Conclusion Générale

«L'interprétation, c'est la patience du sens pour renoncer à la rage de vouloir conclure» (Flaubert)

Nos travaux s'inscrivent dans le cadre d'une démarche de conception et d'exécution d'applications client-serveur. Cette démarche est basée sur le principe de la séparation des préoccupations [Hürsch & al. 95], qui différencie le noyau fonctionnel d'une application de ses facettes techniques et de sa plate-forme d'exécution. On isole alors les algorithmes principaux des médiateurs chargés de les mettre en oeuvre dans un contexte précis (mise en oeuvre centralisée, parallèle et/ou répartie par exemple).

Nous modélisons alors une application client-serveur, comme un ensemble de composants logiciels coopérants. Ces composants logiciels sont des gabarits de conception, décrivant, via un style architectural, une logique applicative. La mise en relation de ces composants logiciels (communication, synchronisation, coopération) entraîne la construction d'une architecture logicielle. Cette architecture logicielle est utilisée pour valider des propriétés de construction et d'utilisation des composants, mais aussi pour générer automatiquement leurs squelettes d'implémentation. La mise en relation de ces composants étant alors fonction de la granularité des composants logiciels, du niveau d'interopérabilité qu'ils offrent et des médiateurs utilisés. Enfin, lors de l'exécution, cette architecture logicielle doit être adaptée à des architectures matérielles mixant des réseaux de stations de travail et des machines parallèles. Le choix des machines formant cette architecture matérielle hybride est réalisée de manière automatique ou manuelle en fonction des besoins.

1. MEDEVER et sa mise en oeuvre

Nous avons proposé la méthode MEDEVER pour intégrer et homogénéiser les approches de développement d'architectures logicielles et les technologies médiateurs actuelles. MEDEVER adopte un point de vue architectural pour la gestion du cycle de vie des applications client-serveur. C'est pourquoi, MEDEVER s'appuie sur les deux langages de description d'architecture HADEL et VODEL-D (cf. Figure 81) et qu'elle est découpée en quatre phases que nous résumons dans la suite de cette section.

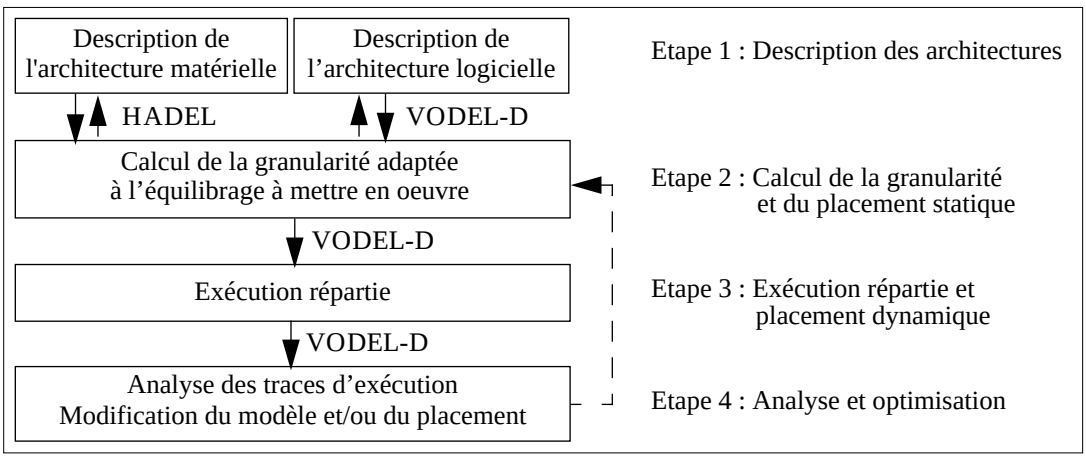

Figure 81 : La méthode MEDEVER et les langages supports de la méthode

1.1. Description des architectures

Pour réaliser le placement statique ou dynamique d'applications client-serveur sur des architectures hybrides, il est à notre avis indispensable de disposer de la description à la fois de l'architecture logicielle et matérielle. Ces descriptions, bien qu'ayant une finalité opérationnelle, doivent être adaptées à la méthode MEDEVER et à ses objectifs. N'ayant pas trouvé de langages de description se conformant à nos besoins, nous avons dû en définir un pour chaque type d'architecture.

1.1.1. Description des architectures matérielles hybrides : HADEL

HADEL est un langage de description d'architectures matérielles hybrides, c'est à dire composées de machine mono et multiprocesseurs. C'est pourquoi, il est fondé sur un modèle hiérarchique à deux niveaux. Le premier niveau, appelé niveau **Macro**, est dédié à la représentation de toutes les machines et de leur topologie d'interconnexion. Le second niveau, appelé niveau **Micro**, est directement lié aux spécificités des réseaux hybrides, à savoir la description fine des architectures de machines multiprocesseurs.

Les modèles créés et validés (contrôle syntaxique et sémantique) avec HADEL sont stockés au sein de bibliothèques interrogeables dynamiquement via un module de requêtes. L'utilisateur a alors la possibilité d'interroger la base pour savoir si une machine spécifique est disponible à un instant donné. La pertinence des informations de cette base de données doit donc être maintenue. Comme Hadel est un langage de description très simple, il n'est pas difficile de mettre à jour périodiquement ces informations (les attributs sont peu nombreux et de faible taille). Enfin, à partir de ces descriptions, il est possible de générer du langage QNAP2 pour éventuellement simuler le comportement des architectures matérielles (comme dans le logiciel Modarch de Simulog).

1.1.2. Description des architectures logicielles : VODEL-D

Dans VODEL-D, l'application est décrite par des entités logicielles, les DVSC, qui communiquent entre elles à travers des DVSL et/ou DVGL. Un DVSC est un composant logiciel hiérarchique qui dispose d'attributs facilitant son placement et son évolution dynamique. L'évolution dynamique concerne la granularité (un DVSC peut contenir d'autres DVSC, de manière hiérarchique) et la localisation des composants. Les attributs d'un DVSC facilitant l'équilibrage de charge et d'application (et donc leur utilisation par des algorithmes de placement) sont :

- **type** : un composant logiciel est *actif* s'il contient au moins un processus, sinon il est *passif* ;
- **l'évolution** : un composant logiciel est *libre* s'il peut être déplacé, sinon il est *fixe* ;
- **hiérarchie** : un composant logiciel peut être *primitif* ou *hiérarchique*.
- **regroupement** : un composant logiciel est *dissociable* si on peut le migrer par parties, sinon, il est *indissociable* ;
- **répartition** : un composant logiciel est *réplicable* s'il représente des données non partagées, migrable s'il peut changer de localisation, sinon il est *centralisé*.
- **attachement** : l'attachement qui existe entre entités logicielles correspond donc à une force d'attraction entre deux composants.
- **liens** : un composant logiciel peut avoir des liens vers du code ou des ressources extérieures qu'il faut prendre en compte.

1.2. Recherche de la granularité des composants et de leur placement

Dans VODEL-D, la modélisation de la granularité de l'application est réalisée au moyen des entités logicielles. Les entités logicielles de définition correspondent à la granularité calculée (au niveau conception), alors que les entités logicielles d'exécution correspon-

dent à la granularité courante (au niveau de l'exécution). Les deux types d'entités logicielles (DVSC passif et actif) donnent une idée du niveau de granularité géré en fonction du type d'équilibrage recherché :

- dans le cas de l'équilibrage de charge, les entités actives sont en général des programmes composés d'un processus et de zéro à plusieurs processus légers. Les entités passives logiques étant représentées par des fichiers de code ou des fichiers de données. Le graphe de DVSC représente alors un graphe de précédence des applications à lancer.
- dans le cas de l'équilibrage d'application, les entités actives sont en général des processus légers ou des squelettes de code et les gestionnaires du prototype sont des processus. Ils sont directement issus de la phase de génération de code, ce qui démontre le rôle prépondérant du générateur de code sur la granularité des composants et sur leur schéma d'interaction. Les entités passives modélisent les ressources logicielles de grain et de durée de vie faibles (fichiers temporaires et stockage de données). Le graphe des DVSC représente, dans ce cas, le squelette de contrôle de l'application et donc l'architecture logicielle intrinsèque de l'application.

L'unification du placement dans les environnements parallèles et répartis, qui était l'un de nos objectifs, est **donc réalisé au niveau conceptuel**, grâce à nos langages HADEL et VODEL-D. Leur mise en oeuvre réelle se heurte encore à des barrières théoriques (il faut trouver des algorithmes de placement efficaces en fonction des problèmes et des machines disponibles) et pratiques (les plate-formes actuelles qui supportent ce type de placement sont encore du domaine de la recherche [Bretelle & al. 95, Cabillic & al. 96, Folliot 96]).

1.3. Exécution répartie

L'exécution répartie est réalisée sur des machines à charge variable ou nulle. En cas de surcharge, la migration de processus peut être demandée. La décision de migration est en général prise au niveau système par le répartiteur de charge. Dans le cas de l'équilibrage d'application, cette décision de migration est soumise au respect des contraintes imposées sur l'application. Cette approche n'est valable que si l'on est conscient que l'utilisateur ne recherche pas une accélération maximale et immédiate, mais cherche plutôt à mettre au point son application. C'est pourquoi nous avons introduit dans VODEL-D deux notions importantes : le contrat d'exécution et le plan d'exécution.

- **le contrat d'exécution** est un évaluateur de la qualité de service. Il est passé implicitement entre le concepteur d'applications qui décrit ses contraintes sur l'architecture de définition et le générateur de code ou le répartiteur de charge qui tentent de les respecter. Ce contrat est le garant de la traçabilité entre l'architecture de définition et celle d'exécution.
- **le plan d'exécution** est une modélisation d'un état de l'application et correspond à un ensemble d'entités logicielles de définition. L'exécution de l'application est modélisée par l'enchaînement des plans. Le changement de plan d'exécution est réalisable si les contraintes imposées dans le contrat d'exécution sont respectées (pré-conditions bloquantes).

La description dynamique de l'application prend donc en compte l'évolution dans le temps des entités logicielles, au sein de plans d'exécution tout en respectant leur contrat d'exécution. Les contrats d'exécution dépendent à la fois du type d'équilibrage et du type d'applications gérés.

1.4. Analyse de l'exécution et optimisation

Une fois l'exécution terminée, l'architecture logicielle finale est comparée avec celle initiale (principalment en comparant la localisation et la granularité des composants logi-

ciels). Une fois les différences trouvées, il reste à déterminer leur degré d'importance et leur(s) influence(s) sur le cours de l'exécution.

Si on se place dans le cadre de l'équilibrage de charge, ce calcul différentiel et son interprétation sont réalisés automatiquement par la plate-forme de répartition de charges. Cette dernière modifie alors l'architecture logicielle pour améliorer les performances lors d'une future exécution. Si on se place dans le cadre de l'équilibrage d'application, il faut non seulement propager les différences de l'architecture réelle vers l'architecture virtuelle, mais aussi de l'architecture virtuelle vers le modèle semi-formel ou formel. Dans ce dernier cas, la connaissance des règles de génération automatique de code peuvent faire gagner du temps lors du travail de rétro-ingénierie. **La traçabilité doit donc être assurée sur tout le cycle de développement mis en oeuvre, si l'on désire bénéficier de cette étape d'analyse et d'optimisation.**

1.5. Conclusion

Le cadre de notre étude étant celui des applications réparties et parallèles, nous avons été conduits à mener de nombreux états de l'art et à mettre au point une approche méthodologique générale. Nous avons ensuite appliqué cette approche pour unifier la gestion de l'équilibrage de charge et d'application et la gestion des applications parallèles et réparties. Pour cela, nous avons dissocié l'architecture logicielle de définition, de l'architecture logicielle d'exécution (cf. Figure 82). Toutes deux sont décrites par le même langage VODEL-D.

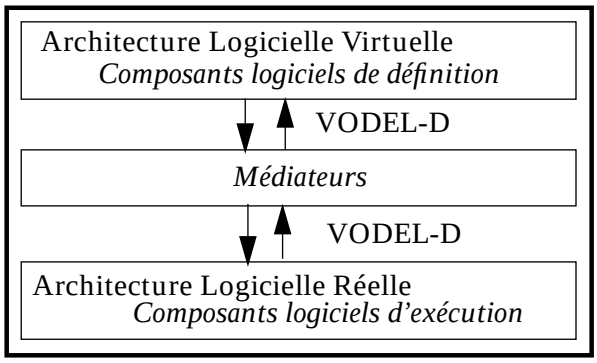

Figure 82 : De l'architecture logicielle virtuelle vers l'architecture logicielle réelle

Le calcul de la granularité de l'architecture logicielle virtuelle est basé sur des algorithmes de placement prenant toujours en entrée des descriptions de composants logiciels décrit en VODEL-D et de machines décrites avec HADEL. Ces algorithmes ne sont pas triviaux et dépendent à la fois du type d'équilibrage réalisé et des plate-formes utilisés. Ainsi, par exemple dans le cas de l'équilibrage de charge, on produit des algorithmes qui tentent de minimiser le temps d'exécution d'une application et d'optimiser l'utilisation des ressources gérées par la plate-forme système (telles que le type de processeur, la taille mémoire des machines, le système d'exploitation utilisé, etc.). Dans le cas de l'équilibrage d'application, nous avons montré que les algorithmes dépendent à la fois du modèle de spécification utilisé (avec une préférence pour les modèles semi-formels), de l'architecture du code généré et des politiques de placement des gestionnaires contrôlant l'exécution du prototype.

Nos études ont aussi montré que l'exécution d'une application est liée à de nombreux critères matériels et logiciels influant fortement sur les performances, sans pour autant remettre en cause la validité de l'architecture. Ainsi, par exemple, la lenteur d'exécution apparente d'une application peut très bien résulter du choix d'une machine lente ou d'un placement réalisé avec de mauvais critères. C'est pourquoi, la modélisation de l'exécution (et de l'architecture logicielle d'exécution) est primordiale. Cette modélisation doit de plus, préserver la traçabilité tout au long du cycle de vie des composants logiciels et

de leurs relations. VODEL-D et sa capacité de description des architectures logicielles de conception et d'exécution, nous a permis d'atteindre nos objectifs de ce point de vue.

A l'interface de ces deux types d'architectures logicielles, on trouve les médiateurs mis en oeuvre via une axe langage (l'application intègre ses propres médiateurs) ou via un axe système (on externalise les services médiateurs, l'application utilisant juste les services fournis).

La Figure 83 présente finalement un environnement de développement tel que nous le concevons idéalement. A partir d'un formalisme opérationnel et de règles de génération, il est possible de créer automatiquement un générateur de code pour un langage donné (C, C++, Ada). Ce générateur de code est construit de telle manière qu'il optimise le code et qu'il préserve la traçabilité bi-directionnelle entre le modèle initial et les gabarits de code générés. Ces gabarits de code sont ensuite utilisés tels quels ou complétés pour obtenir une application complète. L'exécution de cette application est réalisée en s'appuyant sur un environnement d'exécution interceptant tous les appels de services et de données transitant par des médiateurs. Ces appels sont ensuite pris en charge par le médiateur des médiateurs, qui en fonction de la granularité des composants logiciels gérés, du type d'équilibrage souhaité et des médiateurs disponibles, redirige les appels. Le médiateur de médiateur peut par exemple intercepter les appels entre processus locaux et faire appel a un médiateur de gestion des IPC et faire de même avec les appels distants en les transférant vers des médiateurs gérant les RPC, les rendez-vous, les transactions, etc.

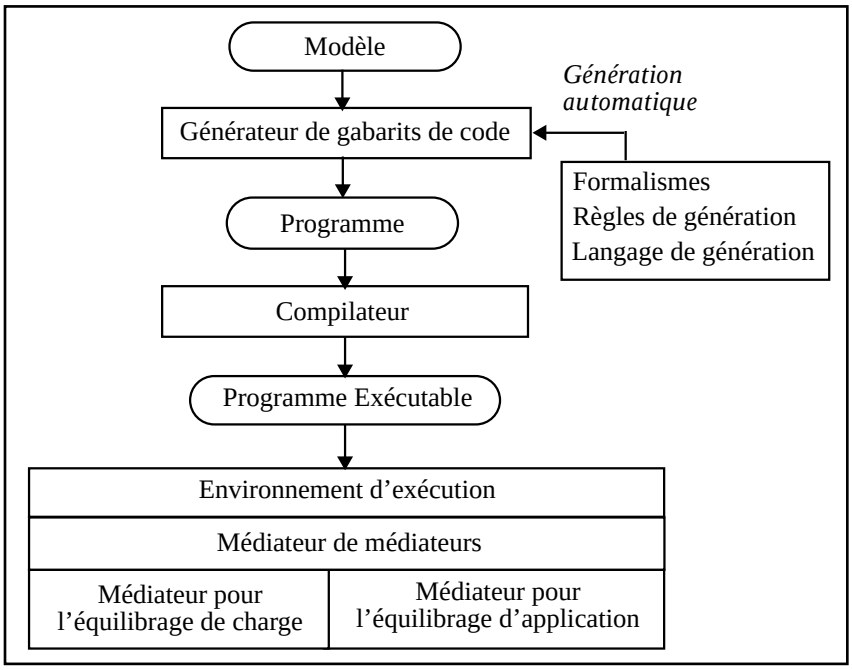

Figure 83 : Un environnement de prototypage idéal

Enfin, l'évolution des architectures client-serveur pose de nouveaux problèmes, aussi bien au niveau des spécifications (car les composants sont autonomes et mobiles et gèrent des interactions multiples et dynamiques), qu'au niveau de la gestion de l'extensibilité des architectures logicielles et matérielles (notamment sur Internet). Il en est de même de leur programmation (on utilise de plus en plus des langages objets compilés dans un langage intermédiaire portable) et de leur déploiement (on cherche alors un médiateur adapté aux besoins). Dans la section suivante, nous présentons quelques perspectives concernant nos travaux sur la génération d'application et sur l'équilibrage de charge.

2. Perspectives

Nous différencions dans nos perspectives celles relatives à l'équilibrage d'application et celles relatives à l'équilibrage de charge et à IDEFIX.

2.1. Equilibrage d'application et génération de code

Une fois un système parallèle spécifié, son implémentation reste difficile et coûteuse. La spécification peut être mal interprétée et des choix d'implémentation peuvent changer le comportement du système. Les techniques de prototypage répondent à ces problèmes et permettent [Burns 93] :

- de valider les spécifications formelles du modèle,
- de produire automatiquement un programme exécutable appelé prototype,
- de maîtriser complètement le cycle de vie du logiciel,
- d'offrir un gain de temps sensible en développement et mise au point du logiciel.

Le prototypage a été largement expérimenté et permet actuellement la génération de code automatique [Luqi 92, Kordon 92 & 94]. Néanmoins, dans le cas de prototype réparti, on constate que le calcul automatique du placement est rarement mis en oeuvre. Le placement est alors plus ou moins géré par l'utilisateur, entraînant parfois une altération des propriétés du système du fait de ses faibles performances [Dolev & al. 94]. L'utilisateur est donc amené à préciser les contraintes de localisation des composants logiciels sur les machines cibles et à prendre en compte la gestion de l'hétérogénéité à la fois matérielle et logicielle. Le problème étant alors d'exploiter au mieux la puissance de calcul disponible sur le réseau, tout en respectant les contraintes imposées.

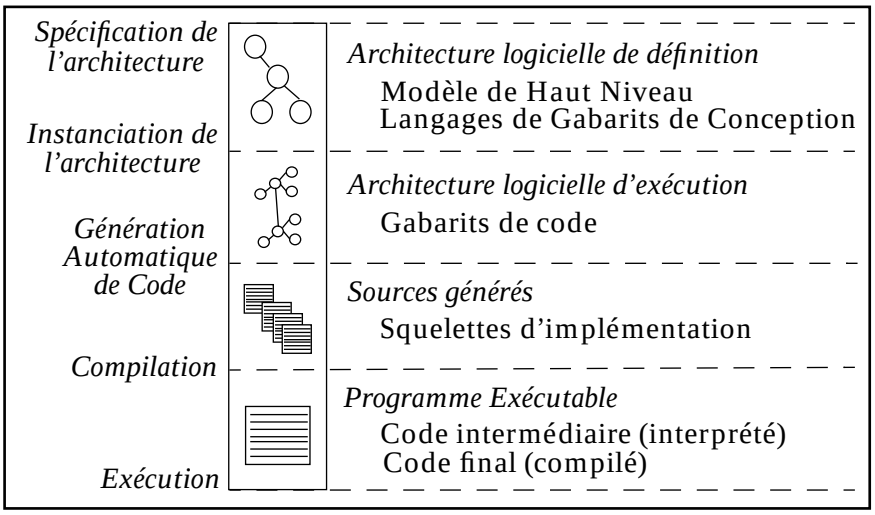

Figure 84 : Une nouvelle vision de l'équilibrage d'application et de la génération de code

De nos études, nous avons conclu qu'il était difficile à partir d'un modèle formel de générer automatiquement du code et le fichier de placement de ce code sur une architecture matérielle donnée. Nous avons donc opté pour l'utilisation d'un modèle semi-formel opérationnel, nommé H-COSTAM. L'utilisation d'un modèle semi-formel donne plus de liberté à l'utilisateur et adopte un point de vue optimiste. Cet optimisme est fondé sur l'idée que l'utilisateur a déjà une petite idée de l'architecture logicielle qu'il souhaite. Il veut néanmoins pouvoir vérifier que ces choix sont justifiés. L'utilisation de passerelles permettant de passer du modèle semi-formel à un modèle formel était alors la meilleure solution. De nos jours, l'utilisateur, veut vérifier que ces choix de construction sont valides au niveau architectural, en fonction des besoins spécifiques de son métier. C'est

pourquoi Alexander a créé des gabarits de conception réutilisables pour des architectes immobiliers. Cette idée reprise depuis modifie le cycle de développement des applications (cf. Figure 84).

Les langages de gabarits de conception donnent les règles et les modes d'emploi des composants de l'architecture en fonction de domaines précis. Ils évoluent peu et conduisent à une architecture générique pouvant faciliter le placement statique. L'instanciation de l'architecture logicielle de définition permet la génération de gabarits de code.

A partir de ces gabarits de code, on génère automatiquement des squelettes d'implémentation en fonction des langages et des systèmes cibles [Budinsky & al. 96] et [Mahmoud 97]. Les squelettes d'implémentation sont ensuite compilés pour donner naissance à des programmes exécutés ou interprétés.

Nous avons remarqué que ces approches architecturales sont actuellement très utilisées dans le cadre des technologies orientées objets. D'où l'idée de faire de VODEL-D un véritable langage orienté objet, qui servirait de langage de gabarits, capable à travers ses classes abstraites et concrètes de faciliter la génération de gabarits de code. Resterait alors à intégrer ces gabarits de code avec des générateurs de code distincts en fonction des langages cibles, tout en gardant la traçabilité des composants logiciels manipulés.

2.2. Equilibrage de charge et gestion de la mobilité

En ce qui concerne l'équilibrage de charge (et donc la plate-forme IDEFIX), il est difficile de mettre en oeuvre des algorithmes complexes, du fait du temps de réaction très court exigé pour assurer de bonnes performances [Folliot 96]. L'extension des critères de charge et des algorithmes est par contre à terme inévitable si on étend IDEFIX pour la prise en compte :

- **des multi-réseaux**. La répartition de charge dans les multi-réseaux est réalisée sur des grappes de machines (*cluster*) qui collaborent. Le schéma de collaboration suit un modèle hiérarchique (des grappes sont regroupées au sein d'une autre grappe, qui elle même est incluse dans une grappe, etc.) ;
- **des réseaux hauts débits**. Ils offrent une bande passante allant de quelques Megabits par seconde à quelques Giga-bits par seconde. Ainsi, certains travaillent sur des plate-formes de calculs parallèles sur ATM [Berger Sabbatel 97]. Certains étudient l'usage de cette bande passante et de nouveaux protocoles de communication, tels que le projet NOW [Anderson & al. 95] ou le projet Mosix [Barak & al. 96] avec le réseau Myrinet [Boden & al. 95].
- **du code mobile**. La gestion du code mobile interprété et optimisé à la volée implique la localisation des machines virtuelles et des compilateurs adaptés. D'autre part, la généralisation de processeurs intégrant des émulateurs natifs (comme les POWERPC émulant le code 68xxx) vont faciliter les règles de placement de logiciels hétérogènes ;
- **des systèmes à base d'ORB**. Les systèmes à base d'ORB découplent l'implémentation des composants logiciels de leurs interfaces, qui sont les points d'entrée de tout accès distant à l'objet. Le modèle d'exécution des implémentations est basé sur des interactions client-serveur entre des objets répartis, ce qui est une contrainte importante lorsqu'on essaye de faire coopérer des composants logiciels hétérogènes. C'est pourquoi, on utilise conjointement aux ORB des bus de messages. Le modèle d'exécution qui en résulte supporte alors des interactions asynchrones basées sur la gestion d'évènements. Les bus de messages sont des fournisseurs d'accès répartis à des services déclarés dynamiquement. Un client demandant un service ne doit pas connaître l'identité des serveurs d'objets. De plus, la distribution des objets est inhérente à l'ORB ou au bus de message et n'est plus gérée au niveau langage ;

- **du placement temporel**. Si le placement spatial (choix des machines) est toujours réalisé, il n'en est pas de même du placement temporel qui représente le choix de l'ordonnancement des composants à placer [Scherson & al. 96]. Ce système a été mis en place dans Choices qui réalise ce placement temporel au sein de groupe de processus (appelé un gang).

Toutes ces extensions sont issues d'une même volonté : la prise en compte de médiateurs multiples gérant des composants logiciels de granularité variable (des programmes, des objets, des processus, etc.). Si l'on se replace dans le cadre de la méthode MEDEVER, cela consiste finalement à fournir les facettes techniques indispensables à l'exécution des applications client-serveur universelles. Les facettes techniques que nous avons commencé à mettre en oeuvre dans IDEFIX concernent la gestion de la mémoire partagée répartie et des réseaux haut débit de type Fast-Ethernet (cf. Figure 85).

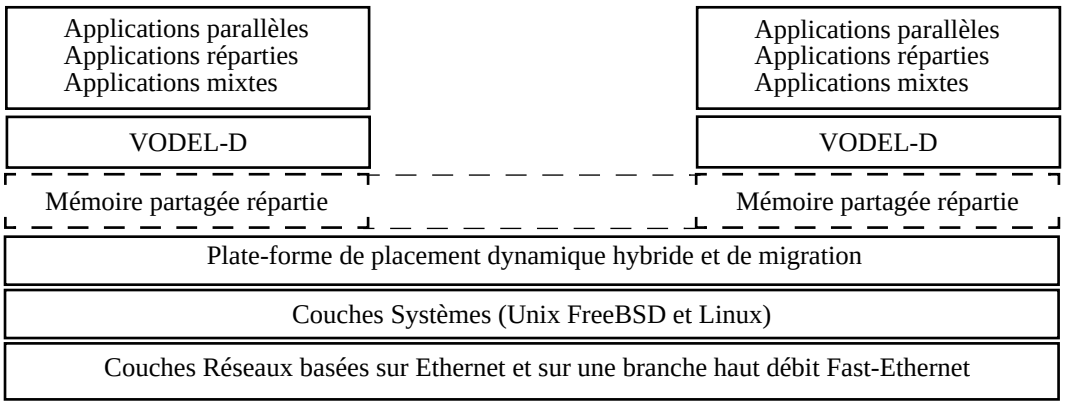

Figure 85 : Architecture d'IDEFIX

Nous nous sommes rendus compte, lorsque nous avons voulu intégrer la gestion de la mobilité des composants logiciels, que la création de facettes techniques est quelque fois insuffisante. Il faut donc modifier le noyau fonctionnel. Aussi, pour réaliser l'intégration de la mobilité des composants logiciels, nous devons intégrer au sein d'IDEFIX, le modèle des Territoires [Raverdy 96] et les facettes techniques qu'il induit.

2.2.1. Intégration du modèle des Territoires

De nos premières expérimentations, il ressort qu'il serait intéressant de disposer d'une gestion hiérarchique des ressources, indépendantes de la gestion des applications. C'est pourquoi, nous pensons redéfinir l'architecture d'IDEFIX en y intégrant le modèle des Territoires [Raverdy 96]. Ce modèle est composé de trois entités de base (cf. Figure 86) :

1) **le Domaine de Calcul (DC)** est un ensemble de noeuds (élément physique autonome du réseau auquel on associe un ensemble de ressources), coopérant pour offrir l'accès à leurs ressources aux applications s'exécutant sur ce domaine. Un domaine de calcul est équivalent au cluster dans Utopia [Zhou & al. 93]. Chaque domaine de calcul possède un gérant qui contrôle l'activité générale des noeuds de son domaine. Son rôle est d'indiquer aux applications les ressources existantes dans le domaine. Il dialogue avec d'autres gérants de domaine afin de rediriger les demandes des applications ne pouvant être satisfaites dans le domaine de calcul.
2) **le Territoire de ressources (TR)** rassemble un ensemble de ressources du même type. Les ressources de ce territoire sont accessibles par les applications s'exécutant dans ce domaine de calcul. Chaque TR possède un gérant qui contrôle l'acquisition et les accès concurrents aux ressources
3) **le Territoire d'exécution (TE)** est un ensemble dynamique de noeuds d'un domaine de calcul sur lesquels s'exécutent les processus d'une application. Le territoire d'exécution évolue en fonction du comportement interne de l'application,

des autres applications et du système. Un TE est associé à une et une seule application. Chaque TE possède un gérant, situé sur un des noeuds du territoire d'exécution, qui est chargé de la gestion des noeuds composant le territoire. Le gérant doit acquérir et rendre les ressources, négocier avec les gérants des territoires de ressources pour réduire les goulots d'étranglement et contrôler l'utilisation des ressources par les processus de l'application.

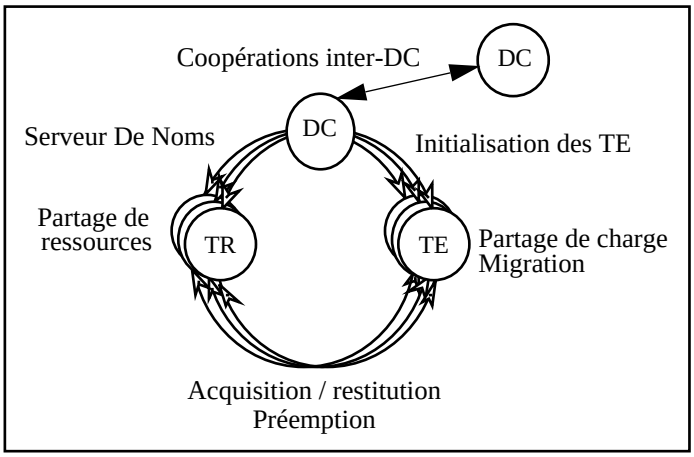

Figure 86 : Les différents éléments du modèle des Territoires et leurs relations

Intégration du modèle des territoires dans IDEFIX

L'architecture matérielle sur laquelle nous fondons nos travaux étant basée sur un réseau local de stations de travail et de machines parallèles, nous en déduisons au niveau du modèle des territoires :

1) que le langage HADEL a donc pour vocation de décrire les Territoires de ressources et les ressources qu'ils contiennent. Ainsi :
   - un unique Territoire de Ressources gère l'ensemble des stations de travail et le gérant de ce territoire est situé sur le serveur de charge de Gatostar ;
   - un unique Territoire de Ressources est associé à chaque machine parallèle et le gérant est placé systématiquement sur la machine hôte si elle existe, sinon, sur l'un des noeuds de la machine parallèle.
2) que le Territoire d'Exécution est formé à partir de la description en VODEL-D des entités logicielles d'exécution et de leur placement. Le gérant du TE manipule les plans d'exécution et vérifie que les contrats d'exécution sont respectés.
3) qu'un seul Domaine de Calcul est nécessaire pour gérer les TR et les TE et son gérant est implémenté de manière centralisée par Gatostar.

Extension vers le placement dynamique hybride

Le modèle des territoires est adapté à la coopération entre systèmes hétérogènes de répartition de charge, ce qui est un point primordial si l'on désire faire du placement dynamique hybride. On confie alors, dans un réseau hybride, la gestion dynamique de la charge à Gatostar en ce qui concerne les stations de travail et à des systèmes spécifiques tels que Athapascan ou Mars pour des machines parallèles exécutant des programmes de type SPMD.

Dans le cas de multi-réseaux, plusieurs DC peuvent collaborer pour étendre ponctuellement leur TR ou leur TE [Folliot 96]. Les demandes de ressources entre DC se font par l'intermédiaire de description en langage HADEL. Le modèle des Territoires est donc utilisé pour unifier HADEL et Gatostar et pour que Gatostar supporte l'ajout dynamique de ressources au sein d'un territoire de ressources donné. Les descriptions des ressources

étant réalisées de fait en langage HADEL. La Figure 87 résume la nouvelle architecture qui résulte de l'intégration du modèle des Territoires.

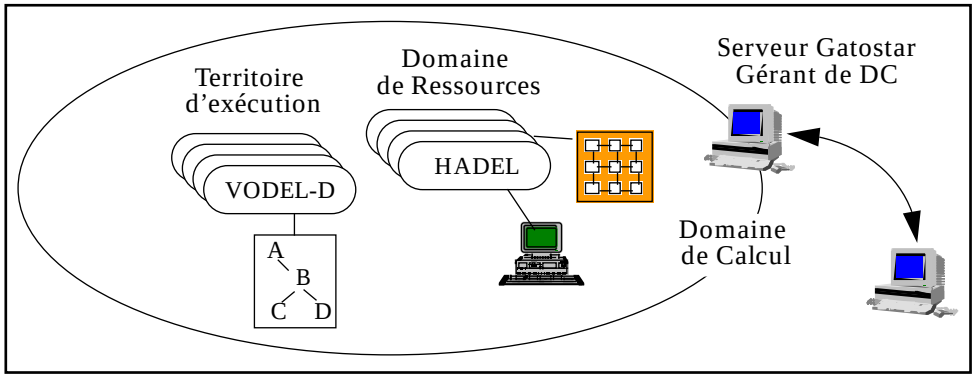

Figure 87 : Intégration de l'environnement et du placement dynamique dans IDEFIX

Synthèse

L'apport principal de ce modèle concerne son approche systématique de la gestion, où un algorithme de placement est associé à une application, un algorithme de répartition de charge est associé à un type de ressources et un algorithme de coopération entre domaines est associé à un domaine [Folliot 96]. La répartition de charge est ainsi décomposée en trois niveaux d'algorithmes coopérant : environnement de calcul, applications et ressources. Les premières implémentations montrent que le modèle des territoires est adaptable, efficace et extensible [Raverdy 96].

3. Références bibliographiques

[Anderson & al. 95]) T.E. Anderson, D.E. Culler & D.A. Patterson, «A case for NOW (Network Of Workstations)», IEEE Micro, Vol 15(1), pp. 54-64, February 1995.

[Barak & al. 96] A. Barak, O. Laden & Y. Yarom, «The Now MOSIX and its Premptive Process Migration Scheme», http://www.cs.huji.ac.il/papers/distrib, 1996.

[Berger Sabbatel 97] G. Berger Sabbatel, «calcul parallèle sur réseaux ATM : état de l'art et perspectives», Technique et Science Informatique, Vol 16 (7), pp. 835-863, 1997.

[Boden & al. 95] N.J. Boden, D. Cohen, R.E. Federman & al., «Myrinet: A gigabit-per-second Local area network», IEEE Micro, Vol. 15 (1), pp.29-36, February 1995.

[Bretelle & al. 95] B. Bretelle, T. Terracol & B. Folliot, «Placement d'applications parallèles en environnement hybride», Université pierre et Marie Curie, Paris, Rapport de Recherche, No MASI 95-19, Juin 1995.

[Budinsky & al. 96] F.J. Budinski, M.A. Finnie, J.M. Vlissides & P.S. Yu, «Automatic Code Generation from Design Patterns», IBM systems Journal, Vol. 35 (2), pp. 151-171, 1996.

[Burns 93] C. Burns, «REE - A Requirement Engineering Environment for Analizing and Validating Software and System Requirements», Proceedings of the 4th International Workshopon Rapid System Prototyping, N. kanopoulos Ed., IEEE Computer Society Press, Research Triangle Parc, pp. 188-193, June 1993.

[Cabillic & al. 96] G. Cabilic & I. Puaut, «Répartition de charge dans Stardust: un environnement pour l'exécution d'applications parallèles en milieu hétérogène», Actes de l'école thématique CNRS placement dynamique et répartition de charge, Presqu'île de Giens, pp. 167-174, Juillet 1996.

[Dolev & al. 94] D. Dolev, R. Strong & E. Wimmers, «Experiences with RAPID prototypes», Proceedings of the Vth IEEE International Workshop on Rapid System Prototyping, Grenoble, France, June 20-23, pp. 62-72, 1994.

[Folliot 96] B. Folliot, «Contribution à une approche système du placement dynamique dans les systèmes répartis hétérogènes», Thèse d'habilitation à diriger les recherches de l'Université Pierre et Marie Curie, Décembre 1996.

[Hürsch & al. 95] Walter Hürsch & Cristina Videira Lopes, «Separation of Concerns», Northeastern University Technical Report, NU-CCS-95-03, Boston, February 1995

[Kordon 92] F. Kordon, «Prototypage de systèmes parallèles à partir de réseaux de petri colorés, application au langage ADA dans un environnement centralisé ou réparti», PhD thesis, Université Pierre et Marie Curie, 1992.

[Kordon 94] F. Kordon, «Propososal for a Generic Prototyping Approach», Proc. IEEE Symposium on Emerging Technologies and factory Automation, Tokyo, Japan, pp. 396-403, November 1994.

[Lieberherr & al. 95] K.J. Lieberherr, I. Silva-Lepe & C. Xiao, «Adaptative Object-Oriented Programming Using Graph-based Customization», Communications of the ACM, Vol. 37(5), pp. 94-101, May 1995.

[Luqi 92] Luqi, «Computer Aided System Prototyping», Proceedings of the 3rd International Workshop on Rapid System Prototyping, IEEE Computer Society Press, Research Triangle Parc, pp. 50-57, June 92.

[Mahmoud 97] Q. Mahmoud, «Implementing Design Pattern with Java», Developer's Java Journal, Vol. 2 (5), pp. 8-14, 1997.

[Raverdy 96] P.-G. Ravedy, «Gestion des ressources et répartition de charge dans les systèmes hétérogènes à grande échelle : application aux environnements mobiles et parallèles», Thèse de l'Université Pierre et Marie Curie, 4 place Jussieu, 75252 Paris cedex 05, 1996.

[Scherson & al. 96] I. Scherson, R. Subramanian, V.L.M. Reis & L.M. Campos, «Scheduling Computationaly Intensive Data Parallel Program», Actes de l'école thématique CNRS placement dynamique et répartition de charge, Presqu'île de Giens, pp. 39-61, Juillet 96.

[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.