# Chapitre 1 - Langages de description d'architectures matérielles hybrides

*«Hélas, tout a déjà été dit et on arrive trop tard !» (La Bruyère)*

## Résumé

La méthode MEDEVER nécessite la description de l'architecture matérielle utilisée pour l'exécution d'applications réparties et parallèles. Cette architecture matérielle est souvent qualifiée d'hybride, car elle est composée de machines mono et multi-processeurs connectées sur un même réseau physique. Nous présentons donc au début de ce chapitre les différents types d'architectures matérielles et leurs classifications, avant de donner notre définition d'une architecture matérielle hybride.

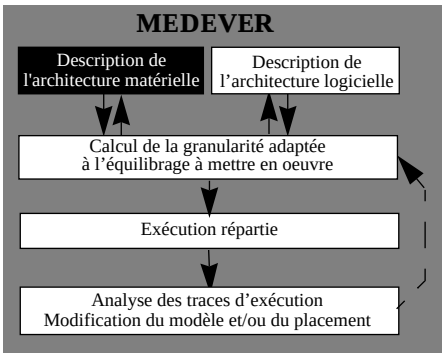
Puis, nous présentons un état de l'art de la description d'architectures matérielles hybrides dans divers types de langages qui sont susceptibles d'être intégrés dans notre méthode MEDEVER. Les critères définis pour cette étude sont la convivialité, la portabilité, l'extensibilité, la gestion de la granularité, la réutilisabilité et la prise en compte de la dynamicité de l'architecture. Nous concluons sur le fait qu'aucun des langages à notre connaissance ne répond complètement à nos besoins.

La dernière partie de ce chapitre propose un langage de description d'architectures matérielles hybrides, nommé HADEL. HADEL gère deux niveaux de hiérarchie : le niveau Macro et le niveau Micro. Le niveau Macro décrit des architectures de machines à Macro, à savoir des stations de travail et des équipements réseaux qui les relient. Le niveau Micro décrit des machines multi-processeurs et les éléments qui la constituent (processeurs, mémoire, ports d'entrées-sorties, etc.). La description d'une architecture est alors réalisée avec notre langage soit via un formalisme graphique ou directement via un fichier texte.

## Apports scientifiques

Nous avons défini un langage adapté à la description statique et à la gestion dynamique **d'architectures matérielles hybrides**. Les architectures une fois définies sont vérifiées, puis stockées dans une bibliothèque. Le choix d'une architecture au sein de cette bibliothèque est mis en oeuvre soit par interrogation de la base via un langage simple, soit directement au travers de fonctions d'accès.

## Plan




**Description d'une architecture matérielle**

| |
|---|
| Classification des architectures matérielles des machines ou réseaux de machines multi-processeurs |
| Définition d'une architecture matérielle Hybride |
| Etat de l'art sur les langage de description d'architectures matérielles<br>- dans les langages de programmation parallèles<br>- dans les langages de configuration<br>- dans les logiciels de simulation<br>- dans les logiciels de répartition de charge<br>- dans les logiciels d'administration de systèmes et de réseaux |
| ☞ HADEL, notre langage de description d'architectures matérielles hybrides |

## Mots clefs

Architectures matérielles, architectures hybrides, langage de description, machines parallèles, langage HADEL, bibliothèques.

L'augmentation rapide des capacités de traitement des ordinateurs et la continuelle baisse des prix des équipements ont permis l'émergence de plate-formes matérielles adaptées à des applications réparties. Les architectures matérielles utilisant simultanément plusieurs dizaines de processeurs sont aujourd'hui monnaie courante. Mais, un réseau de stations de travail connectées sur un réseau local ou sur un réseau haut débit métropolitain n'offrent pas les mêmes caractéristiques et les mêmes interfaces de programmation qu'un super-calculateur. Or, ces deux architectures matérielles offrent aujourd'hui le même ordre de grandeur en terme de puissance théorique. C'est pourquoi, nous étudions l'apport de ces architectures pour l'exécution d'applications réparties et parallèles. Nous concentrons néanmoins notre étude sur un type récent d'architecture : les architectures matérielles hybrides locales (ie. sur un réseau local). Une architecture matérielle hybride est composée de machines mono et multi-processeurs connectées sur un même réseau physique.

La mise au point de l'architecture d'une application parallèle et/ou répartie est plus ou moins dépendante de l'architecture matérielle des machines de calcul et du réseau informatique sous jacent. On cherche alors à concilier les besoins en bande passante, la recherche de temps d'exécution minimaux et les problèmes d'hétérogénéité logicielle et matérielle (protocoles de communication et systèmes d'exploitation différents). La connaissance de l'architecture matérielle utilisée contribue alors à faire des choix de programmation ou de placement des éléments logiciels qui augmentent les performances. Ceci est d'autant plus vrai lorsque les architectures matérielles utilisées sont propriétaires et spécifiques. Le meilleur exemple est alors celui des architectures de machines dites «massivement parallèles», qui sont liées à des classes d'applications spécifiques, auxquelles elles s'adaptent au plus près (il en résulte d'ailleurs une difficulté de programmation plus grande).

A ces contraintes liées à la programmation d'applications réparties et/ou parallèles s'ajoute une volonté d'utilisation optimale et dynamique des machines disponibles. C'est pourquoi, la connaissance des éléments constitutifs de la plate-forme matérielle est primordiale. Ces informations quoi que souvent techniques, sont à la base des solutions de tolérance aux fautes et d'administration de parcs hétérogènes. Chaque machine est interrogée périodiquement sur son état ou configurée pour alerter, elle même, un gestionnaire dès qu'elle vérifie une condition prédéfinie à l'avance (dépassement d'un seuil de charge ou d'un nombre d'utilisateurs connectés). Si l'architecture matérielle n'est pas disponible réellement, la simulation du comportement des éléments qui la constitue donne souvent un ordre de grandeur a priori des débits et de la puissance globale de traitement de l'information. Ces mesures issues de simulation sont alors utilisées pour le dimensionnement et la mise au point de l'architecture matérielle à mettre en oeuvre.

De notre point de vue, la description d'une architecture matérielle s'étend donc sur les trois domaines distincts énoncés ci-dessus, à savoir :

- la description de l'architecture matérielle en vue du placement et du déploiement d'applications réparties et/ou parallèles
- la gestion statique et dynamique d'une architecture de machines hétérogènes
- la simulation d'architectures matérielles.

Le langage de description doit donc intégrer ces trois axes, tout en restant suffisamment ouvert. En effet, il doit être intégrable dans des outils communs de développement et d'exécution répartie sur des architectures hybrides.

La première section rappelle les différentes classifications existantes d'architectures matérielles de machines et conclue par la définition d'un réseau hybride de machines. La section 2 présente un état de l'art sur les langages et les outils de description d'architecture matérielle en respectant les trois axes précédemment définis, à travers les langages de programmation parallèle, les langages de configuration, les logiciels de simulation, les logiciels de répartition de charge et les logiciels d'administration de systèmes et de

réseaux. La section 3 décrit notre langage HADEL de description d'architecture matérielle hybride et conclue sur des perspectives d'extensions du langage.

# 1. Classifications des architectures matérielles multi-processeurs

Il n'existe pas, à l'heure actuelle, une classification unique et universelle des architectures matérielles de machines mono ou multi-processeurs connectées (ou non) par un réseau informatique.

Les types de machines actuelles étant nombreux, des travaux ont été menés en vue d'en établir une classification. Cette classification sert de base à la création de langages de description d'architectures matérielles et est nécessaire à la structuration de bibliothèques prédéfinies d'architectures ou de types d'architectures de machines [Damm 96].

Nous présentons dans la suite de cette section la classification de Flynn, puis d'autres classifications alternatives prenant en compte les architectures hybrides que nous définirons. Ces classifications reposent généralement sur les composantes suivantes :

- *la mémoire centrale* qui stocke le programme et les données durant l'exécution ;
- *l'unité de traitement* chargée de faire les calculs ;
- *l'unité de contrôle* qui cadence l'enchaînement des calculs ;
- *l'unité de stockage* des données persistantes.

## 1.1. La classification de Flynn

La classification la plus ancienne, et la plus utilisée, est fondée sur l'analyse des flots de contrôle et des flots de données [Flynn 72]. Quatre types distincts de modèles de machines sont alors répertoriés :

1) le modèle **SISD (Single Instruction, Single Data)** correspond à la machine séquentielle de Von Neumann, dans lequel une machine est constitué d'une unité de traitement manipulant les séries de donnée l'une après l'autre.
2) le modèle **SIMD (Single Instruction, Multiple Data)** est conceptuellement proche du modèle SISD. Une unité de contrôle unique synchronise des unités de traitements multiples et homogènes qui effectuent la même opération sur des données différentes.
3) le modèle **MISD (Multiple Instruction, Single Data)** découpe les unités de traitement et de contrôle en étages. Chaque opération, à un étage N, est unique est effectuée sur des données différentes. Ce modèle dit «en pipeline» est mis en oeuvre sur des machines RISC, VLIW (Very Long Instruction Word) et de certains super-calculateurs vectoriels.
4) le modèle **MIMD (Multiple Instruction, Multiple Data)** correspond aux machines multi-processeurs qui communiquent par passage de messages. Des unités de traitement hétérogènes gérées par des unités de contrôles distinctes exécutent traitent des flots de données hétérogènes (ou non) de manière asynchrone.

La classification de Flynn ne tient pas compte des architectures parallèles à mémoire répartie qui n'existait pas à l'époque [Sansonnet & al. 91]. C'est pourquoi des extensions à cette classification ont vu le jour.

[Ibbett 92], par exemple, ajoute à la classification de Flynn deux axes :

- le type d'arithmétique : SE pour une exécution série et PE pour une exécution parallèle ;
- le nombre d'espaces d'adressage : SA pour un espace d'adressage simple et MA

pour un espace d'adressage multiple.

Depuis, d'autres classifications axées sur le nombre de processeurs, l'accès aux disques et le type de mémoire utilisée ont été proposées.

## 1.2. Vers des classifications alternatives

[Stonebraker 86] propose de découper les architectures multi-processeurs en quatre classes :

- **tout est partagé *(Shared-Everything)*** : chaque processeur peut accéder à tous les disques et toute la mémoire.
- **rien n'est partagé *(Shared-Nothing)*** : chaque processeur possède ses disques et sa mémoire propre.
- **les disques sont partagés *(Shared-Disks)*** : la mémoire est privée, mais les disques sont partagés entre les processeurs.
- **Hybrides *(Hybrid)*** : les noeuds d'un système à disques partagés sont des systèmes «tout est partagé». Les systèmes hybrides offrent donc un partage de disque logique ou physique.

Cette classification a été étudiée [Dewitt & al. 92] et [Bergsten & al. 93] pour l'accès à des bases de données parallèles. Des critères de comparaison ont été établis pour évaluer le coût, les performances et l'extensibilité de chacune des architectures proposées en fonction d'applications cibles.

Mais de nouvelles classes d'architectures de machines (notamment à mémoire partagée et répartie) sont apparues. Des classifications plus récentes [Chrichlow 88], [ChassinD.K. 95] et [Zomaya 96] tentent de les prendre en compte :

- **les architectures UMA *(Uniform Memory Access)*** : la mémoire physique est partagée et le temps d'accès aux données partagées est uniforme. La plupart du temps ces architectures sont organisées autour d'un bus commun ou de réseaux multi-étages reliant entre eux tous les éléments matériels de la machine. L'accès aux données est donc rapide, mais le nombre de processeurs est du coup limité par le bus unique. Dans cette catégorie, on trouve les Multimax 500 (Encore), la gamme Symmetry (Sequent) et les PC de type SMP (*Symetric MultiProcessing)* composé de $2^n$ processeurs Pentium et Pentium Pro ($n < 7$).
- **les architectures NUMA (*Non Uniform Memory Access*)** : elles se caractérisent par des temps d'accès non uniformes aux données. Les architectures NUMA répondent aux problèmes d'extensibilité des systèmes UMA, en relâchant la contrainte d'un coût uniforme d'accès à la mémoire. Les temps d'accès varient selon la localisation du bloc mémoire référencé. En effet, chaque processeur possède sa propre mémoire et accède aux autres modules mémoires grâce à un réseau d'interconnexion. L'ensemble de ces mémoires locales, distribuées entre les processeurs, forment un espace d'adressage unique. On classe les architectures NUMA en deux catégories :
  - **NCC-NUMA (*Non Cache-Coherent NUMA*)** : seul l'accès direct à des données distantes est géré par le matériel. La gestion de la cohérence des caches relève de la responsabilité du compilateur ou du programmeur.
  - **CC-NUMA (*Cache-Coherent NUMA*)** : ces architectures disposent d'une mise en oeuvre matérielle permettant d'assurer l'accès direct aux blocs distants et la cohérence des caches.

  Le Cray T3D et la BBN TC2000 sont des machines de ce type. La *DASH* de Stanford et la *Alewife* du MIT sont de type CC-NUMA.

- **les architectures COMA *(Cache Only Memory Access)*** : ce sont des architectures NUMA dans lesquelles les mémoires sont utilisées comme des caches de grande taille. La gestion de la cohérence des mémoires est assurée par le matériel [Hagersten & al. 92]. Les adresses logiques ne sont pas associées statiquement à des adresses physiques. Les données ne sont pas associées à un site particulier et migrent vers le site qui les utilise. La gestion du déplacement des données est transparente aux programmeurs car l'adressage est entièrement réalisé par le système d'exploitation. La plupart des machines COMA actuelles, telle que la KSR1, ont été mises au point par la société *Kendall Square Research*. La tendance actuelle est de coupler des architectures de type COMA et des techniques de cohérence de cache répartie, on parle alors d'architectures CC-COMA.
- **les architectures NORMA *(NO Remote Memory Access)*** : elles sont constituées de processeurs qui disposent de mémoire propre, appelée mémoire locale et d'un disque local ou non. On retrouve alors dans cette classification les architectures «rien n'est partagé» et «les disques sont partagés» précédemment étudiés. Ce sont en général des systèmes multi-processeurs à mémoire partagée répartie**.** Les machines Intel Paragon, les IBM SP1 et SP2, ainsi que les réseaux de stations de travail sont des exemples de cette catégorie.
- **Les architectures massivement parallèles MPP *(Massively Paralel Processors systems)*** : elles comprennent des centaines, voire des milliers de processeurs. Les sous classes de ce modèle incluent les machines SIMD ou MIMD à mémoire virtuelle partagée ou répartie.
- **Les architectures multi-ordinateurs *(Multi-Workstations)*** : comprennent plusieurs ordinateurs connectés par un lien haut débit et localisés dans une même pièce (les CM5 de Thinking Machine, peuvent par exemple, être accolées les unes aux autres). Certains considèrent que les réseaux de machines connectées sur un réseau Ethernet (donc à faible débit) font partie de ce type de système.

## 1.3. Synthèse : définition d'une architecture hybride

Les différentes classifications présentées démontrent à quel point les architectures matérielles de ces machines sont liées à des classes d'application données. Pour couvrir le spectre des besoins de développement actuels (calcul numérique intensif, gestion de bases de données importantes), l'intégration de ces différents types de machines au sein d'un même réseau d'interconnexion physique est donc devenu inévitable. Nous proposons sur la Figure 3 notre vision des évolutions et des tendances actuelles des architectures matérielles.

Les architectures de type NORMA intègrent de plus en plus des machines de type MPP ou COMA. Les architectures qui en résultent sont appelées *réseaux hybrides*, car constituées de machines qui fonctionnent de manière radicalement différentes et qui interopèrent difficilement. Décrire des architectures de réseaux hybrides de machines requiert la connaissance non seulement des capacités de traitement des machines, mais aussi des topologies des réseaux d'interconnexions sous-jacents. Ainsi, une architecture matérielle est hybride, si elle est composée de plusieurs machines de types différents (UMA, NORMA, etc.) qui communiquent à travers un réseau d'interconnexion.

Il faut néanmoins tempérer cette vision purement matérielle de l'architecture hybride. En effet, si on se place dans une perspective de placement dynamique d'applications parallèles et réparties, une architecture peut-être hybride pour l'exécution d'une application et pas d'une autre, en fonction de la charge et des processeurs cibles des programmes compilés. C'est pourquoi nous proposons notre propre définition d'une architecture matérielle hybride.

***Définition 1 :*** *Architecture matérielle hybride*

*Une architecture matérielle est dite hybride si elle est composée de machines mono et multi-processeurs interconnectées et utilisées pour l'exécution d'une même application.*

On différencie de plus une architecture matérielle hybride en fonction de la distance entre les deux machines les plus éloignées et du nombre de réseaux informatiques traversés. Ainsi, une architecture matérielle hybride locale est définie comme :

***Définition 2 :*** *Architecture matérielle hybride locale*

*Une architecture matérielle hybride est locale si le réseau physique qui la supporte est un réseau local.*

Nous pensons que les réseaux hybrides locaux vont connaître une croissance très forte et qu'ils vont devenir le support d'exécution de nombreux types d'applications jusqu'alors réservés aux machines parallèles et aux ordinateurs centraux. Notre méthode MEDE-VER nécessitant la description de ce type d'architectures matérielles, nous avons étudié l'adéquation entre ces nouvelles architectures et les langages ou logiciels susceptibles de les décrire. C'est pourquoi le chapitre suivant présente un état de l'art des langages de description d'architectures matérielles, réalisé en fonction de critères que nous précisons.

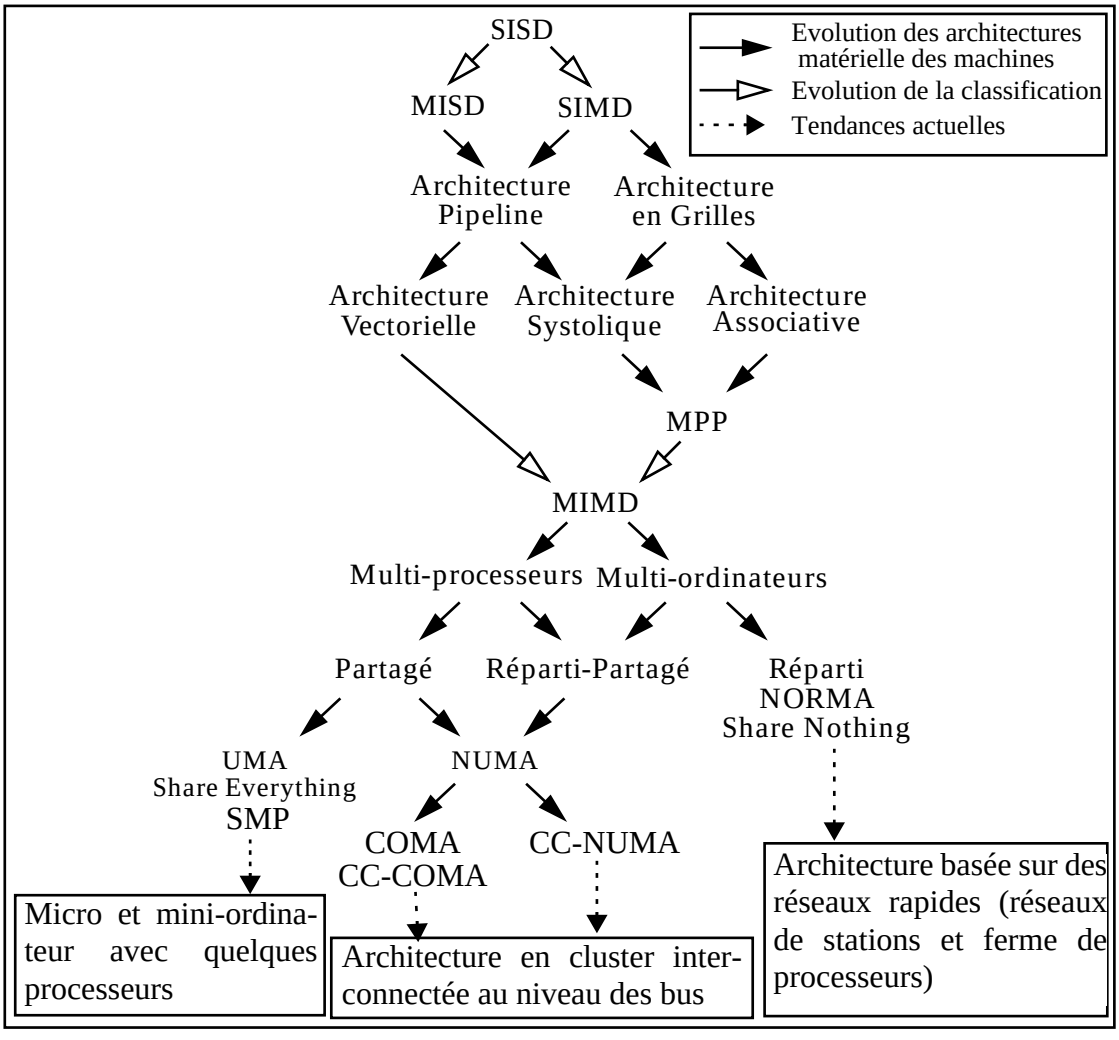



**Figure 3 :** Notre vision de l'évolution des architectures parallèles

## 2. Langage de description d'architectures matérielles

Les informations nécessaires à l'administrateur d'un réseau de dizaines de machines sont, en général, différentes de celles du programmeur d'une application parallèle. Ce dernier désire connaître, à un instant donné, les machines sur lesquelles il peut exécuter son programme étant donné certaines contraintes (tel qu'un programme ne fonctionnant que sur des processeurs Intel avec comme système d'exploitation Windows NT).

De notre point de vue, un langage de description d'architectures matérielles doit répondre aux besoins énoncés ci-dessous.

*Au niveau du langage de description :*

- **Convivialité** : la syntaxe du formalisme textuel doit être simple et conviviale et disposer d'une relation bijective avec le formalisme graphique (s'il existe) ;
- **Portabilité** : l'interfaçage avec des logiciels participant au cycle de développement et d'exécution d'une application (tels que les logiciels de répartition de charge, les logiciels de prototypage rapide et les logiciels de simulation de files d'attente) doit être possibles ;
- **Dynamicité** : permettre la mise à jour dynamique de la description, dans le cas d'architectures reconfigurables dynamiquement (machine à base de Transputer [Inmos 88] par exemple).

*Au niveau des objets décrits :*

- **Extensibilité** : les propriétés des objets représentables varient dans le temps, c'est pourquoi il est souhaitable de pouvoir les modifier (mise à jour incrémentale) ;
- **Granularité** : les formalismes utilisés pour décrire une architecture matérielle décrivent des objets (une station de travail par exemple) et éventuellement les composants d'un objet (les constituants d'une station de travail tels que les processeurs, les bus et les disques). L'utilisation de la hiérarchie dans le formalisme de description offre des niveaux de granularités variables. Un élément d'un niveau donné étant composé de l'agrégation d'éléments d'un niveau de hiérarchie inférieur ;
- **Réutilisabilité** : la construction de l'architecture matérielle doit être réalisée en réutilisant des éléments déjà existants et stockés dans des librairies.

La prise en compte de ces besoins, dans un contexte de développement, de simulation, d'exécution et de gestion d'applications réparties ou parallèles, nous a amené à un découpage de notre état de l'art en cinq parties. Nous étudions alors :

1) *les langages de programmation parallèle*. Ces langages spécifiquement dédiés à des machines parallèles (comme le langage C*) ou tout simplement généralistes (comme le langage HPF), fournissent aux programmeurs des moyens d'utiliser un schéma d'interconnexion fixe des processeurs ou de configurer statiquement et dynamiquement l'architecture matérielle d'exécution ;
2) *les langages de configuration*. Ils sont utilisés de manière statique ou dynamique pour placer les éléments logiciels d'une application sur les processeurs ou les machines d'exécution ;
3) *les logiciels de simulation de systèmes informatiques*. Ils sont utilisés pour concevoir des architectures de machines, des architectures de réseaux, des protocoles de communication, etc. Ils disposent de langages de description généralement propriétaires, qui sont utilisés pour décrire de manière fine des architectures matérielles, mais aussi pour simuler leur comportement. La simulation est supportée par du code automatiquement généré, qui est soit directement interprété par un simulateur, soit compilable et exécutable de manière autonome ;

4) *les logiciels de répartition de charges.* A partir des informations statiques (puissance, type de processeur, taille du disque dur), mais aussi dynamiques (charge moyenne de la machine, pourcentage de mémoire libre) des éléments de l'architecture matérielle ces logiciels prennent des décisions concernant le placement et la migration d'applications. Les moyens de description et de test de la validité des informations sur l'architecture matérielle est donc primordiale dans ce genre de logiciels, si on veut éviter de mauvais choix de placement ;

5) *les applications d'administration de systèmes et de réseaux*. Ces logiciels collectent automatiquement des informations générales et de très bas niveaux sur les éléments matériels d'une architecture en fonctionnement.

## 2.1. La description de l'architecture matérielle dans les langages de programmation parallèle

Les langages de programmation parallèle récents ont dès leur création intégré un sous ensemble de mots clefs spécifiquement liés à la description du placement de modules logiciels sur une architecture matérielle cible.

Notre intérêt pour ce type de langage est double. Dans un premier temps, nous nous sommes intéressés aux modes de description de l'architecture matérielle, de la portabilité, de l'extensibilité et aux limites de ces descriptions. Puis, nous avons évalué la granularité des descriptions produites. Pour effectuer notre étude, nous avons dissocié les langages de programmation dédiés à des architectures matérielles spécifiques de ceux multi-plateformes. Nous concluons sur le fait que ces langages sont limités à la fois en terme de portabilité, de dynamicité et de gestion de la granularité.

### *2.1.1. Les langages de programmation parallèle dédiés à des architectures matérielles spécifiques*

Certains langages de programmation ont été créés pour des processeurs ou des machines spécifiques. La description précise de l'architecture dans ce cas est souvent inutile, car elle est sous entendue et intégrée dans le langage. Les deux exemples que nous avons choisis d'étudier sont Occam qui fonctionne sur des machines à base de Transputers et C*, un dérivé du langage C, adapté à la Connection Machine.

*Le langage Occam*

Le langage Occam [Hoare 88] qui ne fonctionne que sur des processeurs spécifiques : les Transputers (fabriqués par la société Inmos). Deux notations existent en Occam pour définir le type d'un processus : les processus parallèles sont déclarés par le mot clef PAR et les processus séquentiels par le mot clef SEQ. Les interactions entre processus se font via des canaux en utilisant les primitives d'envoi (!) et de réception (?) de messages sur un canal. Le programmeur peut gérer la configuration de l'application en indiquant quel processus s'exécute et sur quel processeur (mot clef PLACED PAR). Des priorités sont assignables aux processus qui s'exécutent sur un même processeur [Gaudiot & al. 92]. Dans ce cas précis, programme et description de l'architecture sont très intimement liés.

Néanmoins, si l'on génère automatiquement du code source Occam à partir des spécifications d'une application parallèle, il devient possible de séparer le programme et son placement sur l'architecture. On peut ainsi tester de nombreuses configurations différentes sans modifier le programme source. TAPIOCA [Bréant & al. 93] est un exemple d'outil de génération automatique de code Occam, à partir d'une spécification en Réseaux de Petri Colorés [Jensen 92]. La description du nombre de processeurs, ainsi que la topologie du réseau d'interconnexion sont indispensables à la génération des directives de placement (PLACED PAR). C'est pourquoi, le formalisme de description de l'architecture cible utilise les caractéristiques des machines à base de transputers :

- pour un processeur : un identificateur, une adresse, un type, une taille mémoire et un temps d'accès à cette mémoire.

- pour un lien de communication : un identificateur, une adresse et un débit. Chaque processeur dispose d'au plus quatre liens de communication.
- pour une machine à base de Transputers : un type et la taille mémoire du Transputer racine ;

Les travaux sur les langages de modélisation et la configuration statique et/ou dynamique des architectures matérielles basées sur des réseaux de Transputer sont nombreux et on peut citer entre autres Trollius [Burns & al. 90], NETMON-II [Zitterbart 90], C_NET [Adamo & al. 92] et Visputer [Zhang & al. 95]. Tous se sont attachés à décrire l'architecture de ces machines et éventuellement à gérer leur reconfiguration en cours d'exécution de manière statique. La description obtenue est simple, peut être graphique ou textuelle, mais n'utilise pas la notion de hiérarchie.

*Le langage C**

Le langage de programmation C* [TM 88] est un autre exemple de langage dédié à une machine spécifique. C'est une extension du langage C, réalisée pour la Connection Machine. Cette machine dispose d'un ensemble de processeurs possédant chacun une mémoire locale et qui exécutent tous le même code sur des données qu'ils peuvent s'échanger. La différence majeure entre C et C* est qu'à chaque donnée on doit associer un processeur. La notion de noeud virtuel est alors utilisée pour gérer les cas où les données sont plus nombreuses que les processeurs (plusieurs noeuds virtuels étant alors regroupés sur un unique processeur). Un programme C* est créé pour s'exécuter sur une machine unique, il est donc inutile d'utiliser un langage de description d'architecture. Le fait de rajouter de nouveaux types de données est suffisant pour faciliter la tâche d'optimisation et de génération de primitives de placement du compilateur. Tout se passe comme si l'architecture était implicitement définie.

*Discussion*

Bien que très puissant et permettant d'obtenir d'excellentes performances, les langages de programmation pour architectures matérielles spécifiques possèdent de nombreux inconvénients. D'abord, ils rendent les utilisateurs captifs d'une machine et d'un constructeur sur un marché où même les plus prestigieuses sociétés ont connu des problèmes de trésorerie (Cray, Thinking Machines et KSR). Ensuite, ils empêchent d'écrire des programmes portables et réutilisables. Enfin, ils imposent une connaissance approfondie de l'architecture de la machine si l'on désire obtenir de performances optimales. Le manque d'outils de déboggage et de mesure des performances est aussi un handicap.

**Tableau 1:** Les langages de programmation parallèle dédiés à des machines

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Non, lié au langage de programmation | Non, lié à une machine donnée | Non, car tout est statiquement décidé lors de la compilation | Non, on utilise uniquement ce qui est définit dans le langage | La granularité est basée sur la connaissance de la machine | Non |

### *2.1.2. Les langages de programmation parallèle non dédiés à des architectures matérielles spécifiques*

Les spécificités des machines parallèles ont un inconvénient majeur, ils empêchent la portabilité des programmes. Las de devoir reprogrammer leurs applications, les utilisateurs ont fait pression pour obtenir des langages normalisés indépendants des architectures cibles. Il a donc fallu construire à la fois des compilateurs conformes aux normes, mais aussi toute la chaîne des outils de développement associés. Deux exemples bien connus sont le langage Ada [Ada 83 & 95] et plus récemment le langage High Performance Fortran [HPF 93 & 94].

*Le langage HPF*

Le langage HPF est une extension du langage Fortran 90, particulièrement adapté aux machines multi-processeurs à mémoire répartie. La distribution des données est spécifiée dans le corps du programme à l'aide de directives `!HPF$` placées en commentaires dans le code source Fortran. Trois niveaux de descriptions sont offerts par HPF : les tableaux de données, les espaces d'indices et les processeurs virtuels (cf. Figure 4).

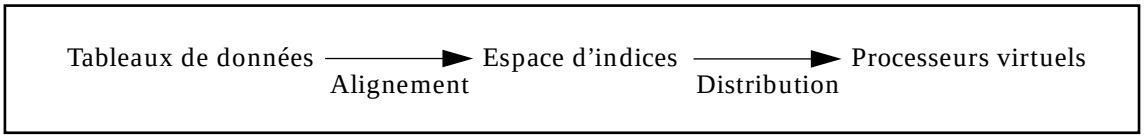



**Figure 4 :** Le modèle de placement à deux niveaux de HPF

Les tableaux de données sont alignés sur un espace d'indices appelés TEMPLATES. Le réalignement dynamique durant l'exécution est possible. Les espaces d'indices sont ensuite placés sur des processeurs virtuels. La directive DISTRIBUTE désigne le placement d'un TEMPLATE sur les processeurs physiques, ce qui correspond au placement de tous les tableaux alignés sur ce TEMPLATE. Chaque dimension du TEMPLATE est distribuable par bloc ou de manière cyclique sur les processeurs associés ou assignés à un unique processeur avec la clause *.

*Discussion*

Finalement, dans des langages comme HPF la description de l'architecture est complètement occultée du fait du modèle de données utilisé. Cette opacité du code complique le contrôle des choix sur les performances obtenues et complique la maintenance et la traçabilité du code applicatif.

**Tableau 2:** Les langages de programmation parallèle généraux

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Non, lié au langage de programmation | Oui. Les performances sont néanmoins liées au compilateur | Non, car tout est statiquement décidé lors de la compilation | Non, on utilise uniquement ce qui est défini dans le langage | La granularité est basée sur la notion de processeur virtuel | Non |

### *2.1.3. Synthèse*

Les modes de description de l'architecture matérielle dans les langages de programmation parallèle sont le plus souvent intégrés dans le langage de programmation. C'est donc un compilateur spécifique qui va produire le code et les directives de placement. Il en résulte une portabilité et une qualité de description de l'architecture matérielle limitées.

L'utilisation d'un langage de description d'architecture matérielle hybride n'est donc pas utile dans le cas de langage de programmation parallèle, sauf si :

- on se place dans le cadre de la génération automatique de code source ;
- on connaît déjà l'architecture matérielle avant de programmer son application.

## 2.2. La description de l'architecture matérielle dans les langages de configuration

La mise en place progressive dans les sites informatiques de machines parallèles et de réseaux de stations de travail nécessite de porter les anciennes applications centralisées sur ces nouvelles architectures. Pour éviter de tout refaire à partir de rien, certains ont eu l'idée de garder leurs programmes tels quels, en comblant les lacunes des langages de programmation dont ils disposaient et en ne modifiant que les environnements d'exécu-

tion des applications. Ainsi, seul le programme chargé de les charger et de les lancer *(loader)* et/ou l'environnement d'exécution *(run-time)* sont à mettre à jour. Il faut quand même fournir des directives de placement à ces environnements pour que l'application puisse être exécutée sur une architecture matérielle donnée. Cette problématique est assez proche de celle développée dans MEDEVER, c'est pourquoi nous nous y intéressons.

### *2.2.1. Etat de l'art*

Dans un langage de configuration la description de l'architecture matérielle est nécessaire, car il décrit les relations entre des programmes et une architecture matérielle évolutive en fonction des phases de calcul d'un programme. Un langage de configuration utilise une description unique et centralisée d'un programme réparti. Dans la suite, nous présentons les langages Pronet, Darwin et Sysl qui nous ont paru intéressants par rapport à notre approche.

*PRONET*

Le langage de programmation orienté messages PRONET [Leblanc & al. 82] (*Process and Network*), dispose de deux sous langages distincts : NESTLA et ALSTEN. Le premier est un langage de configuration spécifiant l'architecture logique (au niveau fonctionnel) et ses reconfigurations possibles sur réceptions d'évènements. Le second décrit les processus. NESLA décrit le placement initial des processus et assigne les ports de communication aux processeurs.

Notons qu'il n'est pas possible de spécifier des contraintes sur les ressources telles que le choix d'un type de processeurs. D'autre part, NESTLA est lié au langage de programmation ALSTEN et n'a pas été prévu pour être réutilisable dans un autre contexte.

*DARWIN*

Le langage de configuration Darwin [Magee & al. 92], successeur du langage de configuration du système CONIC [Dulay 90 et Magee & al. 89], décrit le placement de groupes d'instances de processus communiquant par messages sur une architecture de type MIMD. La technique utilisée pour décrire l'architecture matérielle est simple : on numérote le nombre de processeurs, car ils sont tous similaires. L'environnement d'exécution est ensuite mis à contribution pour le placement effectif des programmes.

L'inconvénient majeur de Darwin est l'utilisation des processeurs virtuels numérotés, qui ne correspondent pas toujours à des processeurs réels.

*SYSL*

SySL [Sommerville & al. 92 et Dean & al. 92], pour System Structure Language, est un langage orienté objet pour la description des architectures matérielles, le développement des logiciels et de la documentation associée. SySL encapsule la notion de classes de systèmes partageant des propriétés (ou valeurs d'attributs). Chaque attribut est complété quand un élément de la classe est instancié. Les dépendances entre classes sont exprimées dans le langage par des relations. Un outil graphique a été développé pour faciliter la description des dépendances entre classes et offre différentes vues d'un même modèle.

L'inconvénient majeur de SySL est qu'il n'est pas destiné à l'utilisateur final, car c'est un véritable langage de programmation orienté objet. Il nécessite la programmation de classes et d'attributs qui se rapportent à l'architecture à décrire. Ce langage est par contre extensible à travers l'utilisation de classes.

*Discussion*

La principale différence entre Darwin et PRONET d'une part et SySL d'autre part est le niveau de granularité des objets gérés. Darwin et PRONET s'intéressent à la composition de modules logiciels au travers d'interfaces d'accès clairement définies. Cet intérêt pour la spécification des interfaces rend ces langages souhaitables pour la construction et le placement d'applications réparties. La description de l'architecture matérielle n'a alors

d'intérêt que si elle est couplée avec un langage de programmation des composants à placer (ALSTEN pour PRONET et les modules TASK dans CONIC). SySL, au contraire, est indépendant des langages programmation et n'impose aucune contrainte sur les composants. SySL est un langage de systèmes répartis à gros grain ou les composants à gérer sont eux-mêmes des applications. SySL est indépendant de tout langage de programmation et de tout système d'exploitation.

**Tableau 3:** Les langages de configuration

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Oui, Sysl en est un exemple flagrant | Possibilité de reconfiguration dynamiquement (cf. Darwin) | Oui | Variable | Oui |

### *2.2.2. Synthèse*

De cette étude, nous avons retenu trois points importants :

- un langage de configuration complètement dissociée du code de l'application est d'un grand intérêt. On dissocie alors complètement l'architecture logicielle et l'architecture matérielle servant à l'exécution, tout en assurant la portabilité du langage de description.
- la description modulaire de l'architecture, avec éventuellement l'utilisation de la hiérarchie, offre un pouvoir de description suffisant pour des taches de placement et d'administration de machines. Il faut par contre éviter que les composants soient d'une granularité trop grande (comme dans PRONET par exemple).
- il faut trouver un équilibre acceptable entre l'extensivité et la charge de programmation induite par la description de l'architecture matérielle (Sysl est par exemple un langage très puissant, mais difficile d'accès au non programmeur). L'utilisateur doit pouvoir ajouter des attributs spécifiques à sa description, sans que cela ne lui impose de la programmation.

## 2.3. La description de l'architecture matérielle dans les logiciels de simulation

La recherche de résultats quantitatifs sur des spécifications d'un système informatique complexe n'est en général pas possible. Les problèmes qui ne sont pas solvables de manière analytique doivent être redéfinis avec des formalismes qui se prêtent à la simulation. On peut répertorier trois grandes techniques de simulation :

1) la simulation de systèmes informatiques par évènements discrets;
2) la simulation de systèmes informatiques par des traces;
3) la simulation de systèmes informatiques par des modèles objets.

Dans la suite de ce chapitre, nous présenterons les contraintes imposées par ces techniques de simulation sur la description de l'architecture matérielle modélisée. Nous en déduisons alors que les modélisations des architectures matérielles, bien que conceptuellement très riches, n'offrent pas de correspondances simples avec des architectures réelles.

### *2.3.1. La simulation de systèmes informatiques par évènements discrets*

Cette simulation est très largement basée sur des formalismes à base de files d'attente [Kleinrock 75] modélisant des phénomènes de partage de ressources. Une ressource est une composante physique ou logique que les entités considérées, appelées clients, d'un système doivent obtenir avant de réaliser une activité. Ainsi, dans le domaine des systè-

mes informatiques, le client est un processus qui est en attente d'une ressource processeur et dont l'activité est quantifiée en fonction de son temps de traitement [Gelenbe & al. 80]. Une file d'attente est caractérisée par :

- A : la suite des instants d'arrivée des clients dans la file d'attente ;
- S : la durée des services ;
- C : le nombre de serveurs ;
- K : la capacité maximale de la file ;
- L : la population des usagers ;
- DS : la discipline de service.

La plupart du temps, l'activité d'un client nécessite tout au long de son cycle de vie l'accès à une succession de ressources, on parle alors d'un réseau de file d'attente [Allen 80]. On comprend alors que ces paramètres se retrouvent systématiquement dans les langages de description d'architecture utilisés pour la simulation par évènements discrets. Le produit de simulation leader dans cette catégorie est sans nul doute QNAP2 et les simulateurs qui l'utilisent ou qui en sont dérivés.

*QNAP2*

Le logiciel de simulation QNAP2 (*Queuing Network Analysis Package*) [Potier 86 & Simulog 88], est un outil de description et d'analyse de réseaux de files d'attente. Il provient du travail conjoint de chercheurs de l'INRIA et de Bull. Ce produit, programmé en Fortran 77 pour en assurer la portabilité, est actuellement diffusé par la société Simulog.

Un programme QNAP2 se décompose en trois parties :

1) la configuration du réseau
2) le traitement effectué sur chaque station
3) le contrôle de la résolution du réseau

Un réseau de file d'attente est composé d'un ensemble de stations à travers lesquelles circulent des clients, selon des règles de routage données. Dans QNAP2, la création de classes de clients simplifie la programmation et permet une meilleure compréhension des résultats de la simulation.

Une station est caractérisée par son nom, son type (positionné par défaut comme un serveur unique), la discipline de service de la file d'attente (FIFO par défaut), l'initialisation du nombre de clients par classe, le paramètre de service pour chaque classe de clients, le paramètre de transit décrivant les règles de routage des clients à la fin de leur service, la priorité de chaque client entrant dans la station en fonction de sa classe et le service d'allocation de quantum pour chaque classe de clients dans la station.

Les solveurs présents dans QNAP2 fournissent des résultats de simulation exacts (CONVOL [Baskett & al. 75], MVA [Reiser & al. 80], etc.) ou approchés (HEURSNC [Neuse & al. 81], DIFFU [Gelenbe & al. 77], etc.).

*Modarch*

Modarch est un logiciel destiné à l'évaluation de performance d'architectures réparties, qui est commercialisé par la société SIMULOG. Modarch décrit les architectures matérielles et logicielles de manière séparée et graphique. A partir des descriptions, Modarch génère du QNAP2 qui est ensuite utilisé pour la simulation. Modarch utilise un formalisme graphique de représentation des architectures matérielles qui est composé des objets suivants :

- les processeurs possédant les attributs suivants : le nom, la taille mémoire, la puissance en Mips (ou millions d'instructions par seconde) et le débit en lecture et écriture.

- les unités de stockage possédant les attributs suivants : la capacité de stockage, le débit en lecture et écriture.
- le bus Ethernet de communication possédant les attributs suivants : le débit et le taux d'erreurs.
- les liens point à point possédant les attributs suivants : le débit, le taux d'erreurs, le temps de propagation et le protocole (HDLC ou aucun).

Une bibliothèque d'architecture existe et offre les caractéristiques des machines les plus courantes. Du fait du lien très étroit qui existe avec QNAP2, l'utilisateur doit modéliser son architecture avec des réseaux de files d'attente et ne peut pas créer de nouveaux types de composants.

*SES Workbench*

SES/Workbench, commercialisé par la société SES (*Scientific and Engineering Software*), est un outil disposant d'une interface graphique très conviviale et utilisant un formalisme à base de réseaux de files d'attente. Un modèle est constitué d'une hiérarchie de graphes de noeuds qui communiquent entre eux via des transactions. Les transactions sont réalisées si les ressources dont elles ont besoin sont accessibles. SES/Workbench dispose de trois types de ressources :

- les ressources actives correspondent à des serveurs de files d'attente ;
- les ressources passives correspondent à un groupement de jetons ou de données ;
- les ressources logiques correspondent à des prédicats booléens.

Il existe de nombreux types de noeuds dans SES/Workbench qui réalisent des opérations diverses telles que :

- la gestion des synchronisations et des accès aux ressources ;
- la gestion des transactions (vie, mort, interruption, transfert) ;
- la gestion de la hiérarchie ;
- l'insertion de portions de code C ou du langage «maison» SES/Sim.

Le principal inconvénient de SES/Workbench est lié à son formalisme unique contraignant. Par contre, son interface graphique est à la fois ergonomique et conviviale, y compris durant la simulation.

*Discussion*

Les réseaux de file d'attente ont été conçus pour représenter le flux d'informations dans un système. Les environnements de simulation actuels basés sur des solveurs puissants étendent les capacités de modélisation de systèmes. Mais, le formalisme de réseau de file d'attente classique est limité et **ne permet pas de modéliser aisément les contraintes qui s'exercent entre les services** (telles que les conflits d'accès, les droits de préemption, les communications) **et l'architecture matérielle utilisée** [Toutain 91].

**Tableau 4:** Les logiciels de simulation à évènements discrets

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Non, lié en général au simulateur | Non | Non | Variable<br>Gestion de la hiérarchie | Oui, par stockage dans des librairies |

Néanmoins, des travaux récents menés par [Bacceli & al. 89] et par [Pekergin 91] avec le simulateur ARCHISSIME et son successeur OPPSI [Cubaud & al. 92 & Pooley 91]

visent à créer un sur-ensemble du modèle des réseaux de file d'attente permettant de gérer la concurrence et les primitives de synchronisation au sein des applications.

### 2.3.2. La simulation et l'émulation de systèmes informatiques par des traces

De manière complémentaire à des modèles à base de files d'attente, on a recours à des algorithmes basées sur l'utilisation de traces. La simulation peut se faire soit en enregistrant les traces produites lors d'une exécution réelle en utilisant un environnement d'exécution parallèle [Burns 92] et [Zhou 88], soit en générant les traces à partir d'une description simplifiée d'une application parallèle [Hémery 94]. La simulation par utilisation de traces est adaptée aux besoins de traçabilité de notre méthode MEDEVER.

*Parallel Proto*

Parallel Proto [Burns 92] permet à l'utilisateur de décrire une architecture matérielle en connectant trois type d'objets : les processeurs, les bus, les mémoires. Chacun possède des attributs qui lui sont propres et qui sont spécifiés en unité de simulation. Ainsi, un processeur est caractérisé par sa vitesse d'exécution, la mémoire par son temps d'accès en lecture et écriture et le bus par le temps de propagation. Des attributs peuvent être rajoutés à chaque objet. Un fichier de configuration peut-être généré pour des simulations sur des machines standards telles que le Multimax d'Encore et l'Hypercube d'Intel.

L'avantage principal de ce modèle très simple, mais très incomplet (il ne gère que trois types d'objets), est qu'il s'intègre parfaitement dans une méthodologie de développement appelée REE (*Requirement Engineering Environment*).

*Un éditeur de topologie de machines MIMD*

Un autre exemple d'éditeur de topologie couplé avec un outil de simulation est donné par [Hémery 94]. Cet éditeur décrit des machines MIMD de type multi-ordinateurs grâce à un graphe de machines. Les liens de communications entre machines sont représentés par des arcs non orientés. Un site est caractérisé par son nom, sa mémoire disponible, la puissance relative de son processeur par rapport aux autres sites et les ressources particulières présentes sur le site (telles qu'une carte vidéo accélératrice, un disque etc.). Les liaisons entre sites sont point à point et s'il n'existe pas de lien entre deux sites, le routage utilisé est celui du chemin le plus court (calculé de manière statique).

Une architecture matérielle est caractérisée par des paramètres globaux représentant les coûts unitaires en temps logique des instructions tracées (telles que la création d'une tâche), le coût de communication, l'attente, etc.

Cet éditeur de topologie possède deux inconvénients :

1) tous les liens de communication sont point à point. Or si on considère un réseau Ethernet simple, toutes les machines sont connectées au même support physique, ce qui n'est pas modélisable avec cet éditeur.
2) il est difficile de différencier une machine multi-processeurs d'une machine mono-processeur, car il n'existe pas de noeud hiérarchique.

*Discussion*

Si Parallel Proto dispose d'un langage de description très simple, il a néanmoins l'avantage d'être intégré dans un environnement de développement, qui permet de créer ses propres librairies. L'éditeur d'Hémery est intéressant, dans le sens où il fournit un modèle très proche des machines MIMD réelles.

**Tableau 5:** Les logiciels de simulation basés sur des traces

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Non | Non | Non | Gros grain, pas de hiérarchie | Oui, par stockage dans des librairies |

On peut regretter qu'aucun de ces modèles ne soit hiérarchique (la description est à plat) et que les informations utilisées sont uniquement celles nécessaires à la simulation. **Mais, le problème principal vient du fait qu'il n'existe pas de règles simples pour trouver les traces représentatives d'un modèle et les mesures que l'on désire effectuer.**

### *2.3.3. La simulation de systèmes informatiques par des modèles objets*

La modélisation de l'architecture par des modèles objets permet à la fois la réutilisabilité et une décomposition du modèle en une agrégation de sous modèles prédéfinis qui coopèrent. Les outils disponibles offrent des bibliothèques d'objets de hauts niveaux réutilisables et extensibles. Nous présentons dans la suite de ce chapitre, un outil basé sur un modèle objet utilisant QNAP2 (appelé NOMAD), ainsi que deux simulateurs construits à partir d'un modèle objet (OPTNET et NESSY).

*NOMAD*

NOMAD *(Nouvel Outil de Modélisation d'Architecture Distribué)* [Nachef 93] est fondé sur une méthodologie de modélisation par file d'attente de systèmes d'information répartis. NOMAD a été développé par Bull, pour évaluer les performances de son modèle DCM [Bull 91]. La méthodologie intègre des principes orientés objets (tels que les classes et leur héritage) au langage QNAP2. Ainsi, après une phase de modélisation du système, NOMAD génère automatiquement le modèle QNAP2 associé.

Le défaut actuel de cet outil est qu'il est issu d'une seule technique de modélisation. Or le processus d'optimisation utilisant des réseaux de files d'attente est lent. Un couplage avec des techniques d'optimisation combinatoire est actuellement à l'étude pour résoudre ce problème.

*OPNET*

OPNET *(Optimised Network Engineering Tool)* [MIL 91] est un outil de simulation développé au MIT et réparti par Delta Partner. OPNET est un outil graphique dédié à l'étude des réseaux de télécommunications. Au niveau architectural, OPNET se décompose en trois parties : la partie modélisation, la partie tests et la partie évaluation chargée de l'analyse des résultats. La description du modèle comporte trois niveaux de hiérarchie : le niveau réseau, le niveau noeud et le niveau processus. Chacun de ces niveaux disposent d'un formalisme graphique propre. Au niveau réseau, les instances de noeuds (provenant de classes prédéfinies ou créées par l'utilisateur) et de liens entre ces noeuds sont créés. Ensuite, chaque noeud est un conteneur acceptant en standard les modules suivants : les processeurs, les files d'attente, les générateurs de charge, les émetteurs (souvent appelés sources) et les récepteurs (souvent appelés puits). Enfin, au niveau processus, les automates à états finis sont édités. Ces automates représentent les processus à l'intérieur d'un module processeur ou file d'attente. Des programmes C faisant des appels systèmes Unix sont aussi utilisés comme des processus. Le programme de simulation est finalement généré automatiquement dans un langage spécifique (appelé EMA code) à partir du modèle.

Les deux points fort d'OPNET sont sa bibliothèque de modèles de réseaux et son interface graphique indéniablement conviviale et puissante. Il souffre néanmoins d'un manque d'ouverture vers l'extérieur et d'une interface avec le C très rudimentaire [Alimi 92].

*NESSY*

Développé au laboratoire MASI, NESSY *(Network Simulation SYstem)* [Soto 90 et Anelli 92] est un simulateur destiné à l'étude des réseaux de données. Il propose deux vues distinctes sur le modèle suivant le niveau de maîtrise du processus de modélisation (une pour le spécialiste et une pour l'utilisateur profane). L'utilisation de l'outil se fait donc en deux phases. La première, dédiée au modélisateur, permet la création des ADS

(Atome De Simulation) et des ADIS (Agrégat D'information Simulée) qu'ils s'échangent. La seconde, fournit à l'utilisateur les ADS, dont il se sert comme briques de base pour décrire son modèle. Les ADS sont décrits dans le langage textuel LASSY [Anelli 92], construit autour du modèle formel des automates d'états finis communiquants [Kurose & al. 88]. Les ADS sont ensuite compilés et stockés dans une bibliothèque. Le lancement de la simulation ne se fait pas à l'aide d'un programme généré, mais directement dans l'environnement.

*Discussion*

L'absence de langage de description intermédiaire de l'architecture du modèle dans OPNET est un inconvénient majeur (tout doit être écrit en langage C et re-compilé à chaque modification). NESSY dispose du langage LASSY, mais ce dernier étant dédié à la simulation est trop complexe pour être considéré comme un langage de description dédié à l'architecture matérielle (et pas des données qui transitent dessus).

---

Tous ces simulateurs ont une interface graphique de très bonne qualité (sauf NOMAD) et disposent de modèles hiérarchiques stockés dans des bibliothèques plus ou moins extensibles. Il est possible de rajouter des attributs aux objets décrivant l'architecture matérielle et de les faire varier lors de la simulation. Il manque cruellement à ces simulateurs des outils de développement pour l'extension des bibliothèques prédéfinies. **Enfin, l'approche modulaire de description des architectures conduit à un niveau de détails souvent trop important pour nos besoins**.

---

**Tableau 6:** Les logiciels de simulation basés sur des modèles objets

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Non, description liée à la plate-forme de simulation | Non | Oui | Variable avec gestion de la hiérarchie | Non |

### *2.3.4. Synthèse*

Les formalismes de description des architectures matérielles pour les simulateurs ne disposent que des informations nécessaires à la simulation. Ainsi, seuls des informations de topologie, de routage et de charge sont disponibles. La simulation ne prend donc pas en compte les paramètres matériels réels des architectures cibles, ce qui rend la correspondance entre le modèle et la réalité difficile. D'autre part, les simulateurs sont encore en majorité des programmes centralisés avec un flot de contrôle unique et à gros grain (ainsi par exemple QNAP2 ne gère pas les processus légers et OPNET génère un programme C centralisé).

---

De cette étude, nous avons retenu deux points importants :

- la nécessité d'intégrer la description de l'architecture au sein d'un environnement de développement complet et ouvert. Cette intégration se fait par l'intermédiaire de bibliothèques contenant à la fois des informations de modélisation (discipline de service d'une file d'attente dans QNAP2 par exemple) et de caractérisation de la machine réelle (adresse physique et logique, sa puissance, etc.).
- la nécessité d'offrir un lien vers des simulateurs du marché (tels QNAP2) pour laisser à l'utilisateur la possibilité de simuler sur des architectures de machines non disponibles (ou non encore réalisée) des calculs de performance.

---

## 2.4. La description de l'architecture matérielle dans les logiciels de répartition de charge

Les outils de répartition de charge rendent le placement de processus transparent sur des machines les moins utilisées [Nichols 87]. L'architecture matérielle cible est encore très souvent un réseau de stations de travail sous Unix. Dans ce cadre, la description de l'architecture matérielle est simple. La description d'architecture de réseaux interconnectés et hybrides pose par contre des problèmes dus aux niveaux de granularité différents des objets représentés et aux multiples équipements d'interconnexions des réseaux informatiques. Un supercalculateur disposant de 1024 processeurs n'ayant absolument rien à voir avec une station de travail en terme de description de ses composants principaux.

### *2.4.1. Etat de l'art*

Nous avons étudié trois logiciels de répartition de charge. Utopia et «Express for Workstations» sont des produits commerciaux et Gatostar est un produit universitaire.

*Gatostar*

Gatostar [Folliot & al 95] est un produit universitaire issu des recherches menées au laboratoire MASI (Paris). C'est un outil de répartition de charge fonctionnant sur un ensemble de machines hétérogènes sur des réseaux locaux interconnectés. Il dispose d'un langage très sommaire de description de l'architecture matérielle. Dans ce langage, les deux objets permettant de décrire l'architecture sont les processeurs et les liens entre ces processeurs. Un processeur comporte les informations suivantes : un nom unique qui sert à son référencement, un type (Pentium, Sparc), une vitesse (en MIPS) et des attributs matériels (capacité mémoire, capacité de stockage, vitesse d'accès aux ressources disques et mémoire). Les liens (arcs) qui unissent les différents processeurs (noeuds) forment un graphe. Les caractéristiques des liens sont leur nom, la topologie du segment auquel ils appartiennent (Ethernet, Token Ring) et leur débit moyen mesuré. Les différents réseaux se représentent par leur nom et par les passerelles permettant de communiquer d'un réseau à un autre. La notion de groupe de machines a été introduit pour permettre de regrouper des machines qui rendent un même service à des utilisateurs (par exemple les machines qui disposent d'une ressource particulière).

*Utopia*

Utopia [Zhou & al. 93] est un produit commercial issu des recherches menées à l'université de Toronto. Semblable à Gatostar quant à l'approche de modélisation, il en diffère néanmoins en terme de gestion des informations et de représentation hiérarchique des réseaux. Ainsi, des attributs supplémentaires sont attachés aux objets pour permettre de collecter les différents indices de charge nécessaires au calcul du placement (taux d'occupation du processeur, pourcentage en moyenne de mémoire libre, débit mesuré des entrées sorties disque, taille du répertoire temporaire de chaque machine, nombre de sessions actives sur la machine, etc.). La création de clusters qui constituent des sous réseaux physiques (regroupement de ressources matérielles) ou virtuels (similaires aux groupes de Gatostar) est indispensable à la modélisation de grands réseaux (ou WAN). Enfin, un module graphique de visualisation en temps réel des indices de charges et des machines, que l'utilisateur peut choisir, facilite la remontée des informations à l'utilisateur.

*Express for Workstations*

Express for Workstations de Parasoft Corp est un produit commercial qui dispose d'un outil appelé Domtool spécifiquement destiné à la représentation de l'architecture matérielle de réseaux hybrides. La définition de types de machines (Station Sun, HP ou machines parallèles par exemple), ainsi que la représentation de tous les types de topologie en font un outil de modélisation puissant. Par contre, Domtool ne dispose pas de la hiérarchie et impose que les liens de communications soient point à point.

*Discussion*

Les outils de répartition de charge décrivent les architectures matérielles avec des informations fortement liées à l'architecture réelle (la taille de la mémoire, du disque dur, etc.). Conçus à l'origine pour fonctionner sur des réseaux locaux (pour des raisons de performance), ils gèrent dorénavant des architectures hybrides et multi-réseaux. Il en résulte alors un besoin d'amélioration des outils de description de l'architecture disponible sur le site de travail. Or, jusqu'à présent la description graphique de l'architecture supportant la répartition de charge n'était pas fournie. Seul l'administrateur du réseau savait manipuler les fichiers de description textuelles d'informations de bas niveaux nécessaires. Ce n'est plus le cas dans les logiciels commerciaux récents tels que Domtool.

**Tableau 7:** Les logiciels de répartition de charge

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Oui, gestion des réseaux hybrides | Oui | Non, dus aux algorithmes de répartition de charge | Gros grain, liée à la gestion des machines | Oui, stockage dans des fichiers de configurations |

### *2.4.2. Synthèse*

Les logiciels de répartition de charge ont, en général, une connaissance précise du parc de machines qu'ils gèrent. La régulation de la charge est réalisée en utilisant un nombre restreint de paramètres de charge mis à jour à partir de la collecte périodique des informations sur chacune des machines. Ces paramètres sont aisément intégrables dans un langage de description d'architecture, en fonction du type d'application à gérer.

## 2.5. La description de l'architecture matérielle dans les logiciels d'administration de systèmes et de réseaux

Les outils d'administration de réseaux comblent les besoins des administrateurs en terme de gestion et de représentation des grands réseaux hétérogènes. L'offre de logiciels de supervision et d'administration de réseaux est pléthorique et malheureusement encore trop souvent propriétaire. Tous ces logiciels disposent d'un module graphique de description d'une architecture matérielle et d'une base de données de stockage de ces architectures. C'est d'ailleurs autour d'une représentation commune et normalisée de ces ressources que se focalisent actuellement la recherche et les énergies. C'est pourquoi dans la section suivante, nous allons étudier les travaux de l'ISO/SMI et du DMTF, qui tentent de normaliser les attributs de description de chacune des ressources d'une architecture matérielle.

### *2.5.1. La normalisation ISO/SMI*

La norme SMI (*Structure of Management Information*), définie par l'ISO, décrit l'ensemble des ressources et des éléments de l'environnement des réseaux comme des objets. Un objet représente une vue abstraite des ressources et appartient à l'une des trois catégories suivantes :

- les ressources physiques (équipements, cartes électroniques, liaisons physiques, multiplexeurs, stations de travail, serveurs connectés en réseau);
- les éléments logiques et virtuels (les logiciels, les alarmes, etc.);
- les objets relatifs à l'environnement et à l'organisation (utilisateurs, fournisseurs, opérateurs, etc.).

Les objets sont caractérisés par des propriétés et des relations. Chaque objet à ses propres attributs, accessibles en lecture ou écriture via l'interface fournie par l'objet lui même, pour les besoins de gestion. Des opérations de gestion peuvent être appliquées à l'objet lui même ou à ses attributs.

La supervision des objets se fait en surveillant les comportements de l'objet suite aux opérations qui l'affectent et les notifications qu'il émet en fonction de l'occurrence d'événements particuliers. On utilise pour cela des classes d'objets, désignant un ensemble d'objets gérés et partageant les mêmes propriétés. La spécification des objets d'une classe est constituée de la définition de la classe. Une instance de la classe est un objet qui peut hériter de plusieurs classes et contenir d'autres objet.

La MIB (*Management Information Base*) d'un système désigne la base d'informations de gestion. Elle est réalisée par une base de données classique géré par un SGBD relationnel [Gardarin & al. 89] ou objets [Gardarin & al. 90 et Adiba & al. 93].

Malheureusement, ces normes de l'ISO ont été peu implémentées, pour des raisons de coût de développement et de difficultés de migration des architectures déjà existantes. Ce n'est pas le cas des normes basés sur SNMP (*Simple network Management Protocol*) [Stallings 93] et sur ses MIB complètement différentes de celles de l'OSI. Néanmoins, GDMO semble devenir l'instrument de la future convergence des deux normes.

### *2.5.2. Les travaux du DMTF*

La mise en place de ces normes reste lourde pour des petits réseaux non hétérogènes, c'est pourquoi, le DMTF *(Desktop Management Task Force)*, créé en 1992 et regroupant une douzaine de sociétés (telles qu'Apple, Compaq, Dell, Digital, HP, IBM, Intel, Microsoft, Novell, Sunsoft, Symantec) a défini une nouvelle architecture. L'architecture DMI *(Desktop Management Interface)* offre une interface standard entre les ressources gérées et le gestionnaire de supervision. Les données recueillies ne sont plus stockées dans des MIB, mais dans des fichiers de type MIF (*Management Information Format*). Il existe de nombreuses MIF pour la description du poste de travail (environ 200 attributs), pour les imprimantes, pour les adaptateurs réseaux, pour les grandes messageries, pour les logiciels, pour les serveurs, pour les périphériques de stockage. Les MIF pour les modems, les cartes sons, les composants PCMCIA, les agents SNMP et les alimentation électriques viendront plus tard.

La présentation des données issues de la MIF, sous une forme lisible et connue de tous, est à la charge de la couche CI *(Component Interface)*. Il faut ainsi, une CI par système d'exploitation. La remontée de ces informations vers un logiciel d'administration se fait grâce à la couche MI *(Management Interface)*. Entre le CI et le MI, on trouve une couche intermédiaire appelée SL *(Service layer)* qui s'assure que les données contenues dans la MIF sont à jour. Notons que l'X-Open travaille au développement d'une version Unix de DMI et que la jonction entre les MIB SNMP et les fichiers MIF sera réalisée par traduction de formats.

### *2.5.3. Discussion*

Tous les logiciels d'administration de réseaux utilisent désormais un outil graphique pour modéliser le réseau sous jacent. Mais un outil d'administration et de supervision est bien plus qu'une jolie interface graphique. C'est avant tout un gestionnaire de données récupérant et stockant les descriptions des ressources matérielles et leur état de fonctionnement. Il n'est donc pas étonnant que les normes de gestion de réseau à l'ISO soient tant débattues et si complètes (cf. [Lecerf & al. 93]).

La description de l'architecture matérielle est ensuite manipulée à travers des librairies de programmation (ou API), souvent propriétaires et résultant d'une volonté à la fois d'ouverture et de respect des normes actuelles. Les attributs associés aux objets gérés sont en général de très bas niveau et en nombre important (il en existe déjà plus de 600 dans certaines «normes»). D'autre part, la remontée des informations vers les stations

d'administration requiert un agent logiciel sur chaque machine et le choix des informations pertinentes à exploiter dans la MIB.

**Tableau 8:** Les logiciels d'administration de systèmes

| Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|
| Oui, formalisme textuel et outils graphiques | Non | Oui | Non, basée sur des normes | réseaux hybrides | Non |

### *2.5.4. Synthèse*

En conclusion, nous notons que comme pour les logiciels de simulation par des méthodes des objets, le niveau de détails dans la représentation de chaque élément de l'architecture matérielle est très au delà de nos besoins. De plus, on a pu déduire que les logiciels d'administration de systèmes et de réseaux :

- supportent de très nombreux protocoles de communication et reconnaissent de nombreux types d'équipements **hétérogènes** (ordinateur, pont, routeur, auto-commutateur, etc.) ;
- offrent un module de découverte automatique de l'architecture du réseau et des machines connectées ;
- stockent dans une base de données des descriptions de ressources ayant des attributs normalisés ;
- fournissent une interface graphique conviviale, gérant des descriptions hiérarchiques ;
- sont construits suivant une architecture ouverte intégrant des produits tiers complémentaires éventuels.

## 2.6. Conclusion

Les langages de programmation parallèle décrivent l'architecture matérielle de manière interne, ce qui empêche leur utilisation en tant que langage multi-fonction. Les langages de configuration, eux par contre, permettent de décrire non pas l'architecture de la machine, mais les machines sur lesquelles tel ou tel programme doit s'exécuter. Des informations précises sur l'architecture réelle de la machine ne sont pas fournies. Ce dernier point est, par contre, le point fort dans les logiciels de répartition de charge et d'administration de réseaux. L'utilisation de bases de données persistantes (les MIB), possédant de nombreux objets, qui eux mêmes sont caractérisés par des dizaines attributs est un frein considérable à leur exploitation. Enfin, les langages de simulation disposent d'interface graphiques et de langages hiérarchiques puissants, mais propriétaires et peu réutilisables. Un résumé de la comparaison de ces langages issue de notre état de l'art est présenté au Tableau 9.

Aucun des langages étudiés n'offre, à notre connaissance, toutes les fonctionnalités recherchées lors de cette étude et notamment l'ouverture vers des environnements variés. Il nous a donc semblé nécessaire de créer un langage de description d'architecture adapté à nos besoins. C'est pourquoi nous avons conçu le langage HADEL *(Hybrid Architecture DEscription Language)*. Il intègre certains concepts clefs qui ressortent de cet état de l'art, permettant d'en faire un langage extensible et réutilisable par d'autres applications. La description du langage HADEL, de ses propriétés et de sa mise en oeuvre sont traités dans le chapitre suivant.

**Tableau 9:** Comparaison des langages et des logiciels de description des architecture matérielle

| Description de l'architecture matérielle | Convivialité | Portabilité | Dynamicité | Extensibilité | Granularité | Réutilisabilité |
|---|---|---|---|---|---|---|
| **Langages de programmation parallèles** | Non | Oui | Non | Non | liée à la machine ou au processeur virtuel | Non |
| **Langages de configuration** | Oui | Oui | Oui | Oui | variable hiérarchie | Oui |
| **Logiciels de simulation** | Oui | Non | Non | Oui | variable hiérarchie | Oui |
| **logiciels de répartition de charge** | Oui | Oui | Oui | Non | Gros grain | Oui |
| **logiciels d'administration** | Oui | Non | Oui | Non | Réseaux hybrides | Non |

## 3. Hadel : un langage de description d'architectures matérielles Hybrides

Dans le paragraphe précédent, nous avons défini 6 critères indispensables à la description d'architectures matérielles hybrides. Ces six critères sont la convivialité, la portabilité, l'extensibilité, la granularité, la réutilisabilité et la dynamicité. Ces critères ne s'appliquent d'ailleurs que sur des langages et des outils qui se répartissent sur les trois grandes classes de problèmes que nous avons précédemment choisis à savoir :

- la description de l'architecture matérielle en vue du placement et du déploiement d'application réparties et/ou parallèles
- la gestion statique et dynamique d'une architecture de machines hétérogènes
- la simulation d'architectures matérielles.

N'ayant pas trouvé de langages et d'outils adaptés à nos besoins, dans le contexte de notre étude, nous avons proposé le langage HADEL, ainsi que les outils et les traitements associés.

Il nous a semblé primordial de disposer d'un formalisme à la fois simple et puissant. La gestion de la hiérarchie a eu pour conséquence de simplifier les descriptions, tout en offrant une prise en compte de niveaux de description distincts. Néanmoins, dans le cadre de notre méthode MEDEVER, la gestion de la hiérarchie ne devait pas être trop compliquée. Enfin, nous avons suivi l'exemple du langage Sysl [Sommerville & al. 92] en ce qui concerne l'indépendance vis à vis des systèmes d'exploitation et des architectures.

**C'est pourquoi, HADEL (*Hybrid hArdware DEscription Language*) est un langage de description d'architectures matérielles hybrides qui dispose d'une hiérarchie de description à deux niveaux.**

Le niveau Macro décrit des architectures à gros grain de machines, à savoir des stations de travail et des équipements réseaux qui les relient. Le niveau Micro décrit des machines multi-processeurs et les éléments qui la constituent (processeurs, mémoire, ports d'entrée sorties, etc.). Les modèles sont créés via un éditeur graphique ou textuel, puis sont ensuite vérifiés syntaxiquement et sémantiquement, avant d'être stockés au sein de bibliothèques. Il est possible de rechercher des architectures matérielles déjà existantes, en utilisant un module de requête créé par nos soins.

Dans la suite de ce paragraphe, la section 1 décrit les différents objets et attributs utilisés dans chacun des deux niveaux de hiérarchie du langage HADEL. Nous montrons ensuite dans la section 2, que le langage HADEL respecte les six critères imposés précédemment. Enfin, la section 3 conclut et présente quelques perspectives concernant l'évolution du langage HADEL.

## 3.1. Description du langage HADEL

Nous présentons dans un premier temps les deux niveaux de hiérarchie du langage HADEL et les objets et attributs qui leurs sont associés. Puis, dans un second temps, nous traitons des relations entre ces niveaux de hiérarchie.

### *3.1.1. Les objets de base définis dans HADEL*

La description de l'architecture hybride est réalisée avec deux types d'objets, les H-Machines et les H-Liens.

Les *H-Machines* décrivent les stations de travail mono ou multi-processeurs et les *H-Liens* représentent les liens physiques entre les H-Machines. Les H-liens ne sont pas forcément point à point.

Ces objets sont utilisés pour construire un graphe où les H-Machines sont les noeuds et les H-Liens les arcs. Ces objets possèdent des attributs dont la description peut être graphique ou textuelle.

La description d'une architecture matérielle s'effectue à deux niveaux :

- *Macro* : chaque machine représente une entité indivisible. Ce niveau est intéressant pour décrire des réseaux de machines faiblement couplées (par exemple, un réseau local de stations de travail);
- *Micro* : les machines sont décomposées de manière plus fine; on décrit alors des processeurs, des mémoires, des ports vers l'extérieur et les relations qui les lient. Ce niveau est intéressant pour décrire un réseau de machines fortement couplées (par exemple une machine multi-processeurs).

La description Macro est utile pour décrire une architecture matérielle cible d'une exécution répartie. Cette description est alors utilisée par des outils de répartition de charge, d'administration simple de réseaux et par des logiciels de simulation de trafic. Le niveau Micro a été créé pour tenir compte des architectures hybrides et intègre un niveau de description très proche des logiciels de simulation actuels.

**L'un des intérêts de cette description est qu'elle donne le moyen d'unifier les techniques de placement et d'administration des architectures parallèles et réparties**.

Une machine multi-processeur est donc gérée au niveau Macro. Elle est considérée comme un tout, ou lorsque cette description ne suffit pas est caractérisée de manière plus fine. Ceci est alors le cas, par exemple, pour détecter la panne d'une processeur de la machine, pour calculer la puissance théorique d'une machine, pour simuler une machine parallèle dans une configuration particulière. Notre problématique ne gère que deux niveaux de description qui ne sont ni exclusifs, ni obligatoires. C'est aussi pourquoi, nous n'avons pas voulu généraliser la hiérarchie, ni imposer que chaque machine soit décrite de manière hiérarchique.

### *3.1.2. Le formalisme Macro : modéliser des réseaux de machines*

Au niveau Macro, on rencontre des H-Machines reliées entre elles par des H-Liens.

*Un exemple de description d'une architecture matérielle hybride*

La description d'une architecture peut-être réalisée avec HADEL de manière textuelle ou de manière graphique. Pour couvrir notre exemple, nous utilisons le formalisme graphique, disponible dans l'éditeur de graphe Macao [Macao 97].

La Figure 5 représente la description graphique d'un réseau de trois machines connectées par un brin Ethernet à 10 Méga-Octets/secondes. Ce réseau comporte deux machines monoprocesseur (Héphaïstos et Elios) ainsi qu'un prototype de machine multi-processeur à base de carte PC (PC_parallele). Tous les attributs ne sont pas visibles sur la figure, pour des raisons de lisibilité.

La description d'une machine monoprocesseur comporte les informations suivantes : le nom de la machine, sa puissance en MIPS, le type de la machine, la taille de la mémoire centrale et d'autres informations paramétrables (comme la charge moyenne). Chaque lien est caractérisé par son type (Ethernet par exemple) et sa bande passante. La description d'une machine multi-processeur se ramène en revanche à une "boîte" qui peut "s'ouvrir" et contenir des informations de niveau Micro.

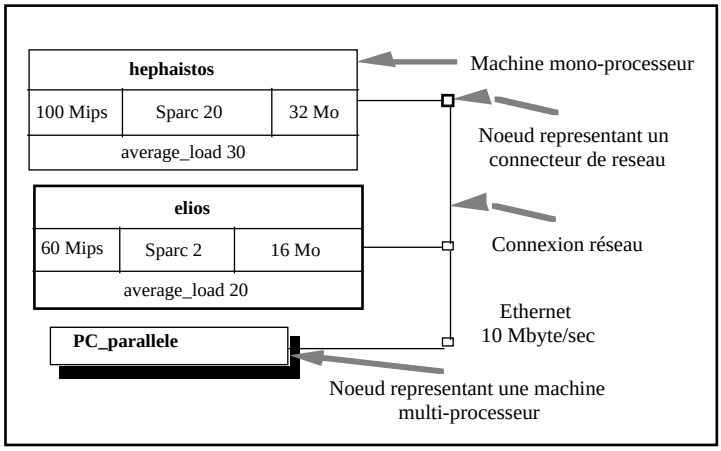



**Figure 5 :** Description Macro d'un réseau hybride avec HADEL

*Attributs de description d'une H-Machine au niveau Macro*

La description d'une H-Machine mono ou multi-processeur au niveau Macro est composé de trois types d'information : les information générales, les paramètres architecturaux et les paramètres de puissance (cf. Figure 6).

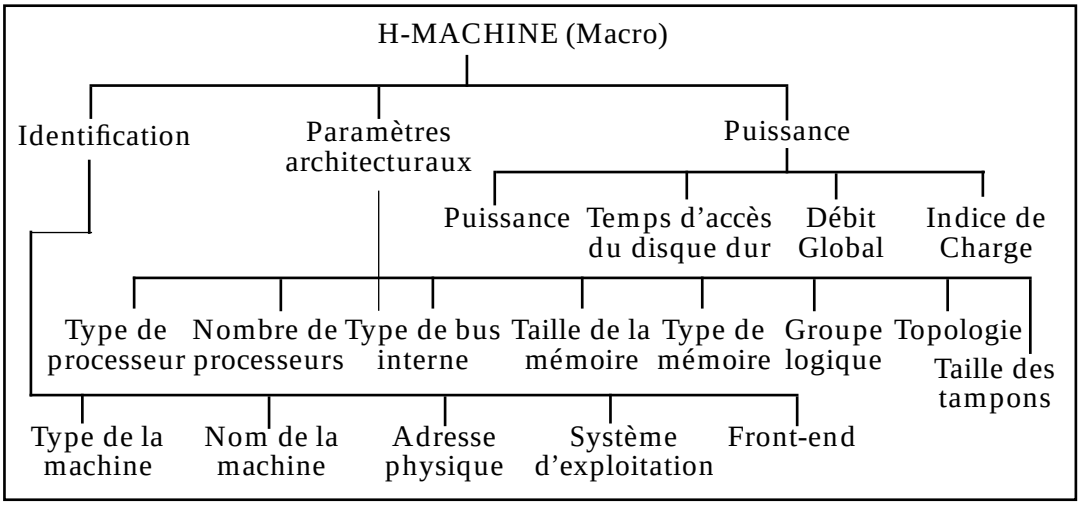



**Figure 6 :** Propriétés au niveau Macro d'une H-Machine

Chacun des attributs présentés à un intérêt global ou spécifique pour les types d'applications que nous avons déjà définis :

- l'administration de réseaux hybrides (mot clef Administration) ;
- la description de l'architecture à des fins de documentation (mot clef Documenta-

tion) ;

- la répartition de charge et le placement de programmes (mot clef Répartition) ;
- la simulation des architectures matérielles (mot clef Simulation).

Les descriptions détaillées des attributs définis pour chaque type d'informations en fonction des quatre critères sont présentés dans les tableaux suivants.

**Tableau 10:** Description des informations d'identification d'une H-Machine (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Type de la machine | X | X | X | | Ce paramètre est obligatoire, car les applications sont généralement compilées pour une architecture cible précise. Grâce à ce champ, il est possible de créer des H-Machines de type serveur, pont, routeur ou passerelle ou des H-machines mixtes (à la fois station de travail et disposant de capacités de routage). |
| Nom de la machine | X | X | X | X | La plupart du temps, les utilisateurs d'une machine ne connaissent pas son adresse, mais uniquement son nom. La récupération automatique de l'adresse physique de la station en fonction de son nom est en général possible grâce à un serveur de noms disponible sur le réseau. |
| Adresse de la machine | X | X | X | | Il est possible de spécifier l'adresse de la machine que l'on désire décrire. Il est conseillé de donner l'adresse IP de la machine, mais ce n'est pas obligatoire. Dans le cas où la H-machine est une passerelle, il est possible de spécifier plusieurs adresses. |
| OS utilisé | X | X | X | | Ce paramètre est utile lorsqu'une application utilise les spécificités d'un système d'exploitation particulier. Ainsi, par exemple, la programmation de processus légers sur une architecture multi-processeurs symétriques n'est disponible sur Solaris, le système d'exploitation de Sun, qu'à partir de la version 4.1.3. |
| Front-end | X | X | X | | Certaines machines sont accessibles uniquement par une machine qui stocke les programmes et la pilote à distance. Il est donc nécessaire de savoir ou elle se trouve. |

**Tableau 11:** Description des paramètres architecturaux d'une H-Machine (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Type de processeur | X | X | X | | Ce paramètre est important car il permet de choisir le lieu d'exécution d'un programme compilé pour tel ou tel type de processeur. Ce paramètre est aussi nécessaire pour des réseaux de machines de type PC, dont l'architecture globale est la même et qui ne se différencie souvent que par le type du processeur. |
| Type de bus interne | X | X | X | | Le type de bus interne donne des informations très intéressantes sur les capacités en terme de débit et d'interconnexion. Les bus peuvent-être de type PCI, IDE, SCSI, etc. |
| Nombre de processeurs | X | X | X | X | Si une machine est multi-processeur, au niveau Macro, seul importe le nombre de processeurs présents. |
| Type de mémoire | X | X | X | X | Mémoire locale, partagée ou répartie. |
| Taille de la mémoire centrale | X | X | X | | La taille de la mémoire centrale influe de manière importante sur les performances et les temps d'exécution des programmes sur une machine |

**Tableau 11:** Description des paramètres architecturaux d'une H-Machine (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Groupe(s) logique(s) | X | X | X | | On peut assigner un numéro de groupe logique à chaque H-Machine sur le réseau, en fonction de paramètres propres à l'utilisateur. |
| Topologie d'interconnexion | | X | X | X | Dans le cas où la H-Machine est une machine multi-processeurs, il est important de décrire la topologie d'interconnexion entre processeurs. Elle est définie statiquement ou dynamiquement (grille, hypercube, etc.). |
| Taille des tampons d'entrées-sorties | | X | X | X | Les performances des communications sont liées directement à la taille des tampons d'entrées-sorties, à la manière dont le système les gère et dont les applications y accèdent. |

**Tableau 12:** Description des informations de puissance d'une H-Machine (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Puissance théorique (en MIPS) | | X | X | X | L'utilisateur peut vouloir tester la vitesse d'exécution de son programme en fonction de la puissance des machines supports de l'exécution. Dans le cas d'une machine multi-processeurs, c'est la puissance cumulée de tous les processeurs. |
| Temps d'accès moyen au disque | | X | X | X | Cet indicateur indique le temps d'accès moyen aux données sur le disque dur de la H-machine. |
| Débit global | X | X | X | X | Cette valeur permet d'apprécier globalement les performances concernant les entrées-sorties de la station. |
| Indice de charge | X | X | X | X | Indique le niveau de charge d'une H-machine. |

*Description d'un H-Lien*

Chaque lien reliant une ou plusieurs H-Machine, appelé H-Lien, est caractérisé par des attributs définis dans la Figure 7 et commentés dans le Tableau 14. Cette liste n'étant pas obligatoirement exhaustive, il est possible de spécifier d'autres attributs à ceux qui existent déjà. L'utilisateur peut donc étendre notre modèle en rajoutant lui même ses propres noms et valeurs d'attributs.

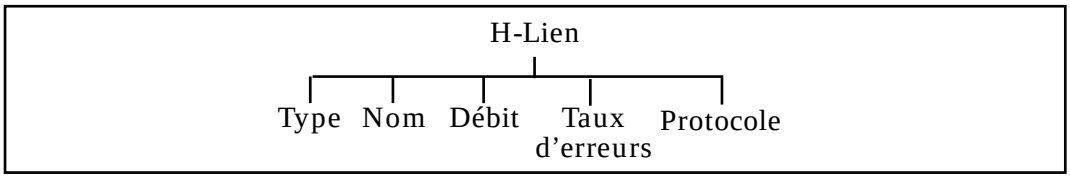



**Figure 7 :** Propriétés au niveau Macro d'un H-Lien

**Tableau 13:** Description des attributs d'un H-Lien (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Type | | X | X | X | Il est indispensable de savoir de quel type de support on dispose, car un bus ethernet ne possède pas le même débit qu'un câble série reliant deux PC. |
| Nom | | X | | X | Ce paramètre permet de gérer des multi-réseaux en donnant un nom logique à chaque segment du réseau. |

**Tableau 13:** Description des attributs d'un H-Lien (Macro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Débit | X | X | X | X | Débit Théorique, Débit Courant Mesuré ou Débit Moyen Constaté. |
| Taux d'erreurs | X | X | X | X | Ce paramètre permet d'adapter éventuellement le protocole au taux d'erreurs constaté. |
| Protocole | X | X | X | X | Pour les machines multi-processeurs les protocoles les plus connus sont *circuit switched*, *store and forward* et *worm-hole*. Pour les machines mono-processeur, on parle plutôt de TCP-IP |

### *3.1.3. Le formalisme Micro : modéliser une machine multi-processeurs*

Le niveau Micro hérite des propriétés du niveau Macro et offre de plus un niveau de description interne de la H-Machine.

*Exemple de description*

Si on reprend l'exemple de la Figure 5, la description de la machine multi-processeur «PC_parallele» est donnée à la Figure 8. Cette description comporte un banc mémoire connecté à chacun des processeurs, qui sont eux mêmes connectés à un même bus disposant d'un port de communication vers l'extérieur.

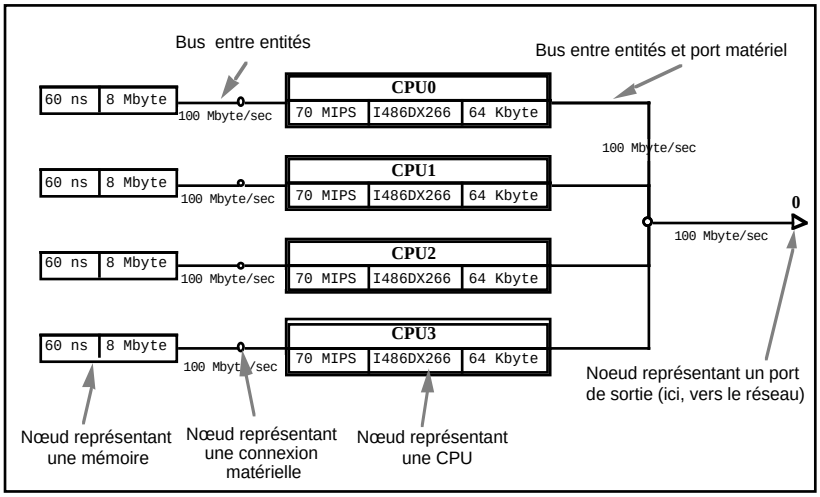



**Figure 8 :** Description Micro de la machine multi-processeur pc_parallele avec HADEL

*Description complète*

Les différents attributs associés à une H-MACHINE au niveau Micro sont présentés dans la Figure 9.

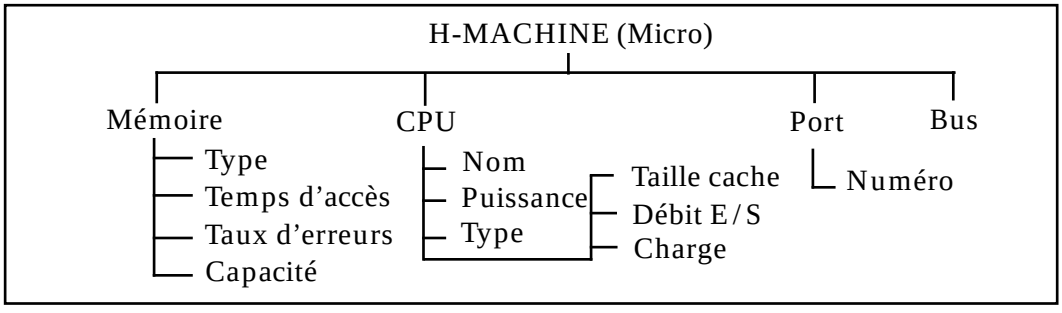



**Figure 9 :** Propriétés au niveau Micro d'une H-Machine

Une description au niveau Micro est donc composée de quatre types d'objets : les mémoires (cf. Tableau 14), les CPU (cf. Tableau 14), les ports d'entrées sorties et les bus. Un port est caractérisé par son numéro et permet des connexions logiques avec le niveau Macro. Un bus est caractérisé par une bande passante et met en relation les objets mémoire, les CPU et les ports d'entrées-sorties.

**Tableau 14:** Description des attributs d'un objet mémoire (Micro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Type | X | X | X | X | Locale, partagée ou répartie. |
| Temps d'accès | | X | | X | Généralement exprimé en nano seconde, il indique le temps moyen nécessaire pour la lecture d'une donnée. |
| Taux d'erreurs | | | | X | Il est donné en nombre d'erreurs par seconde. |
| Capacité | X | X | X | X | Elle est exprimée en Méga-Octet. |

**Tableau 15:** Description des attributs d'un objet CPU (Micro)

| **Attributs** | Administration | Documentation | Répartition | Simulation | **Description** |
|---|---|---|---|---|---|
| Famille | X | X | X | X | Le nom du fabriquant. |
| Puissance | X | X | X | X | Généralement exprimée en MIPS. |
| Type | X | x | X | X | Modèle d'architecture (Pentium, 68xxx). |
| Taille de son cache interne | | X | | X | Elle est exprimée en Kilo Octet. |
| Débit en L/E | | | | X | Surtout pour la simulation, c'est l'équivalent du temps de service d'une file d'attente. |
| Charge Moyenne | X | | X | X | En pourcentage de sa capacité maximale. |

## 3.2. Conclusion

HADEL est un langage de description d'architectures matérielles hybrides. C'est pourquoi, il utilise un formalisme de description à deux niveaux : le niveau Macro (réseaux de machines interconnectées) et le niveau Micro (description d'une machine multi-processeur). Chacun de ces niveaux possède des objets et des attributs qui lui sont propres et qui permettent la réutilisation d'HADEL au sein de nombreux autres logiciels ou environnements. HADEL dispose d'un langage de description textuel et d'un formalisme graphique sous Macao [Macao 97] (cf. BNF du langage HADEL). Les bibliothèques de composants sont accessibles via des API et permettent la construction de requêtes complexes sur la base de données de matériels ou de topologie.

Le langage HADEL mis en oeuvre vérifie donc chacun des six critères que nous avions évoqués lors de notre état de l'art à savoir :

- *la convivialité* : HADEL dispose d'une description graphique et d'une description textuelle basée sur un langage déclaratif très simple (cf. BNF du langage HADEL en Annexe B) ;
- *la portabilité* : HADEL est un langage qui décrit des objets, leurs attributs et les interconnexions entre ces objets. Cette description n'est pas liée à un environnement spécifique ;

- *la dynamicité* : la description d'une architecture avec HADEL se fait en trois phases. Lors de la phase de description, l'utilisateur en utilisant (ou non) des éléments de bibliothèques construit son architecture matérielle et la topologie d'interconnexion qu'il désire. Lors de la phase suivante, l'architecture ainsi décrite est vérifiée, puis validée. La dernière phase assure la correspondance entre l'architecture décrite et le matériel disponible réellement sur le réseau. HADEL a été conçu pour gérer la dynamicité de la phase de correspondance, aussi bien au sein des informations disponibles dans les bibliothèques de composants matériels, qu'au sein des attributs qui existent au niveau Macro et Micro.
- *l'extensibilité* : il est possible de rajouter des attributs à ceux déjà définis pour chaque objet, grâce à un champ extension, prévu à cet effet ;
- *la granularité* : HADEL décrit une H-MACHINE à deux niveaux : au niveau Macro et au niveau Micro. Ces deux types de granularité sont adaptés à la description de réseaux hybrides et aux traitements que nous souhaitons réaliser ;
- *la réutilisabilité* : la gestion de librairies de topologies et de matériels permettent de stocker les caractéristiques d'objets de description. Il est ainsi possible de rajouter des objets à la base ou de l'interroger à tout moment.

La description de la mise en oeuvre du langage HADEL est présentée dans le chapitre 6.

## 3.3. Perspectives

Enfin, au niveau des perspectives, nous nous sommes intéressés à trois technologies récentes qui pourraient modifier la conception, la simulation, l'utilisation et l'administration d'architectures matérielles. Ces trois technologies sont les réseaux virtuels, les réseaux haut débits (tels qu'ATM) et les interfaces graphiques 3D. Nous présentons dans la suite, les synergies à développer entre ces technologies et notre langage HADEL.

### *3.3.1. HADEL et les réseaux virtuels*

La notion de réseau virtuel est directement hériteé de la technologie ATM. Le but d'un réseau virtuel est de regrouper physiquement, selon une topologie à préciser, un ensemble de stations communiquant de manière intense. Le réseau virtuel est actuellement bien adapté aux réseaux locaux et utilise des équipements réseaux spécifiques (en général ce sont des commutateurs ATM). Or, la plupart de ces équipements réseaux sont configurables par logiciel. D'ou l'idée de permettre à HADEL de devenir un langage de configuration de réseau virtuel. Ceci permettrait alors de faire, non plus de la simulation, mais de l'émulation d'architectures de machines complexes, non disponibles à un moment donné, sur un réseau ATM haut débit.

Par ce procédé, on tente d'homogénéiser l'accès aux machines mono et multi-processeur, en permettant la reconfiguration dynamique dans les deux cas. En utilisant les techniques de réseaux virtuels disponibles sur les équipements réseaux récents (commutateurs TRT par exemple), il est dorénavant possible d'offrir une liaison réellement à 10 Mbps (voire 100 et 155 Mbps avec Bagnet [Johnson 95]) en Full duplex, entre chaque station et de se constituer ainsi un réseau d'interconnexion à un niveau Macro. Les politiques de gestion des architectures Micro ou Macro deviennent donc identiques. Les réseaux hybrides se ramènent alors dans ce cas, à des réseaux hiérarchiques ayant des débits et des interfaces de même ordre de grandeur (on parle alors de multi-réseaux).

### *3.3.2. ATM et HADEL*

Malheureusement, l'état actuel des normes ne nous permet pas d'utiliser un langage unique de configuration et d'inter-connexion des équipements ATM encore trop propriétaires. Il faudrait donc réécrire pour chaque équipement un pilote spécifique. Nous pensons que d'ici à quelques années les architectures matérielles seront adaptables aux applications, en terme de débit, de réseau d'interconnexion et de routage. Un langage intégré tel qu'HADEL couvrirait alors les aspects de description et de gestion des configurations,

de gestion de parcs sommaire et de validation des architectures à tester. Enfin, grâce à la norme *Lan Emulation*, l'émulation d'un réseau local sur un réseau ATM offrira des performances supérieures en terme de débit à celles des réseaux locaux actuels (Ethernet et Token Ring à 10 Mbps).

### *3.3.3. HADEL et les interfaces graphiques 3D*

L'émergence de stations de travail peu chères et puissantes conjuguées avec l'essor des cartes graphiques ont conduit les concepteurs de logiciels de description d'architecture à tirer partie de la 3D. C'est notamment le cas du logiciel d'administration *Unicenter*, de Computer Associates, qui offre une interface graphique en trois dimensions. Sur internet, la navigation dans des espaces à 3 dimensions est basée sur la norme VRML [VRML 97]. Partant de ce constat, il serait intéressant de pouvoir générer des descriptions VRML directement à partir de HADEL. L'utilisation d'un navigateur du marche affranchirait alors les utilisateurs d'un outil graphique spécifique.

## 4. Références bibliographiques


[Ada 83] U.S. Department of Defense, Ada joint Program Office, "Reference Manual for the Ada Programming Language", ANSI / MIL-STD 1815A, 1983 (ISO 8652-1987).

[Ada 95] U.S. Department of Defense, Ada joint Program Office, "Reference Manual for the Ada Programming Language", ANSI / ISO / IEC 8652-1995.

[Adamo & al. 92] J.M. Adamo & C. Bonello, «The C_NET Programming Environment : An Overview», LNCS 634, Proceedings of the International Conference CONPAR 92, April 1992.

[Adiba & al. 93] M. Adiba & C. Collet, «Objets et bases de données : le SGBD O2», Hermès Eds., Juin 1993.

[Alimi 92] R. alimi, «Evaluation du logiciel OPNET», Rapport Interne, Laboratoire MASI, Univ. Pierre et Marie Curie, 4 place Jussieu, 75252 PARIS cedex 05.

[Allen 80] A.O. Allen, «Queuing Models of Computer Systems», IEEE Computer, pp. 13-24, April 1980.

[Anelli 92] P. Anelli, «Performance des réseaux informatiques: vers une interface utilisateur pour le simulateur NESSY dans un environnement orienté objet», Thèse de doctorat de l'Université Pierre et Marie Curie, juin 1992.

[Bacceli & al. 89] F. Baccelli & A. Makowski, «Queuing Models for Systems with Synchronisation Constraints», Proceedings of the IEEE, 77 (1).

[Baskett & al. 75] F. Baskett, K.M. Chandy, R.R. Muntz & F.G. Palacios, «Open, Closed and Mixed Networks of Queues with Different Classes of Customers», Journal of ACM, Vol 22 (2), April 1975, pp 248-260.

[Bergsten & al. 93] B. Bergsten, M. Couprie & P. Valduriez, «Overview of Parallel Architectures for Databases», Computer Journal, Vol. 36 (8), 1993, pp.734-740.

[Bréant & al. 93] Bréant F & Peyre J.F., «Occam Prototyping of Massively Parallel applications from Colored Petri-Nets», 7th IEEE International Parallel Processing Symposium, 1993.

[Bull 91] BULL, «Distributed Computing Model», Description générale des spécification, 1991.

[Burns & al. 90] G. Burns, V. Radiya, V. Daoud & R. Machiraju, «All about TROLLIUS», OCCAM User Group Newsletter, No 13, July 1990.

[Burns 92] C. Burns, «REE - A Requierement Engineering Environment for Analyzing and Validating Software and System Requierements», 4th IEEE International Workshop on Rapid System prototyping, Research Triangle Park, North Carolina, USA, June 28-30, 1993, pp 188-191.

[Chassin D. K. 95] Chassin de Kergommeaux, «Environnement d'exécutions de programmes parallèles», Calculateurs parallèles, Vol 7 (2), 1995, Hermès, France.

[Chrichlow 88] J. M. Crichlow, «An Introduction to distributed and parallel programming», Prentice Hall International, 1988.

[Cubaud & al. 92] P. Cubaud & N. Pekergin, «Modèles et outils pour la prédiction de performance des systèmes informatiques parallèle», Thèse de doctorat de l'université René Descartes, Paris, France, 1992.

[Damm 96] G. Damm, «Conception et Réalisation d'une Bibliothèque de Modèles d'Architectures Parallèles», Thèse de doctorat de l'Université Pierre et Marie Curie, 13 Décembre 1996.

[Dean & al. 92] G. Dean, D. Hutchinson, T. Rodden & I. Sommerville, «Cooperation and Configuration within Distributed Systems Management», International Workshop on Configurable Distributed System, London, 25-27 March 1992, pp 80-89.

[Dewitt & al. 92] D.J. Dewitt & J. Gray, «Parallel Database Systems: The Future of High-Performance Database Systems», Communication of the ACM, Vol. 35 (6), June 1992, pp. 85-98.

[Dulay 90] N. Dulay, «A Configuration Language for Distributed programming», Ph.D. Thesis, Dept. of Computing, Imperial College, February 1990.

[Folliot & al. 95] B. Folliot, K. Foughali & P. Sens, «Placement d'applications parallèles dans les réseaux de station de travail : l'expérience GATOSTAR», Journées de recherche sur le placement dynamique et la répartition de charge: Application aux systèmes répartis et parallèles, Université Pierre et Marie Curie, Paris, Mai 1995, pp 75-78.

[Flynn 72] M.J. Flynn, "Some computer organizations and their effectiveness", IEEE Transactions on Computers, Vol 21 (9), 1972.

[Gardarin & al. 89] G. Gardarin & G. Valduriez, «Relational Databases and Knowledge Bases», Addison Wesley Eds., 1989.

[Gardarin & al. 90] G. Gardarin & G. Valduriez, «SGBD avancés: Bases de données objets, déductives, réparties», Eyrolles Eds., 1989.

[Gaudiot & al. 92] J.L. Gaudiot & D.K. Yoon, «A comparative study of Parallel Programming Languages: The Salishan Problem", J.T. Feo Editor, Elsevier Science, 1992, pp 217-262.

[Gelenbe & al. 77] E. Gelenbe & G. Pujolle, «A Diffusion Model for Multiple Class Queuing Networks», Rapport de Recherche INRIA, No 242, août 1977.

[Gelenbe & al. 80] E. Gelenbe & I. Mitrani, «Analysis and Synthesis of Computer Systems», Academic Press Eds., 1980.

[Ghernaouti 94] Ghernaouti-Hélie, «Client /serveur : les outils du traitement coopératif», Masson Eds., Chapitre 7, pp 95-111.

[Hagersten & al. 92] E. Hagersten, A. Landin & S. Haridi, «DDM - A Cache-Only Memory Architecture», Computer, September 1992, pp. 44-54.

[Hémery 94] F. Hémery, «Etude de la répartition dynamique d'activités sur architectures décentralisées», Laboratoire d'Informatique Fondamentale de Lille, Thèse de l'Université des Sciences et Techniques de Lille, No d'ordre : 1339, 1994, 222 pages.

[Hoare 88] C. Hoare, «Occam 2 Reference Manual», Prentice Hall, 1988.

[HPF 93] Forum HPF, «High Performance Fortran Language Specification», Technical Report v1.0, Rice University, May 1993, <http://www.erc.msstate.edu/hpff/home.html>.

[HPF 94] Koelbel, Loveman, Schreiber, Steele & Zosel, «The High Performance Fortran Handbook», MIT Press, 1994.

[Ibbett 92] R.N. Ibbett, «Parallel Computer Architecture: Where Next ?», Transputers'92, M. Becker & al. Eds., IOS Press, pp. 364-380.

[Inmos 88] Inmos, «Transputer Development System», Prentice Hall Eds., 1988.

[Jensen 92] K. Jensen, "Coloured Petri Net : Basic concepts, analysis methods and practical use, Volume 1 : Basic Concepts", EATC Monographs on Theorical Computer Science, Springer Verlag.

[Johnson 95] M.J. Johnson, «Experiences With The Bay Area Gigabit Network Testbed», Proc. of the V$^{th}$ IEEE Workshop on Future Trends of Distributed Computing Systems, Cheju Island, Korea, August 28-30 1995, pp. 26-32.

[Kleinrock 75] L. Kleinrock, «Queuing Systems, Vol 1 : Theory», Wiley-Intersciences Eds., 1975.

[Kurose & al. 88] J.F. Kurose & H.T. Moufta, «Computer-Aided Modeling, Analysis, and Design of Communication Networks», IEEE Journal on selected areas in Communications, Vol 6 (1), January 1988, pp 130-145.

[Leblanc & al. 82] R. J. LeBlanc & A. B. Maccabe, «The design of a programming language based on connectivity network», Proceedings of the 3rd International Conference of Distributed Computing System, 1982, pp 532-541.

[Lecerf & al. 93] C. Lecerf & D. Chomel, «les normes de gestion à l'ISO», Collection Technique et Scientifique des Télécommunications, Masson Eds., 1993.

[Macao 97] <http://www-masi.ibp.fr/macao>.

[Magee & al. 89] J. Magee, J. Kramer & M. Sloman, «Constructing Distributed System in CONIC», IEEE Transactions on Software Engineering, SE-15(6), June 1989.

[Magee & al. 92] J. Magee, N. Dulay & J. Kramer, «Structuring Parallel and Distributed Programs», International Workshop on Configurable Distributed System, London, 25-27 March 1992, pp 102-117.

[MIL 91] MIL 3 Inc., «OPNET (OPtimised Network Engineering Tool), M Version Product description», 1991.

[Nachef 93] A. Nachef, «Modélisation des systèmes distribués», Technique et Science Informatique, Vol. 12 (2), 1993, pp. 163-192.

[Neuse & al. 81] D. Neuse & K. Chandy, «SCAT: A Heuristic Algorithm for Queuing networks on Computing Systems», ACM Sigmetrics, Vol 10 (1), 1981, pp 59-79.

[Nichols 87] D. Nichols, «Using idle Workstations in a Shared Computing Environment», Proceedings of the 11$^{th}$ ACM Symposium on Operating Systems Principles», November 1987, pp 5-12.

[Pekergin 91] N. Pekerkin, «Nouveaux modèles d'évaluation de systèmes parallèles», Thèse de l'Université René Descartes, Paris, France, 1991.

[Pooley 91] R. Pooley, «The Integrated Modelling Support Environment: a new generation of performance modeling tool», Proceedings of the Modelisation Techniques and Tools for Computing Performance Evaluation, Torino, Italy, 1991.

[Potier 86] D. Potier, «The Modeling Package QNAP2 and Applications to Computer Networks Simulation», Computer Networks and Simulation, Tome III, Schoemaker S. Eds., North Holland, 1986.

[Reiser & al. 80] M. Reiser & S.S. lavenberg, «Mean Value Analysis of Closed Multichain Queuing Networks», Journal of the ACM, Vol 22 (2), April 1980, pp 313-322.

[Sansonnet & al. 91] J.P. Sansonnet & C. Germain-Renaud, «Les ordinateurs massivement parallèles», collection 2ai, Armand Colin Eds., 1991, 137 pages.

[Simulog 88] Simulog (Eds.), «Manuel de référence de QNAP2», 1988.

[Sommerville & al. 92] I. Sommerville & R. Thomson, «Configuration specification using a system structure language», International Workshop on Configurable Distributed System, London, 25-27 March 1992, pp 80-89.

[Soto 90] M. Soto, «NESSY (NEtwork Simulation SYstem): conception et implémentation», Thèse de doctorat de l'Université Pierre et Marie Curie, Laboratoire MASI, 4 place Jussieu, 75252 Paris cedex 05, 1990.

[Stallings 93] W. Stalling, «SNMP, SNMPv2 and CMIP: the Practical Guide to Network-Management Standards», Addison Wesley Eds., 1993.

[Stonebraker 86] M. Stonebraker, «The case for Shared-Nothing», IEEE Data Engineering Bulletin, Vol. 9 (1), 1986.

[Toutain 91] L. Toutain, «SAMSON: un Simulateur pour Systèmes Répartis et Temps-Réel», Thèse de doctorat de l'Université du Havre, 1991.

[TM 88] Thinking Machines Co., «C* Reference Manual», Version 4.3, May 1988.

[VRML 97] Mark D. Pesce, VRML Architecture Group, <http://vag.vrml.org/>.
VRML 2.0, http://vrml.sgi.com/worldls/index.html

[Zhang & al. 95] K. Zang & G. Marwaha, «Visputer: A Graphical Visualisation Tool for Parallel Programming», The Computer Journal, Vol. 38 (8), 1995, pp. 658-669.

[Zhou 88] S. Zhou, «A Trace-Driven simulation Study of Dynamic Load Balancing», IEEE Transactions on Software engineering, Vol 14 (9), September 1988, pp 1327-1341.

[Zhou & al. 93] S. Zhou, Z. Zheng, J. Wang & P. Delisle, «UTOPIA: A load Sharing Facility for Large, Heterogeneous Distributed Computer Systems», Software Practice and Experience, Vol. 23 (12), December 1993, pp. 1305-1336.

[Zitterbart 90] M. Zitterbart, «Monitoring and Debugging Transputer Networks with NETMON-II», Proceedings of the International Conference CONPAR 90, Zûrich, 1990.

[Zomaya 96] A. Zomaya, «Parallel Computing, Paradidgm and Applications», Thomson Computer Press, 1996.