dynamique, processeur vectoriel, processeur superscalaire) doit permettre d'obtenir des informations complémentaires sur le temps d'exécution des tâches. Dans le cas des architectures multiprocesseur, il est nécessaire de disposer de moyen de synchronisation entre les processeurs et de prendre en compte les nombreuses impossibilité de paralléliser le traitement. D'autres phénomènes comme l'existence de bus multiples sur ce type d'architecture doit alors être pris en compte et la nécessité de synchroniser les différentes mémoires caches posent de nombreux problèmes. Des études comme [ANDERSON95] donne cependant un certains nombres de renseignement pour permettre l'évaluation des performances de tels systèmes.

En conclusion, on peut dire que l'étude du pire temps d'exécution d'un programme (WCET) est un problème complexe qui s'appuie sur des recherches de type génie logciel (recherche du chemin d'exécution, preuve de programme, compilation…) et nécessite une modélisation très fine du hardware (mémoire cache, pipeline …). La prise en compte des particularités du hardware permet d'améliorer la précision du calcul du pire temps d'exécution de l'ordre de 40%, les différents gains ne sont pas par contre calculable de façon générique et dépende fortement du code de l'application.
Ainsi, même si le pire temps d'exécution est calculable, il dépend très fortement des techniques de compilation/optimisation et de l'architecture finale de traitement.
Dans le cadre du projet Covadis, le but est de générer le temps de traitement maximale d'un traitement donnée en vue de le simuler. En fait ce temps dépendant fortement de l'architecture de projection. Il serait préférable de disposer d'un outils puissant qui génère le code de l'activité en fonction de sa description fonctionnelle et qui puisse alors calculer les temps de traitement de manière précise en fonction du choix d'une architecture. Donner un temps de traitement sans pouvoir passer par la phase de génération du code (au moins dans un langage de type C) se révèle délicat, il doit cependant être possible de générer des résultats avec des erreurs sur l'évaluation du pire cas pouvant dépasser un facteur 4 ou 5. Cependant on peut tout même garantir d'obtenir une borne du pire cas (propriété de sécurité) et la validation à l'aide d'un simulateur garde tout son sens.

Etat de l'art

ADL et validation d'architectures

Il existe plusieurs travaux de description de système par ADL (Langage de description d'architecture), qui offrent des possibilités d'évaluer les propriétés temporelles du système décrit. Il s'agit entre autres des projets Polis, COSYMA, Chinook et Metah.

Les différents projets

Polis *[Polis]*

Le but de ce projet est de fournir un outil de conception de système embarqué et d'aide au co-design (logiciel+matériel). Polis ne propose pas d'outils de placement ou d'aide à l'ordonnancement mais permet aux concepteurs de vérifier leurs choix d'implémentation en particulier à l'aide de méthodes formelles. Le principal intérêt du projet réside dans la génération automatique du code et dans les possibilités de valider a priori le fonctionnement du logiciel .

COSYMA *[COSYMA]*

(COSYnthesis for eMbedded micro Architectures)

COSYMA n'offre pas une méthodologie complète de design mais offre des outils d'exploration des arbres d'exécution qui permettent d'évaluer de manière très fine les temps d'exécutions des programmes en fonction des choix d'architectures (logiciel+matériel).

Chinook *[Chinook]*

Chinook propose des outils de co-design mais offre des services supplémentaires de prise en compte des mécanismes de communication. Les systèmes étudiés sont des systèmes réactifs ayant à respecter des contraintes de temps.

MetaH *[MetaH]*

Ce projet propose un langage de description d'architecture fonctionnelle mais aussi matérielle qui permet la validation de l'architecture opérationnelle. MetaH propose de nombreux outils d'analyse (ordonnancement, placement …) mais ne permet cependant pas de prendre en compte le choix de tel ou tel médium de communication.

Comparaison des projets étudiés avec COVADIS

Le projet Covadis offre les services de spécifications d'architecture et de validation et à plus long terme la prise en charge la génération du code. La prise en compte des temps de traitement se fait directement dans la description de l'architecture fonctionnelle en considérant qu'il est possible de les évaluer ou tout du moins de les borner. Les modèles utilisés dans la modélisation de l'architecture opérationnelle permettent de considérer un temps physique sur l'ensemble de l'architecture et donc de déduire de la simulation des modèles les propriétés temporelles du système dans sa globalité en prenant en compte l'impact du choix du médium de communication et de sa gestion.

Les projets les plus avancés sur la validation d'AOP sont MetaH et Polis. Ils offrent des mécanismes d'analyse de l'architecture beaucoup plus modulaire que ceux de Covadis. Mais ils ne qui permettent que de vérifier des propriétés très particulières et non de valider le fonctionnement global de l'architecture (ce que fait Covadis). MetaH propose contrairement à Polis des outils de projection de l'architecture fonctionnelle sur l'architecture matérielle qui semblent intéressant. En particulier MetaH propose un langage de description d'architecture matérielle qui permet de modéliser finement les processeurs mais qui n'offre pas encore de grande possibilité pour les réseaux de communication.

L'étude des différents projets de design nous montre les difficultés d'obtenir un logiciel qui puisse gérer entièrement la phase de conception, de validation et de génération d 'applications embarqués temps réels. La prise en compte du temps au niveau de l'ensemble de l'applications posent de nombreux problèmes et nécessite une modélisation très fine du système. Le projet Covadis, n'offre pas les outils de génération du code et donc d'estimation des temps de traitements (du moins pour

l'instant) mais offre des mécanismes de validation du fonctionnement globale de l'application en prenant en compte les mécanismes de communication ce qui n'est pas le cas des autres projets.

Génération automatique de modèles

La base de notre projet réside dans la génération automatique de modèles d'architecture opérationnelle à partir d'une description d'architecture fonctionnelle. Les projets de design fournissent des outils qui nécessitent l'existence de ces modèles (validation par évaluation de performance) mais n'expliquent pas les modèles utilisés ou leurs modes de génération. Des pistes sur les méthodes de générations de modèles ont donc été trouvés dans un domaine qui n'est pas celui du design.

Ainsi [Mrabet96] propose une méthode de création de modèle QNAP 2 (qui permet des résolutions analytiques de modèles mais surtout des évaluations de performances par simulation) à partir de spécifications ESTELLE. Cette étude montre notamment la nécessité d'enrichir la description en ESTELLE de commentaires formalisés (Flag) qui permettent au moment de la génération des modèles QNAP de choisir entre plusieurs possibilités de modélisation (forme de projection).

De même dans [Valderruten93] une méthode de génération de modèles à base de files d'attentes à partir de spécifications en LOTOS est proposé. En particulier il est montré la nécessité de définir des informations qui n'ont pas de sens dans la spécifications en LOTOS mais qui sont nécessaires pour la mise ne place des modèles et de la simulation (pour l'analyse des performances).

La génération automatique de modèles peut souvent se révéler complexe quand il s'agit de changer de domaine. Ainsi dans notre cas comme dans les cas cités précédemment le but est de déduire des spécifications (LOTOS,ESTELLE ou ARDECO dans notre cas) des modèles opérationnels afin de pouvoir obtenir des informations par analyse de performances. La transcription est donc compliqué par la nécessité d'inclure des notions supplémentaires, il s'agit en fait plus de problème de projection que de problèmes de transcription comme on pourrait en rencontrer en passant d'un langage de programmation à un autre.

Ordonnancement de tâches

Les tâches :

Les tâches peuvent être indépendantes ou coopérer par des mécanismes de synchronisation (postage / attente d' un événement) ou de communication de données (par boîte aux lettres ou par rendez-vous). Dans un cas comme dans l' autre, ces relations se traduisent, d' un point de vue comportemental, par des contraintes de précédence définissant ainsi un ordre dans l' exécution des tâches impliquées. De plus, plusieurs tâches peuvent partager une ou plusieurs ressources dont l' accès est exclusif. Ce qui veut dire que si plusieurs tâches demandent la même ressource critique, l' exécutif temps réel autorisera l' exécution d' une seule tâche afin d' assurer l' exclusion mutuelle. Par conséquent, les autres tâches seront bloquées en attente de cette ressource.

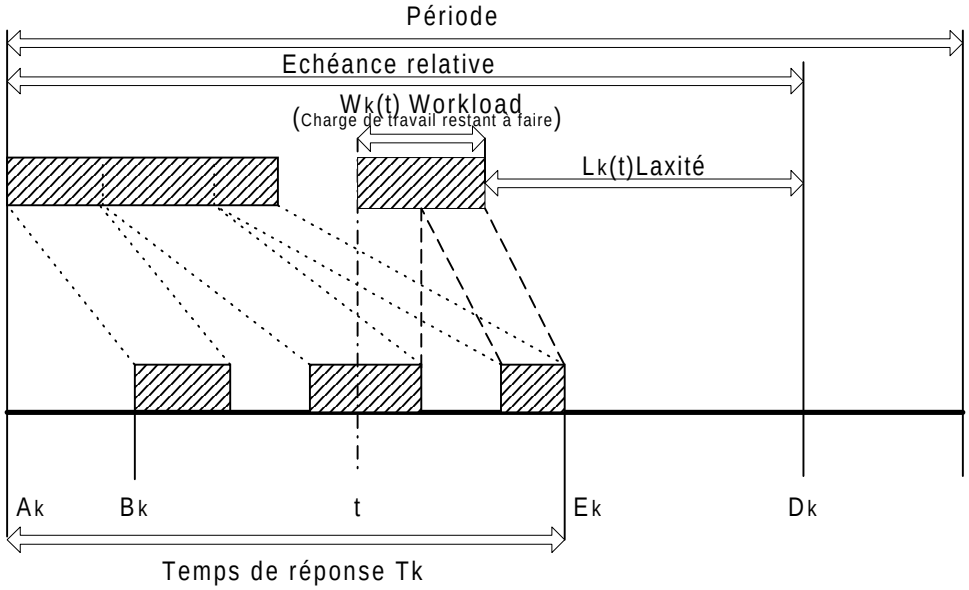


Figure A1 Définition des caractéristiques d'une tâche

Dans un système temps réel, les tâches doivent respecter des délais critiques imposés par les dynamiques de l' environnement à contrôler. à la durée d' exécution de la tâche seule sur le processeur.

- D_k correspond à l' échéance, qui permet de fixer l' intervalle de temps durant lequel la tâche doit finir son exécution. En dehors de cet intervalle son exécution devient inutile.
- P_k correspond à la période, c' est l' intervalle de temps fixe qui sépare deux arrivées successives d' une tâche. Ce paramètre concerne les tâches activées par un signal périodique provenant d' une horloge temps réel interne. Ce type de tâches est appelé tâches périodiques.
- AK :correspond à la date d'activation de la tâche
- Bk à sa date de prise en compte
- Ek à la date de fin de traitement
- Tk au temps de réponse

- Wk(t) au travail restant à effectuer sur la charge à l'instant t
- Lk(t) correspond à la laxité de la tâche c'est à dire la différence entre son échéance le temps de travail restant à exécuter

Il existe d' autres types de tâches : les tâches apériodiques et les tâches sporadiques.

Les tâches apériodiques correspondent à un service dont l' exécution est déclenchée par un événement aléatoire qui provient soit du procédé, soit par une action interne du système mais sans aucune synchronisation avec ce dernier.

Les tâches sporadiques sont comme des tâches apériodiques mais on constate en plus l' existence d' une durée minimale entre deux événements apériodiques (issus de la même source). Pour considérer le pire cas, il suffit alors de les traiter comme des tâches périodiques de période égale à cette durée minimale entre deux activations.

De plus une tâche peut être préemptible ou non. On dit qu' une tâche est non préemptible si lorsqu' elle a commencé son exécution, elle doit être achevée avant qu' une autre tâche obtienne le processeur. Inversement une tâche est préemptible lorsqu' elle peut être interrompue au profit d' une autre et être reprise ultérieurement.

Les différents algorithmes d' ordonnancement

Plusieurs tâches peuvent concourir pour l' obtention du processeur. L' attribution du processeur aux tâches i.e. leur exécution doit être planifiée pour garantir le respect de leurs contraintes temporelles. Cette fonction est gérée par un élément de l' exécutif temps réel appelé ordonnanceur. L' ordonnanceur doit régler les conflits des tâches concurrentes pour l' accès au processeur en utilisant une politique d' allocation qui favorise l' accès aux tâches les plus urgentes. Cette politique est dictée par un algorithme d' ordonnancement qui construit une séquence de l' ensemble des tâches à exécuter en veillant à respecter leurs contraintes temporelles et de synchronisation. Les études menées sur le sujet reposent sur la modélisation des tâches et la définition d' algorithmes d' ordonnancement. [Cardeira 94]

Les algorithmes d' ordonnancement peuvent être qualifiés d' ordonnancement hors ligne ou d' ordonnancement en ligne.

- Le premier type d' ordonnancement nécessite la connaissance de toutes les tâches et de leurs caractéristiques temporelles avant le démarrage de l' application. La construction d' une séquence d' exécution des tâches respectant les contraintes temporelles est possible et l' ordonnanceur se réduit alors à une entité appelée répartiteur qui ne fait que lancer les tâches dans l' ordre dicté par la séquence. C' est la solution actuellement employée dans l' automobile. Evidemment une telle approche ne permet pas de prendre en compte les tâches dont la date d' activation n' est pas connue au départ.
- Le second type d' ordonnancement est l' ordonnancement en ligne qui construit au fur et à mesure du déroulement de l' application, la séquence d' exécution à partir des tâches prêtes à l' instant courant. Ceci permet entre autre la prise en compte des tâches apériodiques. De tels ordonnancements attribuent le processeur à une tâche en fonction d' un algorithme d' ordonnancement. Ce dernier est conduit par priorité s' il exécute d' abord la tâche la plus prioritaire. La priorité est soit affectée par le concepteur de l' application en fonction de la criticité des tâches, soit assignée de manière automatique en fonction d' un paramètre temporel caractérisant les tâches tel que la période ou l' échéance qui reflètent l' urgence et non l' importance de la tâche au sens applicatif.

Les algorithmes d' ordonnancement à priorité fixe :

Un tel algorithme affecte à chaque tâche une priorité qu' elle conserve durant toute la vie de l' application.

L'algorithme Round-Robin

Dans le cas où des tâches de même priorité doivent prendre le contôle du CPU, il est possible de mettre en place une stratégie de type tourniquet (time slicing) qui porte le nom de Round-Robin.

L' algorithme Rate Monotonic

C' est un des algorithmes de base dans la littérature sur l' ordonnancement dans les systèmes temps réel. L' algorithme s' applique à des tâches périodiques sur un seul processeur. La plus grande priorité est associée à la tâche de plus petite période.

L' algorithme Deadline Monotonic :

La plus grande priorité est associée à la tâche possédant la plus petite échéance. Pour les tâches de même échéance, l' affectation de priorité se fait de manière aléatoire. En échéance sur requête l' algorithme est équivalent à l' algorithme Rate Monotonic.

Les algorithmes d' ordonnancement à priorité dynamique :

Un tel algorithme recalcule la priorité de chaque tâche durant leur exécution. Ainsi au cours de la vie de l' application la priorité de la tâche évolue.

L'algorithme Earliest Deadline First (EDF)

Cet algorithme ordonne les tâches selon les échéances croissantes, i.e. la tâche en attente ayant l' échéance la plus proche obtient le processeur. Cet algorithme est optimal pour les systèmes sans partage de ressources.

L' algorithme Least Laxity First :

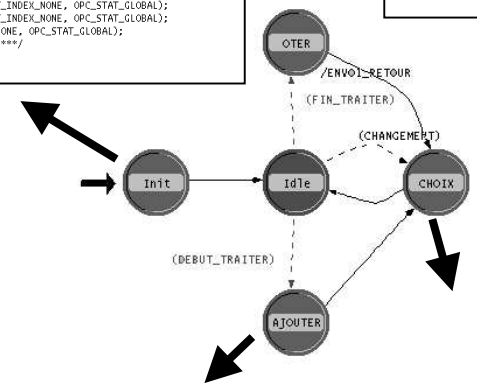
.

Cet algorithme ordonne les tâches selon les laxités croissantes, i.e. la tâche en attente ayant la laxité la plus proche obtient le processeur.

Le modèle de l'ordonnanceur

```
CPU=FREE;
PREEMPTION=NO;
QUANTUM=INIT_QUANTUM=OK;
EDF=NO;
DUREE_QUANTUM=0.1;
traitement=OK;
temp_trait=0.001;
if (EDF==OK) { PREEMPTION=OK; QUANTUM=INIT_QUANTUM=NO;}

/*************************************/
count=counttp1=counttp2=counttp3=0;
/*************************************/

/*********************************************************/
sauvegarde = op_stat_reg ("temps d'attente par tache",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
s2 = op_stat_reg ("Verif",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
s3 = op_stat_reg ("CPU",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
s4 = op_stat_reg ("TACHES_FINIES",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
s5 = op_stat_reg ("TACHES_PRETES",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
ta1= op_stat_reg ("Temps d'attente tache type 1",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
ta2= op_stat_reg ("Temps d'attente tache type 2",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
ta3= op_stat_reg ("Temps d'attente tache type 3",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
tp1= op_stat_reg ("TACHES_PRETES+EXEC type 1",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
tp2= op_stat_reg ("TACHES_PRETES+EXEC type 2",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
tp3= op_stat_reg ("TACHES_PRETES+EXEC type 3",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
cpuplus = op_stat_reg ("TACHE ACTIVE",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
/*********************************************************/
```

```
    if ( CPU==BUSY )
    {
    op_stat_write( sauvegarde, (double) op_sim_time() - une_tache.ARRIVE -une_tache.DUREE);

    if (une_tache.SOURCE != 0.0) count ++;

    CPU=FREE;
    if (une_tache.SOURCE==1.0) {counttp1=-;op_stat_write( ta1, (double) op_sim_time() - une_tache.ARRIVE -une_tache.DUREE);}
    if (une_tache.SOURCE==2.0) {counttp2=-;op_stat_write( ta2, (double) op_sim_time() - une_tache.ARRIVE -une_tache.DUREE);}
    if (une_tache.SOURCE==3.0) {counttp3=-;op_stat_write( ta3, (double) op_sim_time() - une_tache.ARRIVE -une_tache.DUREE);}
    op_stat_write( tp1, (int) counttp1);
    op_stat_write( tp2, (int) counttp2);
    op_stat_write( tp3, (int) counttp3);
    op_stat_write( s3, (int) CPU);
    op_stat_write( s4, (int) count);
    op_stat_write( s5, (int) op_subq_stat (0,OPC_QSTAT_PKSIZE));
    op_stat_write( cpuplus, (double) (une_tache.SOURCE+0.1));
    if (op_subq_empty (0) == OPC_FALSE)  { op_intrpt_schedule_self ( op_sim_time (), 111);}
    une_tache.VERIF=(une_tache.VERIF+(op_sim_time() - une_tache.DATE_SELF));
    printf("stat OK");
    op_stat_write( s2, (double) une_tache.VERIF);
    }
```




```
    double pendant;
    TACHE nouvelle_tache;

    pkptr2 = op_pk_get (op_intrpt_strm ());

    /*******************************************/
     op_pk_fd_get (pkptr2,1,&(source));
     op_pk_fd_get(pkptr2,3,&(nouvelle_tache.RESTANT));
     nouvelle_tache.ARRIVE=op_sim_time();
    /*******************************************/

    /*****************************************************/
    if (EDF==OK)
 op_p   {op_pk_fd_get(pkptr2,4,&(nouvelle_tache.ECHEANCE));
 op_p    nouvelle_tache.DATE_ECHEANCE=nouvelle_tache.ARRIVE+nouvelle_tache.ECHEANCE;
 op_p    nouvelle_tache.PRIORITE=(1/nouvelle_tache.DATE_ECHEANCE);
 op_p op_pk_priority_set(pkptr2,nouvelle_tache.PRIORITE);
 op_p }
/****  /*****************************************************/
/****
if (s /*****************************************************/
if (s printf("sa%g,prio:%g",source,nouvelle_tache.PRIORITE);
if (s /*****************************************************/
op_st
op_st /*****************************************************/
op_st
/**** /*******************************************/
      op_pk_fd_set(pkptr2,5,OPC_FIELD_TYPE_DOUBLE,nouvelle_tache.PRIORITE,32);


if (op_subq_pk_insert (0, pkptr2, OPC_QPOS_PRIO) !=
           OPC_QINS_OK) /* PRIO ou TAIL */
    {
    op_pk_destroy (pkptr2);
     /* ou signaler que l'on ne peut pas prendre sa requete en compte au demandeur
     }
```

```
if ( (QUANTUM==OK) &&
     (  (op_intrpt_code () == 112) || (INIT_QUANTUM==OK)
     )
   )
    {
      INIT_QUANTUM=NO;
     op_intrpt_schedule_self ( op_sim_time () + DUREE_QUANTUM, 112);
       traitement=OK;
       printf("qu(%g)",op_sim_time());     }
if (CPU == FREE && (op_subq_empty (0) == OPC_FALSE)
   )
   { if (op_subq_empty (0) == OPC_FALSE)
    {pkptr=op_subq_pk_remove (0,OPC_QPOS_HEAD);

   /*****************************************************/

     op_pk_fd_get(pkptr,1,&(une_tache.SOURCE));
     op_pk_fd_get(pkptr,2,&(une_tache.IDENTIFICATEUR));
     op_pk_fd_get(pkptr,3,&(une_tache.DUREE));
     op_pk_fd_get(pkptr,4,&(une_tache.ECHEANCE));
     op_pk_fd_get(pkptr,5,&(une_tache.PRIORITE));
     op_pk_fd_get(pkptr,6,&(une_tache.PREEMPTIF));
     op_pk_fd_get(pkptr,7,&(une_tache.RESTANT));
     op_pk_fd_get(pkptr,8,&(une_tache.ARRIVE));
     op_pk_fd_get(pkptr,9,&(une_tache.DATE_SELF));
     op_pk_fd_get(pkptr,10,&(une_tache.VERIF));
     op_pk_fd_get(pkptr,11,&(une_tache.DATE_ECHEANCE));
```

```
   op_stat_write( cpuplus, ((double) (une_tache.SOURCE)-0.1));
   /*****************************************************/

   /*****************************************************/
   une_tache.REVEIL=op_intrpt_schedule_self ( op_sim_time () + une_tache.RESTANT,110);
   une_tache.DATE_SELF=op_sim_time();
   /*****************************************************/
   CPU=BUSY;
   }

   /*****************************************************/
   op_stat_write( s3, (int) CPU);
   op_stat_write( s4, (int) count);
   op_stat_write( s5, (int) op_subq_stat (0,OPC_QSTAT_PKSIZE));

   /*****************************************************/
 {


  }
}

else
   { if (((op_subq_empty (0) == OPC_FALSE) && ((PREEMPTION==OK || (QUANTUM==OK && (op_intrpt_type () != OPC_INTRPT_STRM))) ))
      {
        une_tache.RESTANT=(une_tache.RESTANT- ( op_sim_time() - une_tache.DATE_SELF));
        une_tache.VERIF=(une_tache.VERIF+(op_sim_time() - une_tache.DATE_SELF));
        op_ev_cancel(une_tache.REVEIL);

 /*************************************************************/
             op_pk_fd_set(trait,10,OPC_FIELD_TYPE_DOUBLE,0.0,32);
             op_pk_fd_set(trait,7,OPC_FIELD_TYPE_DOUBLE,temp_trait,32);
          if (op_subq_pk_insert (0, trait, OPC_QPOS_HEAD) != OPC_QINS_OK)
                {
                  op_pk_destroy (trait);     /* PRIO ou TAIL*/
                   /* ou signaler que l'on ne peut pas prendre sa requete en compte au demandeur...*/
                }
             }
  /*************************************************************/
          CPU=FREE;
          op_intrpt_schedule_self ( op_sim_time (), 111);
  /*************************************************************/
         }
      }

          tache = op_pk_create (0);
          op_pk_fd_set(tache,1,OPC_FIELD_TYPE_DOUBLE,une_tache.SOURCE,32);
          op_pk_fd_set(tache,2,OPC_FIELD_TYPE_INTEGER,une_tache.IDENTIFICATEUR,16);
          op_pk_fd_set(tache,3,OPC_FIELD_TYPE_DOUBLE,une_tache.DUREE,32);
          op_pk_fd_set(tache,4,OPC_FIELD_TYPE_DOUBLE,une_tache.ECHEANCE,32);
          op_pk_fd_set(tache,5,OPC_FIELD_TYPE_DOUBLE,une_tache.PRIORITE,32);
          op_pk_fd_set(tache,6,OPC_FIELD_TYPE_INTEGER,une_tache.PREEMPTIF,16);
          op_pk_fd_set(tache,7,OPC_FIELD_TYPE_DOUBLE,une_tache.RESTANT,32);
          op_pk_fd_set(tache,8,OPC_FIELD_TYPE_DOUBLE,une_tache.ARRIVE,32);
          op_pk_fd_set(tache,9,OPC_FIELD_TYPE_DOUBLE,une_tache.DATE_SELF,32);
          op_pk_fd_set(tache,10,OPC_FIELD_TYPE_DOUBLE,une_tache.VERIF,32);
          op_pk_fd_set(tache,11,OPC_FIELD_TYPE_DOUBLE,une_tache.DATE_ECHEANCE,32);
          op_pk_priority_set(tache,une_tache.PRIORITE);

          printf("s%g,prio:%g",une_tache.SOURCE,une_tache.PRIORITE);
  /*********************************************************************/
        op_stat_write( cpuplus, (double) (une_tache.SOURCE));

        if (op_subq_pk_insert (0, tache, OPC_QPOS_PRIO) != OPC_QINS_OK)
                {                       /* PRIO ou TAIL*/
                  op_pk_destroy (tache);
                   /* ou signaler que l'on ne peut pas prendre sa requete en compte au demandeur...*/
                }
       if ((une_tache.SOURCE != 0) && (traitement==OK))
        { traitement=NO;
        trait = op_pk_create (0);
           op_pk_fd_set(trait,1,OPC_FIELD_TYPE_DOUBLE,0.0,32);
           op_pk_fd_set(trait,2,OPC_FIELD_TYPE_INTEGER,0,16);
           op_pk_fd_set(trait,2,OPC_FIELD_TYPE_DOUBLE,1.0,32);
           op_pk_fd_set(trait,10,OPC_FIELD_TYPE_DOUBLE,0.0,32);
           op_pk_fd_set(trait,10,OPC_FIELD_TYPE_DOUBLE,0.0,32);
           op_pk_fd_set(trait,7,OPC_FIELD_TYPE_DOUBLE,temp_trait,32);
        if (op_subq_pk_insert (0, trait, OPC_QPOS_HEAD) != OPC_QINS_OK)
                {                       /* PRIO ou TAIL*/
                  op_pk_destroy (trait);
                   /* ou signaler que l'on ne peut pas prendre sa requete en compte au demandeur...*/
                }
             }
  /*************************************************************/
        CPU=FREE;
        op_intrpt_schedule_self ( op_sim_time (), 111);
  /*************************************************************/
       }
    }
```