3

Le modèle de performance de MCSE

Pour le demandeur, un système est caractérisé par des exigences fonctionnelles et des exigences non-fonctionnelles. Les exigences fonctionnelles expriment le comportement voulu du système en relation avec son environnement. Les exigences non-fonctionnelles décrivent un ensemble de contraintes imposées qui concernent à la fois le produit et le procédé de développement et de réalisation du produit. Pour le produit, ces exigences sont d'ordre technique et d'ordre économique.

On s'intéresse ici tout particulièrement aux exigences techniques. Parmi celles-ci, les exigences de performance quantifient le comportement du système vis-à-vis de critères d'observation qui peuvent être externes au système (temps de réponse, débit, etc) ou internes (taux d'utilisation de ressources, capacité d'un bus), etc [CALVEZ-98a].

Comme tout objectif premier d'un développement est de satisfaire les exigences du demandeur, il est essentiel de pouvoir vérifier tout au long de la conception que la solution retenue permet de satisfaire les performances attendues. Ainsi, l'évaluation des performances concerne toutes les phases d'une méthodologie de conception. Durant l'étape de spécification, elle est utile pour vérifier et valider les spécifications avec le client puis faire une rapide étude de faisabilité du projet. L'analyse des performances qui s'effectue lors de l'étape de conception fonctionnelle ou conception préliminaire aide au dimensionnement des éléments internes du système (taille d'un port de communication par exemple). Lors de l'étape de conception architecturale, l'estimation des performances dynamiques en résultat d'un partitionnement matériel/logiciel permet par une démarche itérative de trouver la solution optimale vis-à-vis des contraintes imposées. Enfin après synthèse de la solution, la rétro-annotation des résultats de synthèse dans le modèle de performance utilisé permet d'évaluer le système sans utiliser

obligatoirement des techniques de prototypage ou de réalisation avec observation des propriétés temps-réel de la solution.

L'évaluation de performances nécessite qu'un système (et ceci est particulièrement vrai durant l'étape de conception car il n'existe pas) soit décrit par un modèle qui soumis à une configuration de charge donnée permet d'extraire des informations quantitatives.

Le modèle MCSE, déjà bien adapté à la conception des systèmes temps réel [CALVEZ-90] et des Asics [CALVEZ-93a], a donc été enrichi par un modèle de performance**.** Bien que la plupart des concepts de ce modèle de performance ont un certain historique car les premiers travaux datent de 1992 [CALVEZ-93b], la sélection des éléments de ce modèle a aussi été influencée par d'autres modèles de performances tels que les "Behavior Diagrams" de RDD100, le modèle de transactions de SES/Workbench, le modèle UVa, etc. Contrairement à la plupart des modèles de performances existants, notre modèle distingue clairement la vue fonctionnelle d'un système de sa vue architecturale. Cette distinction facilite l'exploration du domaine des solutions possibles d'un partitionnement matériel/logiciel. Notre modèle est en effet basé sur l'emploi du modèle fonctionnel et du modèle exécutif préconisés par la méthodologie MCSE et d'un modèle de comportement des fonctions. La méthodologie MCSE possédait déjà un modèle de comportement reposant sur la composition d'opérations élémentaires statiques. Jugé beaucoup trop restrictif pour l'évaluation des performances dynamiques d'un système, le modèle de comportement des fonctions a été modifié pour devenir un modèle de composition d'activités dynamiques. Des attributs ont aussi été associés aux éléments de la dimension structurelle (vues fonctionnelle et exécutive) et de la dimension comportementale.

Ce chapitre a pour objectif de décrire formellement le modèle de performance de MCSE. Nous commençons par préciser la signification du terme *performance* dans le contexte considéré. Nous définissons aussi une liste de critères de qualité d'un modèle de performance utile pour juger la pertinence de notre modèle. Comme ce modèle est composé entre autres du modèle fonctionnel et du modèle exécutif de MCSE, nous décrivons ensuite ces deux modèles dont l'association constitue la *vue structurelle* ou dimension organisationnelle du système. La vue structurelle décrit les éléments actifs (fonction, processeur) d'un système et leurs interconnexions. Le modèle de performance comporte également une *vue comportementale* qui est complémentaire et orthogonale à la vue structurelle. Aussi, nous présentons le modèle de comportement qui spécifie le séquencement temporel d'un ensemble d'activités. Les deux vues sont enrichies avec des *attributs* pour spécifier les caractéristiques de chaque élément. Nous décrivons donc le rôle des principaux attributs. Un exemple d'application du modèle de performance est donné en fin du chapitre.

3.1 LES PERFORMANCES

3.1.1 Définition

Le terme performance est très souvent utilisé sans que la signification soit claire. Il est par exemple usuel de l'employer pour décrire la qualité d'un système à satisfaire un objectif demandé.

Nous définissons le terme performance comme une quantification d'un système vis à vis de critères d'observation externes ou internes. Ainsi par performance d'un système, nous entendons ici des performances globales telles que la capacité de traitement, le taux d'utilisation de ressources, le rendement, etc., mais aussi des performances locales ainsi que des contraintes temporelles impératives en particulier pour les systèmes temps réel: temps maximum de réaction à des événements, fréquence d'activation d'une tâche par exemple.

Les contraintes de performances sont obligatoirement des grandeurs quantitatives et mesurables déterminées par des valeurs numériques. Ces valeurs quantifient la qualité d'un aspect particulier du système placé dans un contexte donné d'exploitation.

3.1.2 Classification des performances

Une première approche consiste à classer les performances en 2 catégories:

- les *performances statiques* qui sont des exigences indépendantes du temps. Citons comme exemples: La capacité mémoire nécessaire, la consommation maximale, le poids, l'encombrement, le coût, etc.
- les *performances dynamiques* qui spécifient les caractéristiques d'évolution temporelle du système dans son environnement (contraintes de temps, de débit, etc), ainsi que les caractéristiques internes (taux d'utilisation de ressources internes, disponibilité d'un bus, etc).

Les performances statiques peuvent se déterminer à l'aide d'estimateurs (calcul analytique et/ou heuristique). Par contre l'estimation des performances dynamiques nécessite l'utilisation de modèles analytiques, d'un prototype ou d'un modèle de simulation. La complexité actuelle des systèmes sort souvent du domaine d'application stricte des modèles analytiques. Le prototypage est aussi trop cher et/ou trop long à mettre en oeuvre. De plus, il ne peut intervenir que tard dans le cycle de développement. Pour estimer les performances dynamiques d'un système, le concepteur a donc recours de plus en plus à la simulation.

Par la suite, nous nous intéressons principalement aux performances dynamiques d'un système qui peuvent aussi se structurer selon 2 catégories:

- les *performances externes* au système telles que:
    - débit en entrée du système,
    - débit en sortie du système,
    - temps de réponse ou de réaction sortie(s) par rapport à entrée(s), temps de latence,
    - contraintes d'interactivité,
    - précisions et erreurs tolérées,
    - capacité globale du système.
- les *performances internes* au système telles que:
    - taux d'utilisation des ressources internes: processeur, ligne de communication, bus, etc.
    - précisions et erreurs tolérées.

Les spécifications externes correspondent à une appréciation externe du comportement du système considéré comme une "boite noire", basée sur l'évolution temporelle ou fréquentielle des entrées et des sorties. Ces performances sont de type contraintes de temps ou contraintes de capacité ou de débit.

Les spécifications internes, qui peuvent surprendre car faisant référence à des éléments de la conception, servent à fixer des contraintes d'utilisation de la structure ou d'un élément de celle-ci.

3.2 CRITERES DE QUALITE D'UN MODELE DE PERFORMANCE

Un modèle est construit par l'assemblage de concepts de modélisation. Un modèle efficace doit posséder un ensemble de concepts de modélisation restreint mais suffisant pour décrire facilement n'importe quel système. Le modèle doit aussi faciliter la transition entre les différentes phases de conception et la traçabilité des informations. Nous considérons qu'un modèle est de qualité s'il répond aux différents critères ci-dessous qui sont donnés sans classification d'importance. Qu'il soit de qualité ou non, un modèle seul n'est pas suffisant. Il doit toujours être accompagné d'une démarche favorisant l'élaboration de solutions. Notre modèle de performance est donc intégré à la méthodologie de conception MCSE.

-A- Efficacité de modélisation

Ce premier critère essentiel concerne la facilité et la puissance de représentation du modèle de description pour exprimer les concepts à modéliser. Il s'agit de disposer d'un moyen permettant de décrire d'une manière succincte et efficace, la structure et le comportement du système avec une souplesse de modification. L'évaluation pour un tel critère n'est pas simple car il s'agit d'une perception relativement subjective et qui en plus dépend des concepteurs.

-B- Efficacité de déduction

Pour disposer d'une continuité des spécifications jusqu'au produit final, ce critère spécifie l'aptitude du modèle à représenter la solution à divers niveaux ainsi qu'à permettre des transformations de niveaux. La traçabilité entre modèles de différents niveaux doit être gérée.

-C- Efficacité d'évaluation

Comme l'objectif est de déduire rapidement les propriétés utiles de la solution décrite, il faut pouvoir disposer d'un moyen efficace et fiable d'extraction des données objectives essentielles. Pour cela, le modèle doit être paramétrable (données ou attributs associés aux constituants et aux relations). Une méthode efficace d'évaluation basée sur la simulation pour les propriétés dynamiques doit également lui être associée de manière à avoir un faible temps d'analyse-correction.

-D- Efficacité de compréhension

Durant le développement, un modèle sert d'intermédiaire entre plusieurs stades de conception. Aussi, il est utilisé par les concepteurs pour dériver ou induire un modèle plus détaillé. Les critères de lisibilité et de compréhensibilité sont donc importants. Cette lisibilité doit aussi se retrouver pour les descriptions produites par les outils associés au modèle. En particulier les programmes générés doivent être simples, lisibles, compréhensibles car leurs mises au point nécessitent généralement une interprétation et compréhension détaillée par le concepteur.

Pour satisfaire les critères ci-dessus, le modèle de performance doit au minimum avoir les propriétés suivantes:

- *Modèle hiérarchique*. Cette propriété permet de modéliser des systèmes complexes en les décomposant en parties plus simples. Les techniques de raffinement, d'encapsulation (ou abstraction) et de réutilisation de modèles existants sont ici essentielles.

- *Modèle d'interdépendance*. Un système est souvent la composition de composants fonctionnels et/ou physiques. Ces composants sont concourants et inter-agissent entre eux par des liens de connexion. Le modèle au niveau système doit permettre d'exprimer ce parallélisme et les dépendances temporelles avec des mécanismes de synchronisation et de communication.

- *Modèle graphique*. La compréhension est facilitée si l'on utilise un modèle graphique qui permet une représentation à au moins deux dimensions. Au besoin, il est possible de représenter et interpréter sur le même graphique la description structurelle d'un système (dimension organisationnelle) et l'évolution temporelle de ses composants (dimension temporelle).

- *Modèle paramétrable*. l'évaluation des performances est facilitée si plusieurs paramètres permettent de configurer le modèle. Pour cela, il est utile d'exprimer les propriétés locales d'un modèle par des attributs associés aux éléments du modèle.

- *Traçabilité*. La continuité du modèle doit faciliter la transition d'une phase de conception à l'autre et ainsi améliorer la traçabilité.

3.3 LES CONCEPTS DU MODELE DE PERFORMANCE DE MCSE

Pour disposer d'une démarche de développement intégrant au mieux la maîtrise des performances simultanément à la maîtrise des fonctionnalités, une bonne solution est de disposer d'un modèle unique permettant de dériver par transformations et enrichissements successifs, à partir des spécifications, une solution de conception puis une solution d'implantation et simultanément de décrire et d'évaluer les propriétés de performances à chaque stade.

Analysons tout d'abord le modèle préconisé dans la méthodologie MCSE pour la spécification et pour la conception fonctionnelle de manière à déduire ses limitations et manques. Cette analyse, décomposée en une analyse de la composante structurelle puis de la composante comportementale, servira de base pour décrire les concepts complémentaires pour disposer d'un modèle approprié et efficace pour les 2 rôles indiqués ci-dessus.

3.3.1 Analyse du modèle structurel

Le modèle structurel permet de décrire les constituants internes d'un système et les interconnexions entre ces constituants. Il est utilisé pour l'étape de conception fonctionnelle et celle de conception architecturale de la méthodologie MCSE. En effet, en donnant des significations différentes aux symboles, on peut à la fois décrire une structure fonctionnelle et une structure d'exécution. Comme le montre la figure 3.1, la différence entre les 2 structures

réside simplement dans la représentation et la signification des constituants. La signification du modèle fonctionnel peut aussi être élargie en considérant des significations différentes aux éléments du modèle. Sa signification peut au moins s'étendre à la description des architectures système, à la modélisation des ateliers, etc.

-A- Le Modèle fonctionnel

Le modèle fonctionnel représente une application selon un ensemble de fonctions qui inter-agissent à l'aide de trois types de relations. Les relations inter-fonctions représentent trois types d'échanges :

- la **relation de partage de variables** (par variable d'état) permet à plusieurs fonctions de partager une donnée ou une ressource sans aucune relation d'ordre. La seule contrainte imposée concerne l'accès à la donnée de façon à conserver l'intégrité de l'information.
- la **relation de synchronisation** (par événement) représente une relation de précédence d'exécution de deux fonctions: la fonction réceptrice ne peut s'exécuter que si l'événement d'activation a été émis.
- la **relation de transfert d'informations** (par un port de communication) permet d'échanger de l'information sous forme de messages. Elle introduit une relation du type producteur/consommateur entre les fonctions. Ainsi, une fonction consommatrice ne devient active que lorsqu'il y a au moins un message dans le port auquel elle accède.

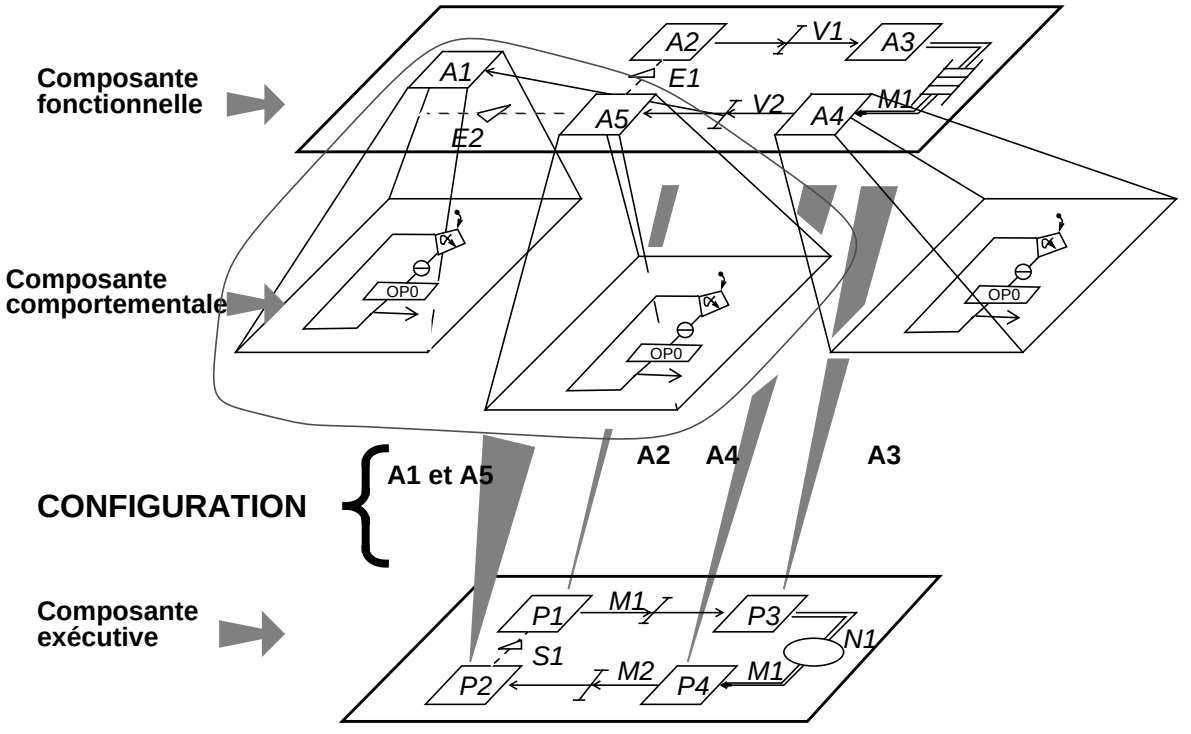
Hiérarchique et graphique, le modèle fonctionnel facilite la recherche progressive d'une solution par transformations de comportements en structures. En effet, une fonction décrite sous forme comportementale peut aussi par raffinement se décrire par une organisation interne. Les éléments de relation possèdent chacun un comportement précis. Il en résulte que toutes les propriétés temporelles du modèle fonctionnel découlent essentiellement des propriétés du modèle comportemental des fonctions.

-B- Le modèle exécutif

Le modèle exécutif est utilisé pour décrire la partie matérielle d'une application. Ce modèle est similaire, en terme de structure, au modèle fonctionnel mais avec une spécification différente des éléments (voir figure 3.1). Il est basé sur quatre éléments : le processeur, le signal inter-processeurs, la mémoire commune et le noeud de communication qui permet aux processeurs d'échanger des informations sous forme de messages.

-C- La configuration

Les modèles fonctionnel et exécutif considérés séparément ne sont pas suffisants pour décrire la solution d'un système. Il est nécessaire de définir l'allocation (ou mapping) des éléments fonctionnels sur les éléments exécutifs. Cette phase de configuration correspond à l'étape de partitionnement et d'allocation en co-design.



Signification des symboles

Structure fonctionnelle	Structure d'exécution
VARIABLE PARTAGEE	MEMOIRE
EVENEMENT	SIGNAL
ACTION	PROCESSEUR
PORT	NOEUD DE COMMUNICATION

-Figure 3.1- Modèle de description des solutions pour la méthodologie MCSE.

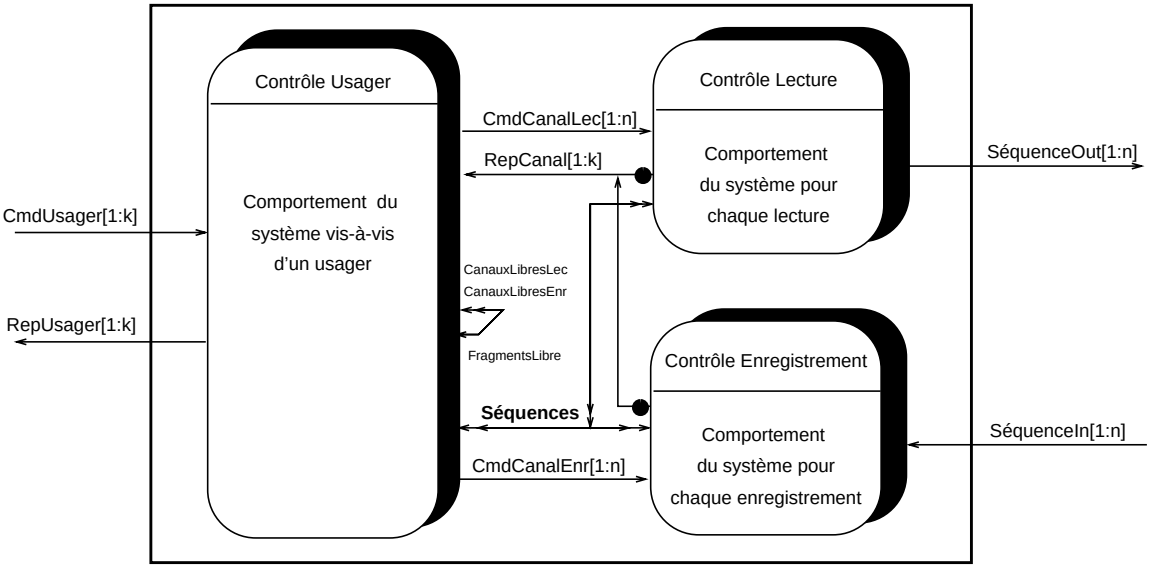
3.3.2 Analyse du modèle comportemental

Par modèle comportemental, nous entendons ici le modèle conseillé pour la spécification, à savoir l'association du diagramme des activités et du StateChart. Le statechart peut aussi être remplacé par un algorithme séquentiel.

Le diagramme des activités favorise la structuration du comportement global de l'élément considéré (système, fonction) en comportements partiels et décrit clairement les interactions entre parties. Le StateChart décrit très précisément un déroulement temporel de chaque activité sous la forme d'un ensemble d'états, de conditions et d'actions. Mais toutes les actions du StateChart sont considérées exécutées en temps nul (hypothèse synchrone), ce qui implique

que les dépendances temporelles sont exclusivement liées à l'évolution de l'environnement (comportement réactif). Rien ne permet de modéliser des caractéristiques temporelles internes sinon l'emploi de variables Temps. En effet, en spécification, la durée d'un état ne représente pas un temps d'exécution mais un temps d'attente d'une condition de fin de l'état.

Le diagramme d'activités de la figure 3.2, représente le résultat de l'étape de spécification sur l'exemple du serveur vidéo temps-réel présenté dans le chapitre 7. Durant cette étape, le concepteur détermine la liste des fonctions du système et le comportement du système (StateChart, FSM ou algorithme séquentiel) pour ces fonctions. Les fonctions de service du système pour son environnement sont pour l'exemple considéré: la lecture d'une séquence vidéo, l'enregistrement de séquence et la gestion du serveur. Le diagramme d'activités fait clairement apparaître la multiplicité des activités de contrôle du système et l'importance de la grandeur Séquences. C'est autour de cette grandeur qui modélise l'ensemble des disques du système que la solution fonctionnelle a ensuite été organisée lors de l'étape de conception fonctionnelle. Les doubles flèches sur les grandeurs internes en entrées et sorties signifient qu'il s'agit de données permanentes.



-Figure 3.2- Exemple de diagramme d'activités.

3.3.3 Les concepts complémentaires pour les performances

La modélisation de performances nécessite d'ajouter la notion de temps d'exécution des opérations ou des actions pour pouvoir extraire des propriétés temporelles globales d'un système (vues par ses entrées et sorties) à partir des propriétés temporelles locales. Les concepts suivants servent, d'une part à exprimer ces propriétés, d'autre part à disposer d'un modèle facilitant la transition progressive des spécifications en une solution interne détaillée.

-A- Association du modèle de structure et du modèle de comportement

Le premier aspect intéressant est de constater qu'il est possible d'exploiter simultanément le modèle de structure fonctionnelle et le modèle comportemental rappelés ci-dessus. En effet, le modèle résultant va de cette manière servir comme modèle intermédiaire entre une spécification fonctionnelle pour laquelle la structure fonctionnelle n'existe pas encore et la

description fonctionnelle résultat de la conception fonctionnelle et qui elle, utilise une description une description structurelle. Ainsi, par itérations successives durant la conception, le parallélisme exprimé dans la spécification de départ est transformé en un parallélisme de fonctions.

-B- Extension du modèle fonctionnel

Chaque fonction du modèle fonctionnel est une unité d'encapsulation. Pour l'analyse des performances, une fonction est considérée comme une ressource d'exécution servant de support pour les activités et opérations qu'elle inclut. Ainsi une fonction sera définie par des attributs représentatifs de son rôle. Indiquons par exemple la possibilité de limiter le degré de parallélisme des opérations internes et dans ce cas, il faut alors définir la politique d'ordonnancement des opérations.

Les éléments de relation que sont les événements, les variables d'état, les ports sont aussi caractérisables par des spécifications temporelles que sont par exemple les temps de lecture et d'écriture, le degré d'accès simultanés.

-C- Extension du modèle comportemental

Le modèle comportemental préconisé pour les spécifications est uniquement statique pour les activités, c'est-à-dire que toutes les activités du diagramme des activités existent à tout instant, chacune étant dans un état dépendant de son comportement propre et de son environnement. Le terme action est généralement limité à une intervention ponctuelle considérée comme instantanée. En fait, en évoluant vers la description détaillée de la solution interne, la différence entre action et activité tend à se réduire: une action nécessite du temps pour son exécution, une activité exprime une intervention temporaire dans le système.

Ainsi, il est tout d'abord aisé d'ajouter la propriété temporelle de *temps d'exécution* pour une action et pour une activité. Lorsqu'une activité est considérée comme élémentaire, elle sera appelée *opération*.

Ensuite, il est intéressant de lever la restriction du caractère statique du diagramme d'activités. De cette manière, il sera possible de représenter des comportements internes plus complexes en exploitant la possibilité d'instanciation dynamique d'activités.

-D- Modèle générique et paramétrable

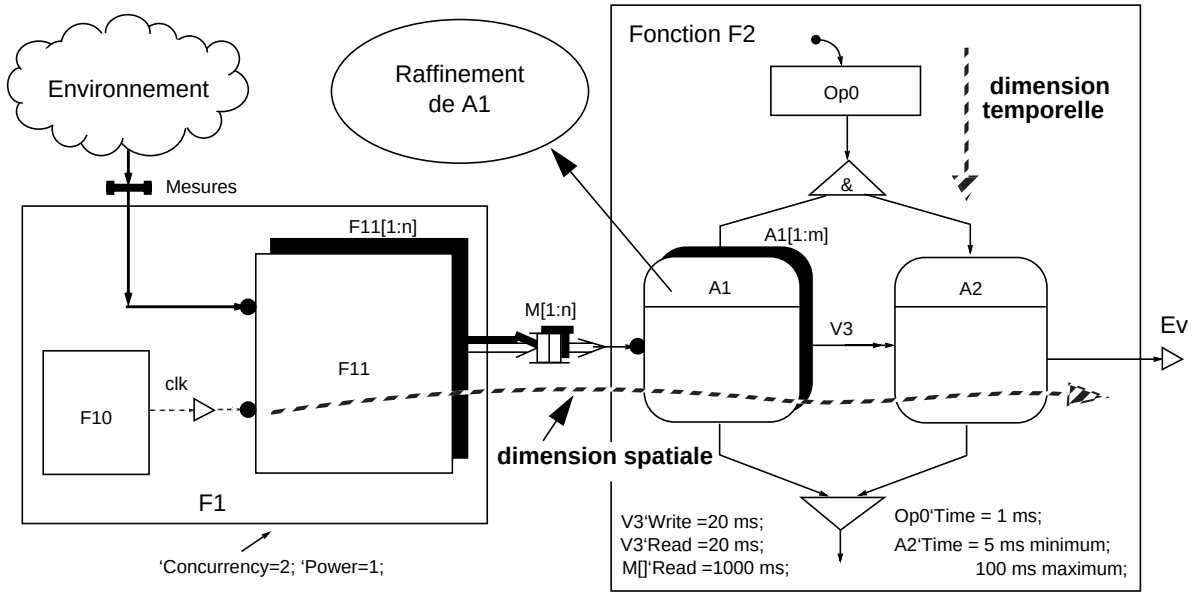
Pour dimensionner une architecture durant la conception, le concepteur doit pouvoir définir et modifier des paramètres dans le modèle. Ces paramètres concernent aussi bien la vue structurelle que comportementale. Pour le modèle structurel, nous considérons important de pouvoir disposer du concept de composants génériques. Ceci veut dire, l'emploi de vecteurs de composants. La taille du vecteur est alors un paramètre générique mais obligatoirement statique. Pour le modèle comportemental, les activités et opérations sont au contraire dynamiques et définissables aussi sous forme de vecteurs.

Tous les paramètres du modèle sont appelés des attributs. Certains doivent pouvoir se définir et se modifier par l'utilisateur. De cette manière, il pourra étudier leur influence sur les propriétés du système.

Il est aussi intéressant d'avoir la notion de modèle d'activité et de fonction, ce qui veut dire la réutilisation possible d'un modèle provenant d'un autre projet. Un modèle peut posséder des paramètres génériques qui définissent son emploi à l'instant de l'instanciation.

3.4 PRESENTATION GENERALE DU MODELE

Pour illustrer les principaux concepts introduits dans les paragraphes précédents, la figure 3.3 représente un exemple de modèle de performance.



-Figure 3.3- Principaux concepts du modèle de performance.

Le modèle structurel (partie gauche de la figure 3.3) est composé des fonctions Environnement, F1 et F2 et des éléments de relation Mesures et M[1:n] pour le premier niveau de description. F1 est raffinée par une structure comprenant F10, F11[1:n] et clk. n est un paramètre générique de la structure. Le modèle de comportement (partie droite de la figure 3.3) explique l'évolution temporelle de la fonction F2. Dans cette description, A1[1:m] est un vecteur d'activités dont la taille est définie par le paramètre générique m. A1 est supposée être raffinée. A[1:m] et A2 sont des activités concurrentes liées par la donnée permanente V3.

Comme le montre la figure 3.3, le modèle de performance est surtout graphique et basé sur deux vues complémentaires: la vue structurelle (dimension organisationnelle) et la vue comportementale (dimension temporelle).

- La *vue structurelle* permet de décrire les éléments actifs (fonction et même processeur) d'un système et leurs interconnexions. Les composants sont couplés entre eux par l'intermédiaire d'éléments de relation. Ils sont de 3 types pour la structure fonctionnelle (Evénement, Variable partagée, Port) et de 3 types pour la structure d'exécution (Signal, Mémoire commune, Noeud de communication). Le modèle structurel est hiérarchique (un élément actif peut être raffiné) et permet la réplication (instance d'un modèle).

- La *vue comportementale* permet de décrire le comportement des fonctions. Le modèle de comportement est un modèle non-interprété qui repose sur des activités décomposables par raffinement ou élémentaires (temps d'exécution) et des opérateurs de composition d'activités (séquence, parallélisme, alternative, répétition et attente conditionnelle).

Tous les éléments du modèle sont définis avec des *attributs spécifiques,* par exemple, le temps d'exécution (attribut 'Time) d'une activité élémentaire, le temps de lecture (attribut 'Read) et d'écriture (attribut 'Write) pour une variable partagée, la capacité d'un port (attribut 'Capacity), le degré de concurrence d'un processeur (attribut 'Concurrency), etc...

La figure 3.3 montre le modèle de comportement de la fonction F2 selon 2 axes: *l'axe temporel* du haut vers le bas qui représente le séquencement (expression du contrôle), *l'axe spatial* de la gauche vers la droite qui représente les échanges, les transactions, le flot de données, en un mot les interdépendances (expression du flux). La fonction F2 est décrite comme l'exécution de l'opération Op0 à partir de l'instant initial puis l'exécution simultanée des activités A1 et A2. Lorsque ces 2 activités sont achevées, la fonction F2 devient inopérante sans pour cela disparaître. Ainsi, il est possible de représenter des comportements plus généraux que le modèle cyclique et séquentiel d'une fonction du modèle fonctionnel initial de MCSE. Les activités A1 et A2 sont couplées entre elles par la donnée permanente V3 (notation du modèle de spécification) et sont aussi liées à l'environnement de la fonction par M et Ev. Les activités sont détaillées par le même type de modèle (raffinement) car le modèle est hiérarchique. Chaque opération (rectangle) est caractérisée par son temps d'exécution (attribut 'Time). Chaque activité est caractérisée par: son comportement interne, son temps d'exécution.

Il ne faut pas confondre *activité* (et opération) et *état,* ce qui justifie la notation légèrement différente. Le temps d'intervention d'une activité est essentiellement défini par les temps d'exécution internes. Une activité ou une opération implique l'utilisation d'une ressource d'exécution pour son intervention. Un état représente généralement une situation d'attente, ce qui veut dire absence d'utilisation de ressource d'exécution. La durée d'un état est définie par les conditions de passage dans d'autres états. Lorsque la durée d'un état est définie par un temps, ceci peut correspondre à une confusion entre état et opération.

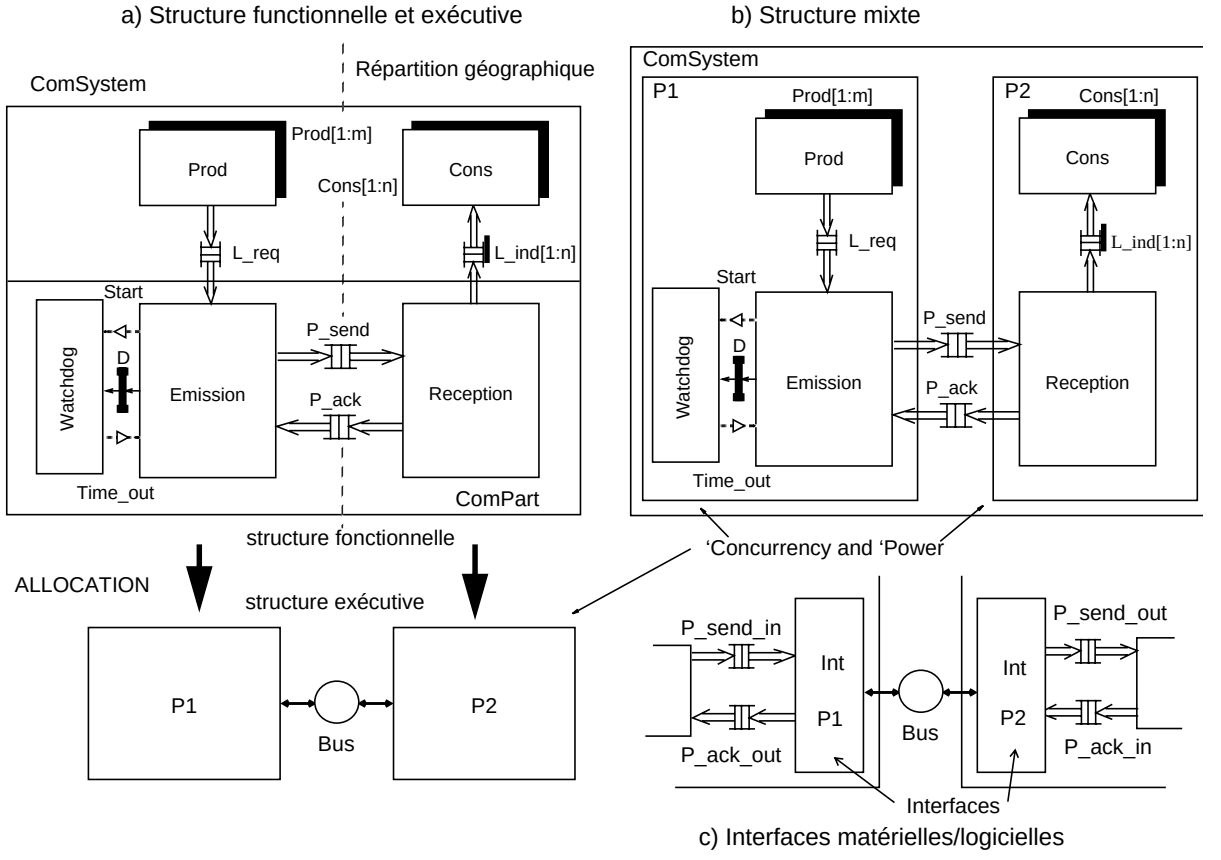
Une activité peut se décrire par un diagramme d'activités ou par un diagramme d'états. Un état ne peut se décrire que par un diagramme d'états plus élémentaire (raffinement du statechart).

3.5 LA COMPOSANTE STRUCTURELLE DU MODELE DE PERFORMANCE

Pour décrire les extensions du modèle structurel nécessaire pour l'évaluation des performances, un exemple d'illustration qui concerne un système de communication permet tout d'abord de montrer la représentation graphique pour la solution fonctionnelle et pour l'architecture matérielle. On montre aussi que les 2 structures peuvent être combinées en une seule pour décrire une solution complète avec son implantation. La figure 3.4-a représente une structure fonctionnelle et une structure d'exécution.

L'exemple considéré ici est un système simplifié de communication permettant le transfert de messages entre des producteurs Prod[1:m] et des consommateurs Cons[1:n]. La fonction Emission envoie un message reçu du port L_req vers la fonction Reception via le port P_send. Pour garantir un transfert correct, chaque message doit être acquitté via le port P_ack. Emission utilise une fonction de chien de garde pour limiter le temps d'attente de l'acquittement.

La structure d'exécution est composée de 2 processeurs P1 et P2 reliés entre eux par le noeud de communication appelé Bus.



-Figure 3.4- Modèle structurel pour la modélisation des performances.

La structure représentée par la figure 3.4-b, appelée *structure mixte ou architecturale*, regroupe les 2 points de vue fonctionnel et exécutif. La figure du haut représente les processeurs comme unités d'encapsulation. Elle ne représente pas le lien inter-processeur Bus qui doit supporter les 2 liens fonctionnels P_send et P_ack car il s'agit d'un modèle simplifié.

La différence de signification entre les relations du modèle fonctionnel et celle du modèle exécutif nécessite normalement de faire apparaître des interfaces qui sont des fonctions particulières pour assurer les couplages entre liens de modèles différents. On constate bien ici (figure 3.4-c) que le modèle exécutif sert de support. Ainsi un port tel que P_send ou P_ack est scindé en 2 parties: Port_X_in, Port_X_out. Puis on y ajoute 2 interfaces et un lien pour la communication.

Dans la suite, on ne fera pas obligatoirement de distinction entre les constituants du modèle fonctionnel et ceux du modèle exécutif. La distinction est intéressante à faire uniquement lorsqu'il s'agit de faire apparaître les interfaces entre le fonctionnel et l'exécutif.

Le modèle structurel, qu'il soit fonctionnel ou exécutif ou mixte, explicite les constituants et leur signification par leur nom et leur rôle, les échanges possibles entre constituants (signification et structure des données) mais n'indique rien quant au comportement du modèle car le comportement des fonctions et processeurs n'est pas défini.

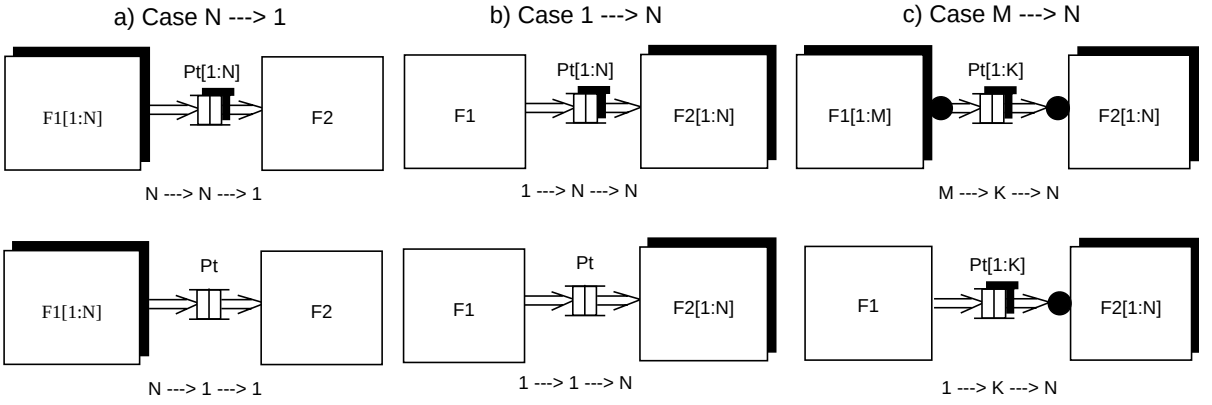
D'autre part, des instanciations multiples de fonctions ou d'éléments de relation sont possibles. On détaille dans la suite ces 2 aspects.

3.5.1 Généricité et instanciations multiples

Un constituant actif ou de relation peut être instancié plusieurs fois, ceci pour représenter facilement une organisation exploitant de multiples objets d'un même type (replication). Il s'agit alors de vecteurs de fonctions, de ports, de variables partagées, d'événements, de noeuds.

Un constituant actif multiple est représenté par son symbole avec son ombre. Sa multiplicité est décrite comme un vecteur défini par l'indice de début et l'indice de fin. Un élément du vecteur est désigné par son nom et son indice. Ces indices peuvent être des constantes ou des variables (paramètre générique expliqué plus loin). Le mot générique veut dire qu'un indice est défini par un nom symbolique et la valeur est fixée par un objet englobant. Ceci permet de disposer de la notion de modèle générique pour l'instanciation. La même définition est utilisée pour les vecteurs de constituants de relation.

Se pose le problème des règles d'interconnexion entre composants actifs et éléments de relation. Les cas suivants représentés par la figure 3.5 sont à considérer: N ->1, 1 -> N, N -> N, N -> M.



-Figure 3.5- Les différents cas de l'instanciation multiple.

-A- Cas N -> 1

La valeur N ou 1 peut concerner aussi bien un objet actif qu'un objet de relation. Le premier cas ci-dessus indique la correspondance F1[i] -> Pt[i] avec i allant de 1 à n, puis la visibilité de tous les Pt[1:N] par F2. Le deuxième cas en dessous signifie que toutes les fonctions F1[i] exploitent en écriture un seul port Pt en respectant son degré de partage.

Cette signification est similaire pour les relations par synchronisation et pour les éléments partagés.

-B- Cas 1 -> N

Il s'agit ici de la relation de distribution de 1 vers N. Le premier cas spécifie que F1 peut utiliser l'un quelconque des ports du vecteur Pt[1:N] en le désignant par son indice. Le deuxième cas en dessous indique que toutes les fonctions F2[i] peuvent faire un retrait de Pt.

-C- Cas M -> N

Il s'agit ici de la relation de distribution de M vers N qui permet une interconnexion complète. Le premier cas spécifie toujours que F1 peut utiliser l'un quelconque des ports du vecteur Pt[1:K] en le désignant par son indice, mais chaque F2[i] peut accéder à tous les éléments du vecteur Pt[1:K] (signification du rond noir pour la relation multiple). Le deuxième cas en dessous indique que chaque fonction F2[i] peut accéder à tous les éléments de Pt.

3.5.2 Comportement macroscopique de la structure

Il s'agit de spécifier le comportement directement et obligatoirement induit par la structure. L'expression de ce comportement s'effectue simplement en ajoutant des attributs aux éléments de la structure. Pour cela, on passe en revue ci-après les 4 types d'éléments. Les attributs décrits ci-après existent obligatoirement; chacun possède une valeur par défaut (valeur donnée entre parenthèses). D'autres attributs peuvent être ajoutés par l'utilisateur pour enrichir sa description selon l'objectif qu'il vise.

-A- Constituant actif

Une fonction, un processeur, un système, c'est-à-dire un constituant structurel actif peut se spécifier globalement par son comportement qui est alors macroscopique. Un premier paramètre concerne sa vitesse d'exécution pour tous les constituants inclus qui correspond à sa puissance exprimée par rapport à l'unité (puissance relative). Ensuite, le comportement du constituant est défini par son degré de parallélisme pour l'ensemble de ses constituants internes. Ce paramètre peut évoluer de 1 qui représente le fonctionnement séquentiel à l'infini (*). Le séquentiel veut dire que, à tout instant, une seule fonction, opération, activité inclue est active. L'évolution séquentielle permet par exemple de caractériser le déroulement d'un ensemble de fonctions sur un processeur séquentiel.

Lorsque le nombre de constituants internes actifs est plus élevé que le degré de parallélisme, il est alors nécessaire de spécifier la politique de déroulement. Parmi les possibilités, citons:

- ordonnancement préemptif, non-préemptif, partage équitable ou time sharing (PS, NPS, TS),
- ordonnancement par priorité, par date critique ou deadline (P, D) pour le préemptif et le non-préemptif,

Ainsi, les attributs associés au constituant actif que nous avons retenus sont:

- '*Power* qui est une valeur de la puissance relative (nombre en flottant), (1 par défaut)
- '*Concurrency*, qui est un nombre entier positif, (0 pour la valeur infinie)
- '*Policy*: (PSP, PSD, NPS, TS), (PSP)
- '*Overhead*: un temps, (0)
- '*Level*: (Global, Local, Processor, Function, Activity, Operation), (Function)
- '*Priority*: un nombre entier croissant dans le sens de la priorité, (1)
- '*Deadline*: un temps, c'est-à-dire valeur et unité. (0)

Les 3 premiers concernent le constituant servant de support d'exécution pour la structure correspondant à son raffinement. L'attribut 'Power sert à faire varier simultanément les temps d'exécution de tous les constituants actifs inclus dans le constituant selon un coefficient qui représente la puissance en tant que ressource d'exécution. La valeur neutre est 1. L'attribut 'Concurrency permet de contrôler le degré de parallélisme à l'exécution. On peut donc de cette manière décrire l'effet d'un microprocesseur comme support exécutif pour un ensemble de fonctions, activités, opérations, et étudier l'influence de sa vitesse d'horloge en agissant sur 'Power. L'attribut 'Overhead permet d'introduire le temps de commutation entre 2 tâches (fonction ou activité) lors du partage de la ressource d'exécution. L'attribut 'Level est optionnel et permet d'indiquer le niveau ou la catégorie de l'élément dans la structure complète. Les 2 derniers attributs concernent le constituant lorsqu'il n'est pas raffiné.

-B- Synchronisation (événement, signal)

Une synchronisation peut être du type booléen, du type compteur ou du type fugace. Dans le deuxième cas, toutes les générations vont engendrer une évolution (ceci se traduit par une liste pour son implantation). Pour le type fugace, il n'y a pas mémorisation de l'événement si un constituant actif destinataire n'est pas en attente. Ce dernier cas permet de modéliser le comportement des fonctions sur une entrée servant non pas d'activation mais de condition d'évolution.

Une synchronisation est aussi caractérisable par les temps de génération de l'événement, d'activation des correspondants et par son degré de partage pour l'accès.

Les attributs intéressants sont les suivants:

- '*Policy*: (Boolean, Counter, fugitive), (Boolean)
- '*Concurrency*: un entier positif, (1)
- '*Write*: un temps, (0)
- '*Read*: un temps. (0)

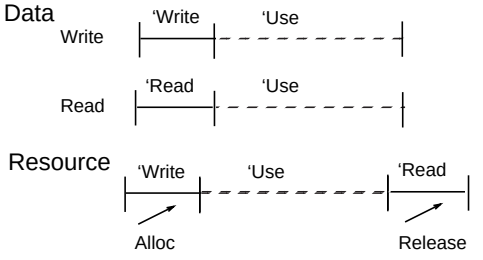
-C- Elément partagé (Donnée ou ressource commune)

Une donnée ou ressource commune est caractérisée par son degré de partage et sa capacité en nombre de bits ou de mots (grandeur statique ou dynamique). 2 temps sont à considérer pour son utilisation: allocation ou écriture/modification, libération ou lecture. Pour enrichir la signification, il est souhaitable de différencier: Données et Ressources. Pour la donnée qui a une signification au niveau fonctionnel, le degré de partage concerne les accès simultanés en lecture et modification. Pour une ressource qui a plus une signification au niveau exécutif, le degré de partage concerne l'intervalle entre allocation et libération.

De plus, il est intéressant d'ajouter 2 types de politiques en cas de concurrence d'accès: selon la priorité ou selon l'ordre des demandes et donc Fifo.

Les attributs retenus sont les suivants:

- '*Policy*: (DataFifo, DataPriority, ResourceFifo, ResourcePriority), (DataFifo)
- '*Concurrency*: un entier positif, (1)
- '*Capacity*: un entier positif, (1)
- '*Write*: un temps, (0)
- '*Read*: un temps, (0)
- '*Use*: un temps (0).



Le comportement pour une donnée permet de modéliser l'exploitation d'une donnée complexe, y compris sa lecture et écriture sur un support de masse. Le temps 'Use représente alors le temps d'écriture ou d'accès pour la lecture, tandis que les temps 'Write et 'Read représentent les temps d'échange avec le contrôleur du support de masse (exploitation d'un cache par exemple). Pour une ressource, il s'agit de sa période d'utilisation, le temps 'Use est non-significatif.

-D- Transfert d'information ou d'objet, communications

L'objet de couplage est assimilable à un tampon caractérisé par sa capacité qui peut varier de 0 (rendez-vous) à l'infini (*) et peut aussi être du type fugace. Le type fugace veut dire qu'un message n'est pas mémorisé si aucune fonction n'est en attente. Par défaut, la capacité est infinie. 2 temps sont aussi à considérer pour son utilisation ceci dans le cas de place disponible pour le dépôt, ou information disponible pour le retrait. Il faut y ajouter son degré de partage pour des accès multiples. Le service assuré par le port, c'est-à-dire l'ordre de rangement des messages, sera aussi définissable selon une politique Fifo ou selon la priorité des producteurs.

Dans le cas d'un lien physique de communication ou de transfert, qui lui peut être bidirectionnel ou unidirectionnel, le degré de partage permet de différencier entre liens full-duplex ou half-duplex.

Les attributs retenus sont les suivants:

- '*Policy*: (Fifo, Priority), (Fifo)
- '*Concurrency*: un entier positif, (1)
- '*Capacity*: un entier positif ou nul ou * ou fugitive, (*)
- '*Write*: un temps, (0)
- '*Read*: un temps. (0)

-E- Attributs pour les liens de connexion

Des attributs peuvent aussi être ajoutés sur les liens liant les composants aux éléments de relations. Pour cela, les attributs sont définis à l'intérieur du composant et ceci en utilisant le nom de l'entrée, de la sortie ou de l'entrée/sortie.

Ces attributs peuvent être utilisés pour modifier (forme de surcharge) les attributs des éléments de relation car ils sont alors considérés prioritaires. En effet, si l'attribut concerne par exemple le temps d'écriture ('*Write*), ce temps vient en remplacement du temps d'écriture de l'élément connecté. Cette possibilité permet de modifier sélectivement les paramètres du comportement du modèle de structure. Les 3 attributs particulièrement concernés sont donc *'Write*, *'Use* et *'Read*.

-F- Attributs libres

Pour permettre toute forme d'évaluation sur la structure, l'utilisateur peut ajouter les attributs qu'il souhaite pour les éléments de la structure. Ceci se traduit pour chacun par l'emploi d'un nom d'attribut, d'une valeur, d'une unité.

De tels attributs permettront de faire, en autres, des évaluations statiques de la solution, tels que: la consommation, la surface, le coût, etc.

-G- Paramètres et vecteurs génériques

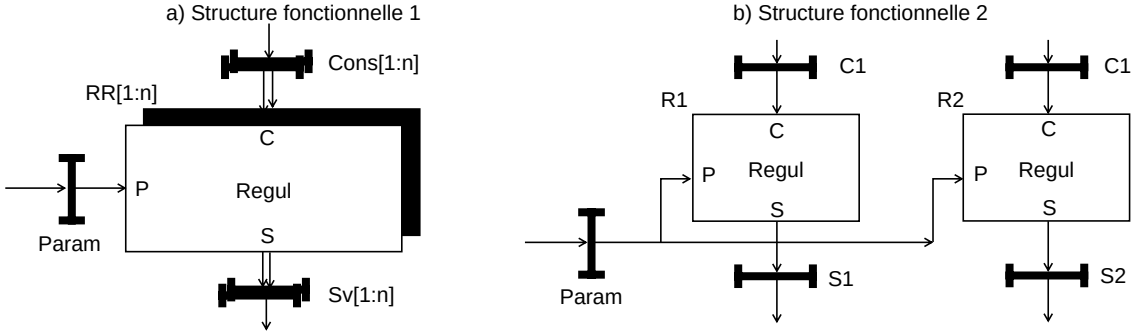
Une description peut comporter des paramètres génériques de manière à dimensionner l'application pour plusieurs grandeurs. Ceci est intéressant pour des instances multiples de fonctions, d'activités, d'éléments de relation. Par suite de ce caractère générique, il faut définir la notation pour les vecteurs. [] veut dire l'élément dans le vecteur du même indice que l'objet englobant. [:] veut dire tout le vecteur et ses bornes sont celles de sa définition.

3.5.3 Utilisation de modèles

La description d'un composant doit aussi pouvoir se faire à partir de composants déjà décrits. Ceci doit favoriser la réutilisation en conception.

Pour ce faire, un composant est considéré comme un type (ou une classe, similaire à la notion de symbole pour un éditeur de schémas). Un tel composant est défini par son nom de type, ses entrées et sorties formelles, sa description interne. L'emploi du composant conduit à créer une ou plusieurs instances chacune ayant un nom pour l'application.

La figure 3.6 montre 2 exemples de description avec l'emploi d'une fonction REGUL comme modèle.



-Figure 3.6- Exemples de structures fonctionnelles utilisant un modèle.

Chaque composant servant de modèle est déclaré avec son nom de modèle (Regul), ses entrées et sorties formelles (P, C, S), sa description interne. Les composants instanciés sont déclarés avec un nom d'instance (RR[1:n], R1, R2) et le nom du modèle. Les noms utilisés pour les entrées et sorties de l'instance correspondent aux connexions réelles dans la structure englobante (Param, Cons[1:n], Sv[1:n] ou bien S1, S2, C1, C2).

Concernant les paramètres génériques, lorsque le modèle contient des vecteurs d'instances, la taille d'un ou plusieurs vecteurs internes peut être dynamique. Dans ce cas, l'exploitation du modèle nécessite de spécifier la taille de ces vecteurs. Ceci se fait par l'intermédiaire de paramètres génériques déclarés à la fois pour le modèle et pour l'instance. De cette manière, il est possible de définir une correspondance entre les noms donnés dans la description englobante et les noms donnés dans le modèle.

Cette facilité sera aussi utilisée pour définir des attributs génériques qui seront fonction d'un ou plusieurs paramètres généraux.

3.6 LA COMPOSANTE COMPORTEMENTALE DU MODELE DE PERFORMANCE

Le modèle de comportement définit les règles de construction que doit respecter toute description comportementale d'un composant actif utilisé dans le modèle structurel. A noter que pour le concepteur, tout constituant pourra posséder plusieurs descriptions aussi bien structurelles que comportementales. Le point de vue du comportement d'un composant est une vision orthogonale au point de vue structurel.

Lors de l'étape de spécification, un modèle de comportement est souvent basé sur un diagramme d'activités, un StateChart ou un automate à états finis. Le diagramme des activités favorise la structuration du comportement global de l'élément considéré (système, fonction) en comportements partiels et décrit clairement les interactions entre parties. Le StateChart décrit très précisément un déroulement temporel de chaque activité sous la forme d'un ensemble d'états, de conditions et d'actions. Toutes les actions du StateChart sont considérées exécutées en temps nul (hypothèse synchrone).

Durant le processus de conception qui consiste en une succession de transformations de la spécification initiale en un modèle de plus en plus structurel, lorsqu'on s'approche de la solution interne d'un système, la différence entre action et activité tend à se réduire: une action demande du temps pour son exécution et une activité joue un rôle temporaire dans le système. Par conséquent, la modélisation des performances durant la conception nécessite l'ajout du concept de temps d'exécution pour les actions et les opérations afin d'extraire des propriétés temporelles du système. A notre point de vue, le StateChart (et également le SpecChart) est un modèle trop détaillé pour l'évaluation des performances.

Dans le modèle de comportement que nous avons retenu, nous avons éliminer le caractère statique d'une activité d'un diagramme d'activité ou d'un flot de données. De cette façon, il est plus facile de représenter le comportement interne des systèmes complexes en utilisant la possibilité d'instanciation dynamique d'activités.

Dans la suite, nous présentons tout d'abord globalement le modèle comportemental par un exemple. Les règles de description sont ensuite expliquées.

3.6.1 Un exemple de modèle comportemental

Comme pour le modèle structurel, le modèle comportemental comprend 2 parties:

- la description des éléments internes et les dépendances temporelles entre eux,
- la description des propriétés de tous les éléments, aussi appelées attributs.

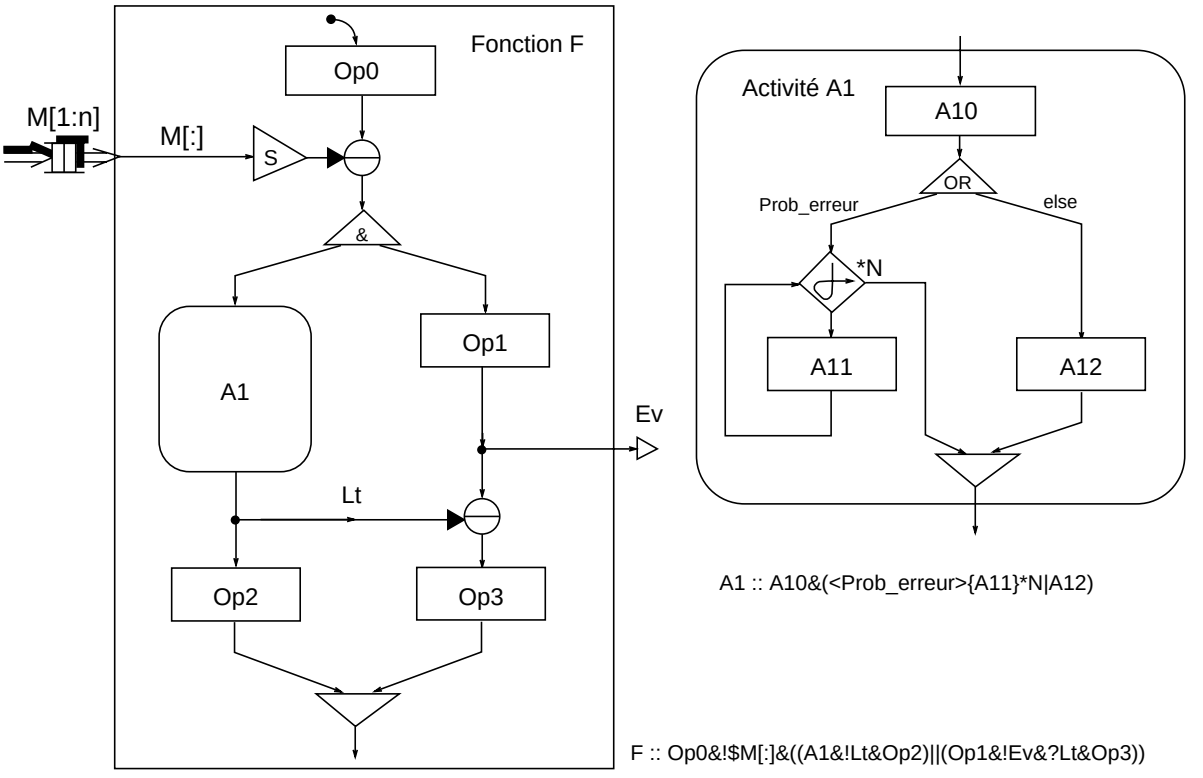
La description du comportement est décrite sous une forme graphique représentant le déroulement temporel selon l'axe vertical ceci pour faciliter la compréhension. Un comportement est construit sur la base d'opérations (rectangles) et d'activités (rectangle à coins arrondis). Les déroulements divergents ou simultanés apparaissent comme des branches parallèles verticales. Les dépendances temporelles, c'est-à-dire les synchronisations, les communications et les sorties, sont représentées horizontalement. Cette description est hiérarchique car une activité peut se raffiner. Une activité élémentaire est appelée opération.

La figure 3.7 montre un exemple de représentation du comportement d'une fonction et d'une activité.

La fonction F commence par une opération Op0 et se met ensuite en attente d'un message sur l'un des ports du vecteur M. Ce vecteur de ports est noté M[:] lorsqu'il s'agit d'un vecteur non contraint. L'opérateur avec la lettre S est un opérateur de sélection. Il y a ensuite activation simultanée de 2 branches avec une synchronisation par Lt. Les 2 branches sont achevées lorsque Op2 et Op3 sont achevées. On notera que la génération des sorties n'est pas associée aux opérations mais après celles-ci.

A1 est une activité raffinée. Elle inclut une évolution selon 2 branches exclusives avec une sélection aléatoire définie par une probabilité. L'une des branches inclut une boucle représentant l'exécution multiple (ici N fois) de l'opération A11.

A cette représentation graphique correspond une description textuelle donnée sur la figure 3.7 et simple à interpréter. Le texte suit l'évolution temporelle. Des symboles spécifiques sont exploiter pour représenter, d'une part l'enchaînement (structure de contrôle), d'autre part les actions associées aux éléments de synchronisation ou de communication. La notation sera expliquée dans les paragraphes suivants.



-Figure 3.7- Description comportementale d'une fonction et d'une activité.

Une description temporelle est la composition d'opérations ou d'activités (flux vertical) conformément à 5 règles d'association:

- la séquence, (&)
- l'alternative, (|)
- la simultanéité, (||)
- la répétition, ({})
- l'activation conditionnelle multiple ([?-&- | ?-&-]).

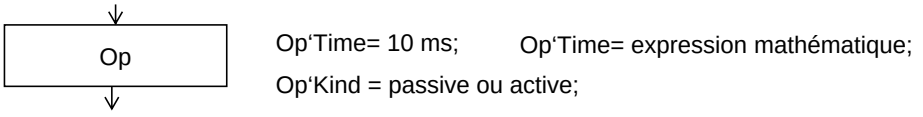
Des règles d'interaction par les symboles ? et ! sont ajoutées pour exprimer les interdépendances par échange de données ou d'informations (flux horizontal).

Le modèle qui en résulte est un modèle dit non-interprété car les valeurs des données et des informations pour l'application n'interviennent pas dans la description. Ceci ne veut pas dire que les variables d'état et les informations ne contiennent pas de données. Elles peuvent contenir des valeurs d'attributs qui sont significatifs comme information abstraite pour la modélisation de performance. Comparativement à l'utilisation du StateChart, il s'agit d'une description plus macroscopique qui mixe l'expression temporelle en vertical avec le flot d'information en horizontal. En associant des propriétés à chaque élément par des attributs, ce modèle permet une extraction de propriétés de performances sans avoir à décrire le comportement détaillé (c'est à dire algorithmique) de chaque opération.

3.6.2 Opération, activité élémentaire

L'opération est l'unité comportementale la plus élémentaire, donc non-décomposable. Une activité non-décomposée est assimilée à une opération, tout en pouvant être très différente pour sa signification en modélisation. Une opération ou une activité élémentaire est définie vis-à-vis de son contexte englobant par une entrée en dessus et une sortie en dessous. Il ne s'agit pas ici d'une entrée ou d'une sortie de donnée pour le couplage comme pour le modèle structurel mais de l'indication de début d'évolution et de fin d'évolution. Les données utilisées par une opération ne sont pas représentées car le modèle est du type flot de contrôle et non pas flot de données.

Un temps d'évolution caractérise l'opération (attribut 'Time). S'ajoute un autre attribut ('Kind) permettant de différencier une opération d'attente qui ne nécessite pas de ressource d'exécution (passive) d'une opération d'exécution impliquant l'emploi d'une ressource (active).



-Figure 3.8- Représentation et spécification d'une opération.

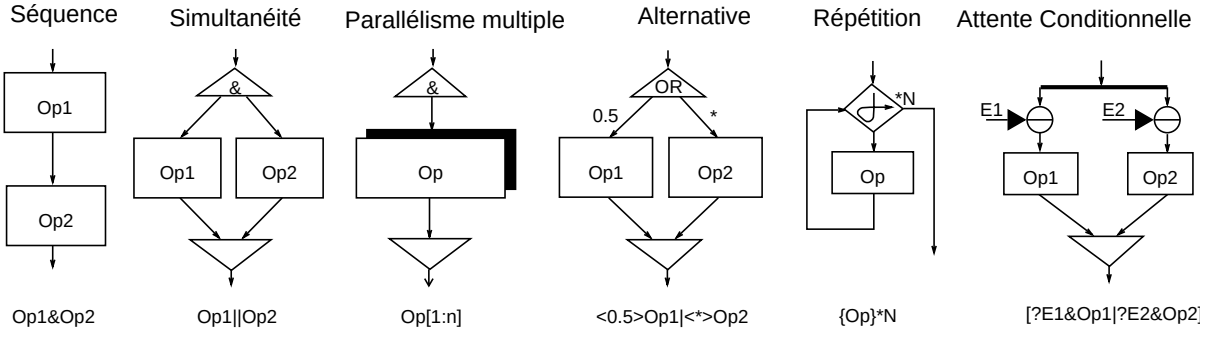
La durée de l'opération est un attribut qui peut se spécifier par toute expression mathématique utilisant des constantes, variables, paramètres réglables, probabilités. La particularité d'une opération est sa durée qui est finie et qui n'est pas conditionnelle à une circonstance externe. Cette durée représente l'utilisation de ressources pour sa réalisation.

Il faut donc bien différencier une opération d'un état utilisé dans un modèle du type diagramme à états finis. La durée dans un état d'un diagramme est souvent un temps d'attente et est dépendant de conditions qui indiquent l'achèvement de l'état.

NOTA: Il n'y a pas de différence entre une opération et une activité non raffinée (même symbole). Le symbole est choisi différent pour une activité raffinée ceci pour faciliter l'interprétation du modèle graphique.

3.6.3 Composition d'opérations et d'activités

L'expression d'un comportement résulte de l'association d'opérations et d'activités en respectant des règles de bonne structuration. Les opérateurs de composition usuels pour la spécification du contrôle sont: séquence, simultanéité ou parallélisme, alternative, répétition et attente conditionnelle. La figure 3.9 donne la représentation graphique et la notation syntaxique pour chaque type de construction.



-Figure 3.9- Représentation des constructions pour exprimer un déroulement.

Pour la séquence, la fin d'évolution de OP1 engendre le début d'évolution de OP2. Pour la simultanéité, toutes les opérations sont actives simultanément, l'activité résultante s'achève seulement lorsque toutes les opérations sont achevées. La notation d'instanciation multiple est utilisable pour le parallélisme.

Pour l'alternative, une seule opération devient active. Il faut donc y ajouter un critère de sélection qui peut être déterministe ou probabiliste. Pour cela, le pourcentage pour chaque opération ou une condition logique est associé au lien entrant. Une probabilité peut être utilisée, ainsi que toute expression mathématique comme pour la durée. Le total doit toujours être de 1; pour cela il y aura toujours une branche "else" ou notée aussi *.

La répétition exprime l'exécution conditionnelle (Repeat until, While) et inconditionnelle (For). La notation graphique est celle de SES/Workbench. La répétition doit être spécifiée. Elle peut être infinie (*) ou exprimée par une expression mathématique quelconque (comme pour les temps). Il peut s'agir d'une exécution N fois (*N) ou d'une exécution tous les N fois (correspond à une division (/N)). La répétition peut aussi être spécifiée par une condition de fin (until (=)).

Pour les applications qui nous concernent, il est indispensable de pouvoir modéliser l'activation d'une opération ou d'une activité parmi un ensemble et ceci lorsque la condition associée devient vraie. On dit alors qu'il s'agit d'une activation conditionnelle avec garde. Il s'agit d'une divergence OU mais dont la branche activée n'est pas prédéterminée comme pour la divergence OU mais est fonction de conditions sur des entrées. Pour éviter toute confusion d'interprétation, le symbole OR de la divergence n'est pas utilisé et est remplacé par une barre. Lorsque l'opération précédente est achevée, il y attente des conditions E1 et E2 (figure 3.9). La première condition présente engendre l'exécution de l'activité correspondante. Si les 2 conditions sont présentes simultanément, il y a sélection non-déterministe. La notation textuelle utilise des crochets pour exprimer l'exécution alternative et le symbole ? pour exprimer l'attente.

Le symbole «Exit» (carré noir) peut aussi être utilisée (voir plus loin) de manière à indiquer la fin de l'activité courante, ceci pour faciliter la représentation graphique. Ce symbole sera aussi utilisé pour sortir de l'activité englobante par un achèvement forcé.

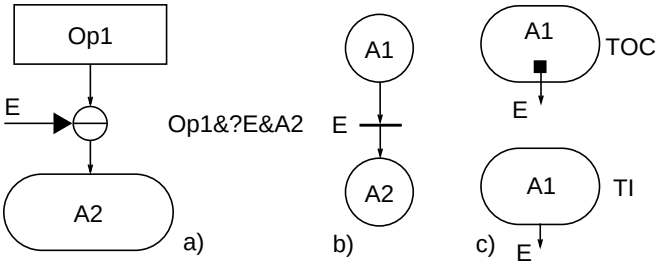
A noter que pour respecter des règles de bonne structuration, tout résultat de composition possède un point d'entrée temporel en dessus et surtout une seule sortie en bas. Ceci doit être le cas pour toute activité composée. Elle peut alors s'exprimer comme une opération possédant un attribut de temps représentant une vue macroscopique de l'activité.

3.6.4 Conditions d'évolution et actions

Un modèle doit être capable de représenter des dépendances temporelles autres que la fin d'une opération ou d'une activité. Pour cela, il faut exploiter les entrées de la fonction considérée du modèle structurel et disposer d'une technique pour exprimer des interactions internes à la fonction et de celle-ci avec son environnement.

-A- Activation conditionnelle

L'évolution d'une activité ou d'une opération doit pouvoir être conditionnelle à une entrée. La notation retenue est la suivante.



-Figure 3.10- Représentation d'une évolution conditionnelle.

Cette notation (a) se veut explicite pour bien montrer que lorsque Op1 s'achève, il y a attente de E pour poursuivre par A2. Op1 ne s'achève qu'après son temps d'exécution et non pas dès la présence de E (comportement non-préemptif). Dans la notation textuelle, le symbole? est utilisé pour décrire l'attente.

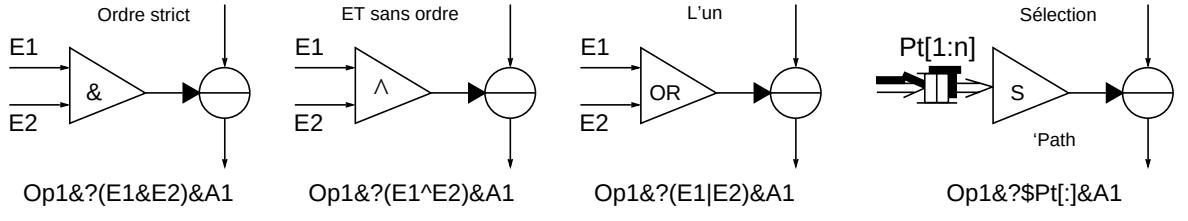
Ainsi, cette notation est différente de celle utilisée pour un diagramme d'états (b): l'état A1 est quitté dès la présence de E, l'état A1 ne peut pas être quitté sans E. Dans un SpecChart (c) [GAJSKI-94], une différence est introduite entre TOC (Transition On Completion) qui veut dire que l'état A1 est achevé mais n'est quitté que sur la condition E, de l'état TI (Transition Immediately) qui correspond au diagramme à états finis ou au Statechart: sortie immédiate de A1 sur la condition E.

-B- Construction d'une condition

Une condition d'évolution s'élabore à partir des entrées disponibles. Les opérateurs de composition qui concernent des occurrences sont:

- & pour le ET séquentiel qui impose une relation d'ordre strict d'apparition,
- ^ pour le ET logique sans ordre d'apparition,
- OR pour le OU logique,
- S pour la sélection.

La figure 3.11 donne la représentation graphique et la notation textuelle pour ces 4 compositions. Pt[:] veut dire le vecteur complet. Le symbole ? est utilisé pour exprimer une condition.



-Figure 3.11- Symboles pour la description d'une condition composée.

Rappelons que les 3 types de relation pour le modèle de structure sont:

- la synchronisation (événement),
- le transfert d'information,

- la donnée ou ressource partagée.

Pour ce dernier cas (lien double flèche dans le modèle comportemental), il faut différencier le cas de la donnée partagée du cas de la ressource commune. Pour la donnée partagée, il peut s'agir d'une lecture (consultation) ou d'une évolution souhaitée après une modification faite par une autre activité. Pour différencier les 2 cas, un attribut est associé au lien: ‘Read, ‘Change ou ‘Write.

Dans le cas d'une ressource partagée, il s'agit d'une demande d'allocation de la ressource. Lorsque la ressource est libre la condition d’évolution est vraie et dans ce cas la ressource est allouée. Pour plusieurs entrées de ressources, le OU conduit à l'obtention d'une ressource libre parmi l'ensemble des ressources possibles, le ET sans ordre correspond au cas de la nécessité de disposer de toutes les ressources, et le ET avec ordre impose l'ordre d'obtention des ressources.

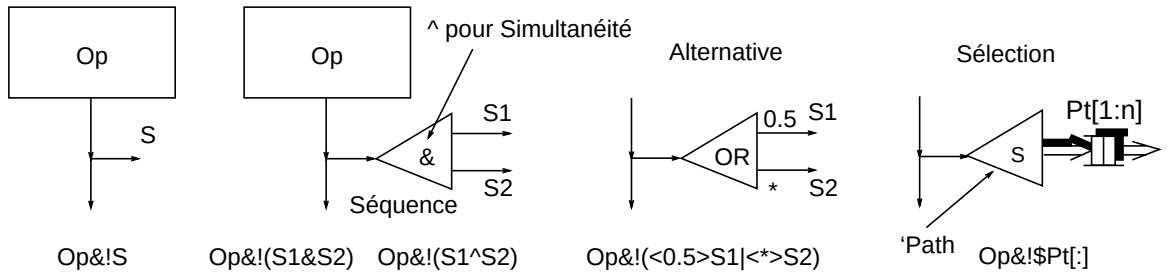
L’opérateur de sélection permet de spécifier l’entrée considérée. Celle-ci est indiquée par un attribut particulier appelé ‘Path associé à l’opérateur. La sélection peut être fixe (‘Path=Pt[5], ‘Path=me‘Index) ou dynamique en exploitant le contenu de messages ou de transactions (‘Path=mess‘Id) transitant par le sélecteur. Ceci est décrit plus loin.

Cette notation est exploitable en mixant les 3 types d'entrée: synchronisation, transfert d’information et donnée permanente. De plus, elle s'avère aussi utilisable pour des instanciations multiples d'éléments de relation, en particulier avec la sélection.

-C- Actions

Les actions concernent la génération d'information ou d'événements vers les sorties de la fonction ou vers d’autres activités internes. Pour un élément partagé, il s'agit de son exploitation: écriture, modification pour une donnée partagée, libération pour une ressource.

Pour des raisons de lisibilité et de structuration, les actions ne sont pas effectuées durant une opération mais après des opérations ou activités. La figure 3.12 donne la représentation graphique et la notation textuelle correspondante. Le symbole ! est utilisé pour exprimer la génération.



-Figure 3.12- Représentation pour la génération d’actions.

Une action vers des sorties peut concerner les 3 types de relation.

L'alternative conduit à ne générer qu'une seule sortie. Une règle doit être définie pour discerner la sortie concernée: déterministe, probabiliste. Pour cela, à chaque lien de sortie est affecté un pourcentage, l'un des liens possède le qualificatif "else", sachant que ce cas n’est pas obligatoire, ce qui veut dire qu’il n’y a pas toujours génération d’une sortie.

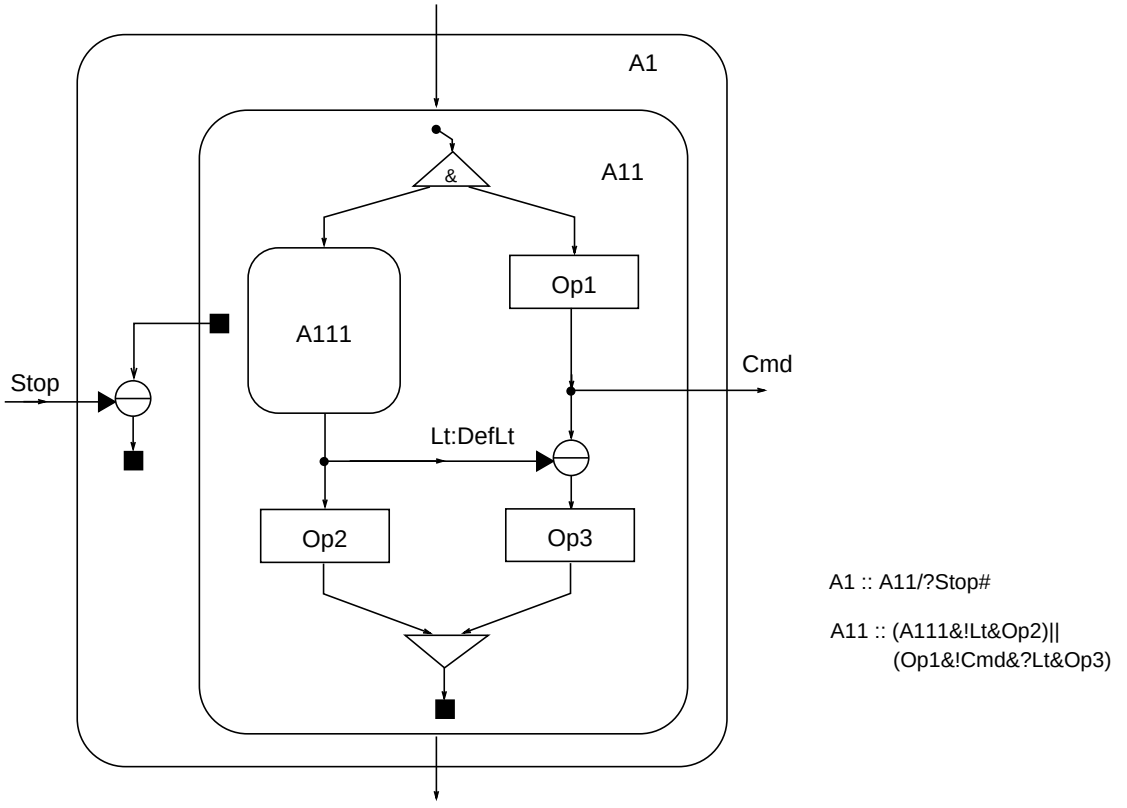
Le symbole Séquence (&) fixe l'ordre de génération avec exclusion. Le symbole Simultanéité (^) représente une génération commune qui ne s'achève que lorsque toutes les générations ont été faites en tenant compte des temps d'écriture.

Le symbole Sélection permet de représenter l'aiguillage vers une sortie particulière. La sortie concernée est désignée par l'attribut 'Path associé à l'opérateur. Il peut s'agir d'une désignation statique ('Path=me'Index) ou d'une désignation dynamique ('Path='Id) ce qui signifie l'aiguillage vers le destinataire désigné dans la transaction ou message. me'Index représente l'indice de l'activité courante.

3.6.5 Raffinement d'une activité et modèle réutilisable

Une activité peut se raffiner à l'aide des opérateurs décrits précédemment. Elle ne possède obligatoirement qu'une entrée et une sortie temporelle pour représenter le début et la fin de l'activité.

La figure 3.13 illustre la technique de raffinement. Plusieurs caractéristiques intéressantes y sont représentées.



-Figure 3.13- Raffinement d'une activité et achèvement forcé.

L'activité A11 inclut 2 branches concurrentes ou simultanées mais qui ne sont pas temporellement indépendantes. Le lien Lt sert à assurer une synchronisation (lien simple flèche) entre la fin de A111 et le début de Op3. Lt est un élément du type information car il possède un type; il se définit par des attributs similaires au port de communication. Cmd est un lien du type événement; il possède les attributs d'un événement. Ainsi une activité raffinée peut exploiter des liens de 3 types: information, événement, donnée partagée - chacun défini par les attributs décrits pour les 3 types de relation du modèle structurel.

L’activité A1 montre un raffinement particulier puisque le symbole "Exit" est utilisé (carré noir). A1 est simultanément en exécution de A11 et en attente de l’événement Stop. Lorsque celui-ci apparaît, il y a achèvement immédiat de l’activité A1 et donc de toutes les activités incluses (A11 en l’occurrence). De même lorsque A11 s’achève, le symbole Exit associé engendre l’achèvement de A1 et donc la suppression de l’attente sur Stop. Le symbole Exit permet donc une simplification de la représentation pour l’achèvement d’une activité. Mais il permet aussi lorsqu’il est utilisé comme source pour une activité d’engendrer un achèvement forcé ou inconditionnel de cette activité. Ceci est une forme de comportement préemptif. Les règles de construction permettent seulement à une activité de ne supprimer que des activités inclues (A11 est inclue dans A1). Le symbole "#" représente Exit dans la description textuelle et le symbole "/" représente la préemption.

Pour certaines descriptions, la même activité doit être utilisée à des instants différents ou/ et dans des activités ou fonctions différentes ou doit être instanciée plusieurs fois.

Pour faciliter la modélisation, il faut pouvoir disposer de la notion de modèle d’activité, tout comme la notion de modèle de fonction existe pour le modèle structurel. Aussi, une activité peut se définir par un modèle possédant des paramètres qui définissent les liens utilisables en entrée et en sortie pour le flot de données. Des valeurs génériques associées à l’activité permettent aussi de disposer d’activités génériques de manière à créer au moment de l’instanciation une activité avec des caractéristiques particulières.

3.6.6 Attributs et paramètres du modèle de comportement

Pour pouvoir extraire des résultats, il faut ajouter au modèle décrit précédemment tous les paramètres ou attributs nécessaires. Ces attributs sont associés sous forme textuelle au composant décrit.

On différencie une *valeur* d'une *grandeur paramètre*. Une *valeur* pour un attribut est déterminée par son expression textuelle: constante, résultat d'un calcul, valeur probabiliste.

Un *paramètre* est une grandeur qui peut varier dans un domaine de définition donné. Par exemple, la taille d'un message, la puissance d'un processeur, le facteur N d'instanciation multiple peuvent être des paramètres d'une modélisation qui servent à étudier son influence sur les résultats. La valeur d’un paramètre peut être modifiée entre 2 simulations ou peut même varier durant une simulation (incrémentation par exemple).

Un paramètre est utilisable dans toute expression. Durant une évaluation, un seul paramètre au maximum doit être considéré comme variable. Si plusieurs existent, une valeur doit être fixée pour tous les autres.

Les *attributs* servent à définir des valeurs aux constituants du modèle. Les attributs considérés ici ne concernent que le modèle comportemental car les attributs pour le modèle structurel ont été vus précédemment. Il s'agit ici des attributs de temps (‘Time), de taille de données ou d'information (‘Size), de sélection de chemin (‘Path), de condition (‘Cond), d'identification (‘Id).

-A- Attributs ‘Time, ‘Write, ‘Read

L’attribut ‘Time sert à définir le temps nécessaire pour une opération ou une activité et les attributs ‘Write et ‘Read pour une interaction avec l'environnement: lecture d'une donnée, dépôt d'un message, etc.

A1‘Time = 5 ms;
Op‘Time = exp (10) s;
T_Req‘Write = (K + 0,5 x T_Req‘Size) ms;

T_Req étant ici un port, l'attribut ‘Size du message ou de tout autre message est utilisable dans une expression temporelle. K est ici considéré comme un paramètre. "exp" est une fonction probabiliste selon une loi exponentielle.

Un temps est obligatoirement associé aux éléments de relation (voir les attributs du modèle structurel). Toutefois, il est aussi possible d'ajouter dans le modèle comportemental un temps ‘Write ou ‘Read aux liens d'accès pour surcharger celui de l’élément de relation.

-B- Attribut ‘Size

L’attribut ‘Size sert à définir la taille d'une donnée échangée ou d'un objet de transaction (un message émis par exemple).

T_Req‘Size = Uniform(5, 160) bits;
P_send‘Size = (L_Req‘Size + 64) bits;

Les unités à utiliser dépendent de l'application modélisée (bits pour un système de communication, mots pour une variable partagée, poids pour une pièce, etc). Ainsi, les unités sont définies par l’utilisateur (les plus courantes étant définies par défaut). Les expressions doivent rester cohérentes pour les unités.

Lorsque la taille concerne un objet de transaction, l'attribut est associé au lien, sinon elle est associée à l'élément de relation et est dans ce cas utilisé par défaut.

-C- Attribut ‘Id

Cet attribut est défini pour identifier une information ou un objet dans le cas d'un transfert. Sa valeur est un nombre entier qui peut être:

- une suite croissante,
- un tirage aléatoire dans un intervalle,
- l'indice de l'élément englobant (me‘Index).

me‘Index veut dire l’indice de l’élément courant dans le cas d’une instance multiple.

-D- Attribut ‘Path

L’attribut ‘Path sert à définir le chemin qui sera choisi dans le cas d'une sélection. Il est possible d’utiliser une désignation fixe ou d’exploiter l'identité (‘Id) du contenu dans le cas d'une information. Ce dernier cas permet de modéliser un comportement conditionnel au flot d’information.

La méthode d’aiguillage par la sélection est différente de la divergence OU. Pour la divergence, une expression fixe la probabilité pour chaque branche. La somme de toutes les branches doit donner 1 au maximum. Ainsi, l'une des branches peut prendre la valeur "else". Si aucune valeur n'est fixée, il s'agit d'une sélection équiprobable, donc sélection à tour de rôle.

-E- Attribut ‘Cond

Cet attribut est utilisé pour l’opérateur Répétition. Il peut s'agir:

- d'une exécution N fois (*N),
- d'une exécution une fois tous les N (/N),
- d'une exécution jusqu'à une condition (=expression).

Une expression mathématique quelconque est utilisée pour spécifier N ou la condition. La valeur N est calculée à l'entrée dans le symbole. Par contre la condition est évaluée à chaque sortie possible.

Pour l'exécution N fois, si N n'est pas spécifiée, il s'agit d'une exécution infinie.

-F- Attributs sur les liens

Des attributs sur les liens sont parfois nécessaires pour fixer d'une manière plus fine le comportement souhaité. Ces attributs sont associés aux informations en transit pour les attributs Id et Size, ou sont associés aux entrées et sorties de la fonction ou de l'activité décrite. Les attributs 'Read et 'Write sont aussi utilisables.

-G- Attributs utilisateur

Tout autre attribut peut être ajouté par l'utilisateur sur son modèle, en particulier lorsqu'il s'agit de spécifier un contenu de variable partagée ou d'un message. Ces attributs se traduisent alors par des champs de données supplémentaires exploitables par toute fonction, activité ou opération pouvant y accéder.

-H- Paramètres génériques

Toute valeur d'attribut peut se définir par une expression mathématique. Pour étudier l'influence d'un ou plusieurs paramètres, ceux-ci sont utilisés dans l'expression de calcul de l'attribut et déclaré comme paramètre générique pour le composant.

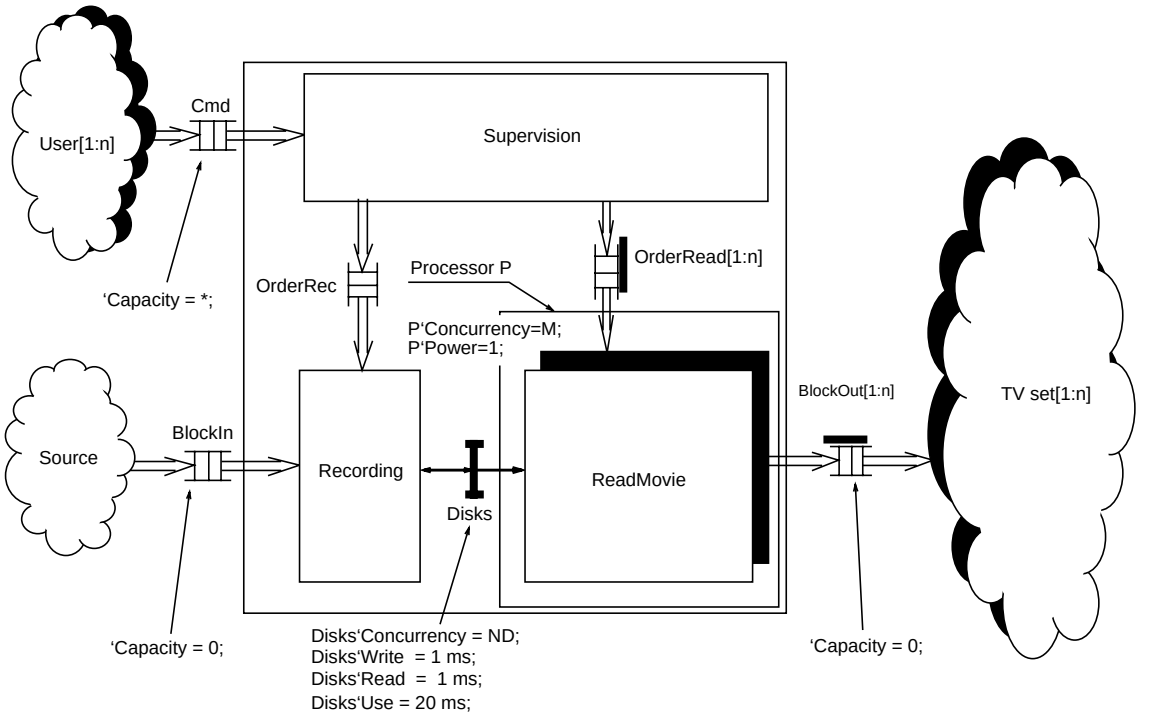
3.7 EXEMPLE DE MODELISATION

L'exemple que nous décrivons dans ce paragraphe permet de voir l'application et l'intérêt du modèle de performance de MCSE sur un système simple mais intéressant. Il s'agit d'une version très simplifiée du serveur vidéo temps-réel extrait du problème global posé par le CCETT de Rennes et présenté dans le chapitre 7.

Le rôle d'un tel serveur est de transmettre en temps-réel à chaque utilisateur sur son téléviseur le film vidéo qu'il sélectionne. Le système est imaginé capable de servir N utilisateurs simultanément. L'objectif de la modélisation de performances est de permettre de dimensionner l'architecture de la solution: nombre de disques nécessaires, nombre de processeurs ou puissance du processeur pour un nombre N d'accès simultanés, taille mémoire interne nécessaire, etc. L'exemple et les résultats obtenus ont été présentés dans [CALVEZ-95a] [CALVEZ-96d].

3.7.1 Modèle de performance pour le serveur

Nous ne détaillons pas ici la spécification du système. Le lecteur trouvera ces informations dans le chapitre 7. Nous donnons directement la solution fonctionnelle permettant une estimation de performances. La figure 3.14 représente la description fonctionnelle retenue pour le serveur et les 3 entités de son environnement. Il s'agit d'un modèle générique et très abstrait car: les disques contenant les films sont modélisés par une variable partagée, les lignes de transmission et les interfaces nécessaires sont modélisées par des ports de communication.

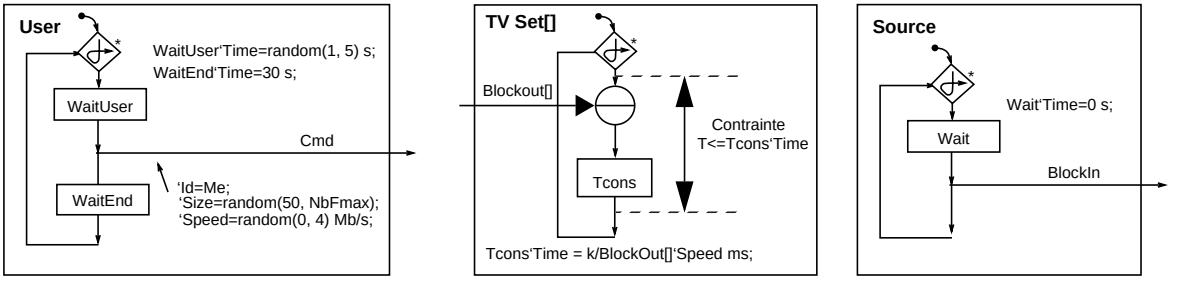


-Figure 3.14- Solution fonctionnelle pour l'application Serveur vidéo.

Un utilisateur (User[i]) sélectionne un film en produisant un ordre Cmd. La fonction Supervision vérifie la demande et si elle correcte et que l'usager est autorisé, une demande de lecture OrderRead[i] ou d'enregistrement OrderRec est transmis. La fonction Recording se charge de l'enregistrement d'un film sur les disques du serveur à partir de l'entité Source. Les films sont enregistrés sous la forme d'un ensemble de blocs successifs, chaque bloc étant de taille fixe. Une fonction ReadMovie est associée à chaque usager. Une telle fonction débute par la lecture de A (figure 3.16-c) blocs sur disques puis assure la transmission des blocs un par un vers le téléviseur correspondant tout en poursuivant la lecture des blocs suivants. Le paramètre A est à déterminer pour éviter toute rupture de séquence. La période de transmission des blocs est définie par le débit pour le film (déterminé par la qualité et le codage en compression).

Sur la figure 3.14, on voit que toutes les fonctions ReadMovie sont considérées implantées sur un même processeur P. Ceci va permettre d'étudier l'influence de cette ressource comme contrainte d'exécution. Son degré de concurrence 'Concurrency est M et sa puissance 'Power est supposée de 1 par défaut. L'ensemble des disques est modélisé par une variable partagée possédant les attributs 'Concurrency décrivant le nombre de disques à accès simultanés, 'Read et 'Write comme temps de lecture et d'écriture d'un bloc au contrôleur d'un disque, 'Use comme temps d'utilisation du disque.

Le comportement de chacune des 3 entités de l'environnement est décrit par la figure 3.15.



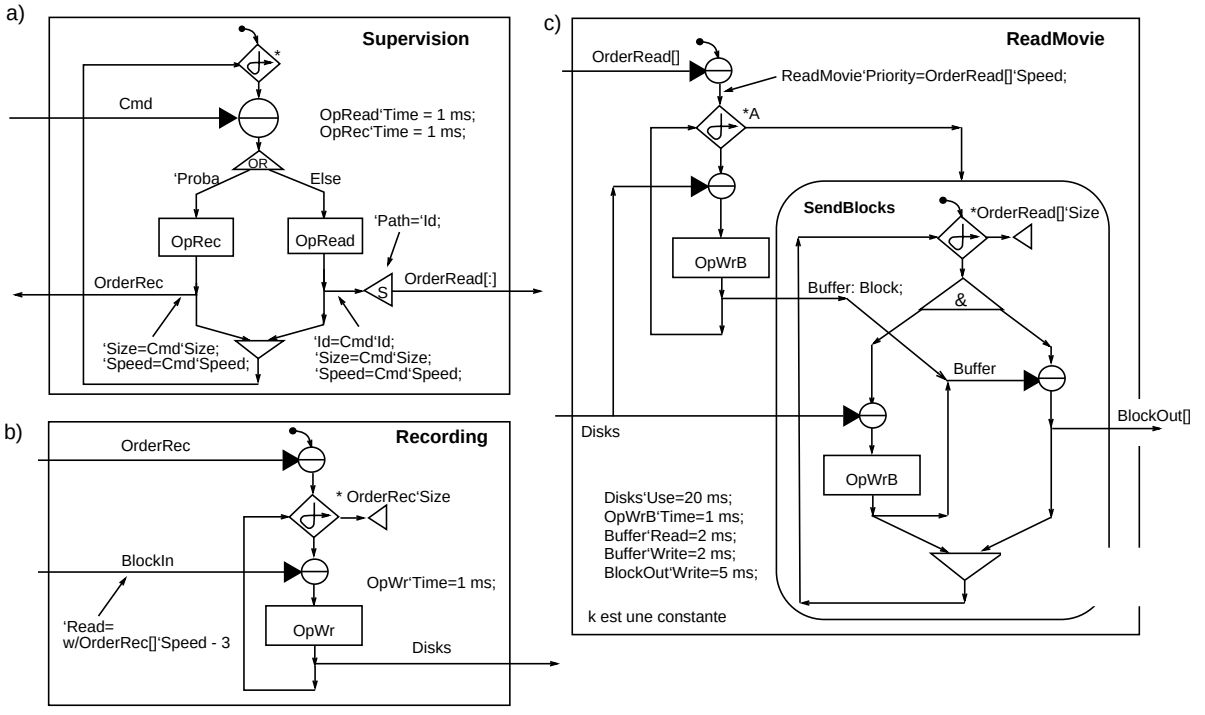
-Figure 3.15- Comportement des entités de l'environnement.

L'entité User génère après un temps aléatoire entre 1 et 5 secondes un ordre Cmd comprenant les attributs: 'Id qui est l'identité de l'usager, 'Size représentant la taille en nombre de blocs de 32 ko, 'Speed qui définit la vitesse de transmission en lecture ou en enregistrement (valeur aléatoire ici). Cette entité permet ainsi de définir complètement la charge du système (workload).

Un téléviseur est modélisé comme un récepteur attendant les blocs successifs sur sa ligne de réception modélisé par BlockOut[] séparés apr une attente égale à la période entre 2 blocs consécutifs. Ce temps est défini par le débit contenu comme attribut du message. La rupture de séquence en lecture se détecte par le non-respect de la contrainte de temps T qui correspond à la période.

La source est modélisée comme un générateur idéal de blocs. Les ports BlockOut[:] et BlockIn sont du type rendez-vous ('Capacity=0).

Le comportement de chacune des 3 fonctions de la solution interne fonctionnelle est décrit par la figure 3.16.



-Figure 3.16- Comportement des fonctions internes du serveur.

La fonction Supervision attend les demandes des usagers par l'entrée Cmd. Si la demande est correcte, un tirage aléatoire (Proba) permet d'orienter l'ordre vers la fonction d'enregistrement Recording par OrderRec ou vers une fonction de lecture ReadMovie associée au téléviseur du même indice par OrderRead[‘Id]. Les attributs ‘Size et ‘Speed de Cmd sont transmis par OrderRec ou OrderRead[].

La fonction Recording assure la lecture auprès de la source de Size blocs et leur copie sur les disques. Le temps de lecture est défini par la vitesse de transmission qui dépend de l'attribut ‘Speed reçu.

La fonction ReadMovie assure la lecture des blocs successifs sur les disques et leur transfert vers le téléviseur. Pour éviter un défaut de séquence au début par suite de la latence des disques, cette fonction se charge de lire A blocs en anticipation et les stocke dans Buffer. Ensuite, chaque bloc de Buffer est transmis et un nouveau bloc est lu sur disques. Buffer est une fifo de capacité supposée infinie. La priorité d'exécution de ReadMovie dans le cas d'un processeur restreignant le degré de concurrence est définie en rapport avec la vitesse de transmisssion défini par l'attribut ‘Speed reçu.

3.7.2 Description textuelle du serveur vidéo

Le modèle graphique ci-dessus est transcriptible en une notation textuelle. La description suivante donne l'écriture textuelle pour l'ensemble de l'application Serveur Vidéo. Ce texte sert de point d'entrée au générateur de code VHDL présenté dans le chapitre 6.

```
<RelationTypes>
 DefCmd;
 DefBlockIn;
 DefBlockOut;
 DefOrderRec;
 DefOrderRead;
 DefSequences;
 Block;
<EndRelationTypes>
<GenericParameters>
  n,INTEGER,"nonbre de canaux haut debit",10;
  nd,INTEGER,"nombre de disques",10;
  A,INTEGER,"taille des fifo",4;
<EndGenericParameters>
<FunctionalStructure>
<Component> ServeurVideoSimple Generic n,nd,A
();
 <Structure> StructureServeurVideoSimple;
  <Port> Cmd : DefCmd;
  <Port> BlockIn : DefBlockIn;
      <Attributes>
        `Capacity=0;
      <EndAttributes>
<Port> [1:n] BlockOut : DefBlockOut;
      <Attributes>
        `Capacity=0;
      <EndAttributes>
<Component> [1:n] User
   (Out Mess Cmd : DefCmd;);
   <Attributes>
    Cmd`Id = me;
    Cmd`Size = UniformInt(50,120);
    Cmd`Speed = UniformInt(100,4000);
    WaitUser`Time = UniformTime(1000,5000,ms);
```

```
     WaitEnd`Time = 30 sec;
    <EndAttributes>
    <Behavior> BehaviorUser;
      User::{WaitUser&!Cmd&WaitEnd}*;
    <EndBehavior>
<EndComponent>
<Component> Source
    (Out Mess BlockIn:DefBlockIn;);
    <Attributes>
      Wait`Time=NullTime;
    <EndAttributes>
    <Behavior> BehaviorSource;
      Source::{Wait&!BlockIn}*;
    <EndBehavior>
  <EndComponent>
<Component> [1:n] TVset
    (In Mess BlockOut[]:DefBlockOut;);
    <Attributes>
      Tcons`Time=(262136/BlockOut[]`Speed) * 1 ms;
    <EndAttributes>
    <Behavior> BehaviorTVSet;
       TVset::{?BlockOut[]&Tcons}*;
    <EndBehavior>
  <EndComponent>
<Component> ServeurVideo
    (In Mess Cmd:DefCmd;
     In Mess BlockIn:DefBlockIn;
     Out Mess BlockOut[1:n] : DefBlockOut;);
    <Structure> StructureServeurvideo;
     <Var> Disks : DefSequences;
       <Attributes>
         `Concurrency=nd;
         `Use=20 ms;
         `Write=1 ms;
         `Read=1 ms;
       <EndAttributes>
     <Port> OrderRec : DefOrderRec;
     <Port> [1:n] OrderRead : DefOrderRead;
      <Component> Supervision
           (In Mess Cmd:DefCmd;
           Out Mess OrderRec:DefOrderRec;
            Out Mess OrderRead[:]:DefOrderRead;);
       <Attributes>
           OrderRec`Size=Cmd`Size;
            OrderRec`Speed=Cmd`Speed;
           OrderRead[:]`Path=`Id;
           OrderRead[]`Id=Cmd`Id;
            OrderRead[]`Size=Cmd`Size;
            OrderRead[]`Speed=Cmd`Speed;
            OpRec`Time=1 ms;
            OpRead`Time=1 ms;
      <EndAttributes>
      <Behavior> BehaviorSupervision;
           Supervision::{?Cmd&(<0.00001>(OpRec&!$OrderRec)
                          <*>(OpRead&!$OrderRead[:]))}*;
       <EndBehavior>
      <EndComponent>
      <Component> Recording
           (In Mess BlockIn:DefBlockIn;
        In Mess OrderRec:DefOrderRec;
             InOut Var Disks:DefSequences;);
      <Attributes>
```

```
        BlocIn`Read=(262136/OrderRec[]`Speed) * 1 ms;
        OpWr`Time=1 ms;
       <EndAttributes>
       <Behavior> BehaviorRecording;
          Recording::{?OrderRec&({?BlockIn&OpWr&!Disks}*OrderRec`Size)}*;
        <EndBehavior>
       <EndComponent>
       <Component> [1:n] ReadMovie
       (In Mess OrderRead[]:DefOrderRead;
         In Var Disks:DefSequences;
         Out Mess BlockOut[]:DefBlockOut;);
         <Attributes>
            BlockOut`Write=5 ms;
            Buf`Read=2 ms;
            Buf`Write=2 ms;
            OpWrB`Time=1 ms;
         <EndAttributes>
         <Behavior> BehaviorReadMovie;
           <InfoLink> Buf:Block;
           <Attributes>
             `Capacity = A;
               <EndAttributes>
            <EndInfoLink>
ReadMovie::{?OrderRead[]&({?Disks&OpWrB&!Buf}*A)&SendBlocks}*;
       SendBlocks::{(?Disks&OpWrB&!Buf) ||
       SendBlock{`Priority=me+1;}:(?Buf&!BlockOut[])}*OrderRead[]`Size;
         <EndBehavior>
    <EndComponent>
 <EndStructure>
 <EndComponent>
<EndStructure>
<EndComponent>
<EndFunctionalStructure>
```

3.8 CONCLUSION

Ce chapitre a permis de formaliser et d'illustrer le modèle de performance de MCSE. Pour cela, les caractéristiques d'un tel modèle ont tout d'abord été présentées, ainsi qu'une analyse des concepts essentiels avec leurs propriétés. Le modèle de performance a ensuite été décrit en présentant successivement les 2 vues que sont la vue structurelle et la vue comportementale.

Le modèle retenu satisfait les critères présentés dans le paragraphe 3.2. Les critères de lisibilité et de compréhensibilité sont satisfaits grâce à l'emploi d'une notation essentiellement graphique. L'efficacité de déduction provient de l'intégration complète du modèle de performance à la méthodologie de conception système MCSE. La continuité du modèle par transformation d'un comportement en un modèle structurel facilite la transition d'une phase de conception à la suivante et assure la traçabilité. L'efficacité d'évaluation résulte du modèle (généricité et réutilisabilité) et de la technique de simulation associée. La technique de simulation associée au modèle repose sur la transcription de la représentation textuelle du modèle en un langage exécutable ou simulable. Comme nous le verrons dans le chapitre suivant, nous avons retenu et expérimenté l'utilisation du couple VHDL/Simulateur VHDL.

Basé sur l'association de la vue structurelle et de la vue comportementale, le modèle de performance de MCSE est:

- *évolutif*. Le concept d'attribut pour définir des propriétés aux constituants permet d'enrichir le modèle à volonté.

- *macroscopique*. Le système n'a pas besoin d'être entièrement détaillé. Pour une évaluation des performances, les opérations sont caractérisées par un temps d'exécution et non par un algorithme particulier de traitement. Une évaluation des performances peut donc se faire à un niveau quelconque d'abstraction et éventuellement très tôt dans le cycle de développement.
- *non-interprété*. Le terme non-interprété signifie que seul le comportement ou les dépendances temporelles entre les sorties et les entrées sont observées. Les valeurs en entrée et celles des données internes ne sont pas prises en compte. Les entrées et les données internes influencent le comportement du système uniquement par l'intermédiaire d'attributs. Par exemple, les attributs 'Size (taille) et 'Id (destinataire) d'un message remplacent le contenu du message.
- *générique* et *paramétrable*. L'utilisation de paramètres génériques associés aux attributs des éléments du modèle de performance permet de parcourir un espace assez vaste des solutions possibles d'un partitionnement sans nécessiter de mise à jour du modèle de performance.

Un modèle seul n'est pas suffisant. Un modèle doit toujours être accompagné d'une démarche favorisant l'élaboration de solutions. Rappelons que la modélisation de performances a pour objectif de spécifier quantitativement les propriétés temporelles d'une solution. Une solution est une représentation de la vision du concepteur tout au long du processus de développement. Elle évolue depuis la spécification jusqu'au système réalisé. La démarche de modélisation des performances d'un système doit permettre d'élaborer une représentation des propriétés du système à différents stades du développement. Ainsi plusieurs modèles de performances sont développés, tout d'abord un modèle macroscopique pour représenter les spécifications de performances, ensuite des modèles de plus en plus détaillés faisant apparaître la structure interne retenue pour le système comme résultat d'une phase de conception. Cette démarche a été expliquée dans le chapitre 2.

Notre modèle est très utile pour la problématique du CoDesign qui correspond à l'étape de définition de la réalisation de la méthodologie MCSE. Pour une solution faisant intervenir du matériel et du logiciel, lorsque plusieurs fonctions s'exécutent sur un même processeur séquentiel, notre modèle permet de représenter et de simuler l'exécution d'un ensemble de fonctions sur un processeur. Ainsi, il est possible d'observer l'influence de la politique d'ordonnancement (attribut 'Policy) et de la puissance du processeur (attribut 'Power). Notre modèle autorise aussi la co-simulation macroscopique et non-interprétée des fonctions logicielles et des composants matériels, ce qui permet alors au concepteur de trouver par une démarche itérative le partitionnement et l'allocation optimale vis à vis des critères de coûts et de performances qui lui sont imposés. En remplaçant ensuite, les opérations élémentaires par du code interprété, le concepteur peut également faire une vérification fonctionnelle de sa solution.

Le modèle de performance présenté ici ne limite pas son champ d'application à l'analyse des systèmes électroniques. En donnant une signification différente à ces concepts, il peut aussi être utilisé par exemple pour l'analyse de tout type d'architecture, des réseaux de communication, la gestion de production et des stocks, voir même la gestion des ressources humaines.