Numéro d'ordre : 85 Année 2000

HABILITATION A DIRIGER LES RECHERCHES

présentée à

L'UFR DES SCIENCES ET TECHNIQUES DE
L'UNIVERSITÉ DE FRANCHE-COMTÉ

pour obtenir

LE DIPLÔME D'HABILITATION À DIRIGER LES RECHERCHES DE L'UNIVERSITÉ DE FRANCHE-COMTÉ

Spécialité Automatique et Informatique

Administration de l'exécution de composants logiciels sur les plates-formes à objets répartis.

par
Laurent PHILIPPE

Soutenue le 30 novembre 2000 devant la Commission d'Examem :

Rapporteurs	François	Bourdon	Professeur, Université de Caen
	Jean Marc	Geib	Professeur, Université de Lille I
	Michel	Riveill	Professeur, Université de Nice - Sophia Antipolis
Directeur de recherche	Hervé	Guyennet	Professeur, Université de Franche-Comté
Examinateur	Michel	Tréhel	Professeur, Université de Franche-Comté

Table des matières

Table des figures

Remerciements

Bien sûr ce travail n'a pu être mené à bout que grâce au soutien et l'aide de nombreuses personnes.

Je tiens à remercier François Bourdon, Jean-Marc Geib et Michel Riveill d'avoir accepté d'être mes rapporteurs avec tout le travail et l'investissement que ça implique.

Je remercie Michel Tréhel d'avoir accepté de participer à ce jury et de son soutien lors de ma candidature à l'HDR.

Bien sûr, Hervé Guyennet, en tant que directeur de recherches mérite également de figurer ici mais je me dois aussi d'insister sur la qualité des conseils qu'il a toujours su m'apporter depuis.. ma maîtrise ! Sans lui ce travail ne serait pas, je lui en suis grandement reconnaissant.

Je tiens aussi à remercier tout ceux qui ont eu confiance en moi et ont accepté de me "subir" : Pascal Chatonnay, Ludovic Bertsch, Huah Young Chan, Vincent Portigliatti et Yohann Bardin. Une part de ce travail leur revient.

Il y a aussi tous les collègues avec lesquels j'ai travaillé de près ou de loin JM, Sylvie, David, Christophe, François et tous les autres qui ont contribués à ce que mes recherches se passent dans un cadre agréable. Parmis eux, je ne peux pas ne pas accorder une mention spéciale à tous les gens du blockhaus qui ont permis que ce travail se déroule dans la bonne humeur, il n'est plus mais je le regrette.

Et puis, j'ai évidemment gardé le meilleur pour la fin... il y a Béné, mais je crois que je lui ai déjà dit ;-), et puis Félix et Basile même si ça ne sert à rien puisqu'ils ne savent pas lire :-)).

Chapitre 1

Introduction

Le développement logiciel est soumis à des contraintes fortes telles que la réduction des coûts de mise au point et de maintenance des applications, la course à l'ergonomie et à l'adaptabilité. Afin de faire face à ces besoins de plus en plus incontournables, plusieures solutions sont apportées. L'approche qui semble obtenir le plus de suffrages est la notion de composant issue directement de la technologie objet. Dans la programmation par composants, le logiciel est découpé en parties indépendantes, chacune fournissant un service. Ces parties sont implantées sous la forme d'objets encapsulés qui sont par la suite assemblés pour produire des services plus complets et des applications. Par rapport à la programmation objet, la démarche par composants met en avant la modularité, la définition claire des interfaces. Par contre, elle limite la portée de l'héritage et intègre un modèle d'exécution. Cette notion se trouve au centre de technologies clefs telles que CORBA, Java ou DCOM. Un des intérêts de cette technologie est de permettre aux applications de tirer aisément partie d'un environnement matériel intrinsèquement distribué puisque basé sur l'Internet. Il s'agit alors, pour construire une application, de développer et d'assembler des composants, briques logicielles, dont les interactions avec les autres composants sont réalisées par des invocations. L'exécution peut donc se faire facilement dans un environnement distribué en utilisant des supports d'exécution qui offrent une transparence des invocations distantes. Le but final étant d'arriver à mettre à disposition du programmeur des bibliothèques de composants (composants sur étagère) dont il réalisera l'assemblage.

Dans ce cadre, l'administration et le placement de composants devient une phase critique de l'exécution, puisque la dynamicité des applications ne permet plus une administration statique. Il est nécessaire, pour obtenir de bonnes performances, de savoir gérer de façon dynamique le placement de chacun des composants en tenant compte de ses besoins propres. L'administration de composants dans les systèmes répartis à objets est donc le thème central de ce document. Je présente les travaux que j'ai réalisé ou auxquels j'ai participé dans ce domaine. Cependant l'ensemble de mes recherches depuis la fin de ma thèse n'est pas limité à ces travaux. Je souhaite montrer dans cette introduction leur évolution vers les thèmes actuels. J'en profiterai donc pour en donner une vue plus large et présenter quelques unes de mes préoccupations qui n'apparaissent pas dans le corps du document.

La répartition de charge pour multicalculateur

Le travail que j'ai mené durant ma thèse m'a conduit à cerner les besoins, en terme de système d'exploitation des machines à mémoire distribuée. L'architecture de ces machines et leur potentiel laissaient alors espérer de grandes possibilités dans les domaines où une puissance de calcul importante était nécessaire, leur rapport coût/performance étant très intéressant par rapport aux super-calculateurs traditionnels. Tels que nous les avons spécifiés, les systèmes d'exploitation à image unique devaient s'adapter à ces configurations en intégrant des services de transparence évolués. Nous avons ainsi spécifié et mis en place les services nécessaires à un ordonnancement distribué, base de la transparence d'accès au processeur, et intégré ces services dans le système d'exploitation distribué Chorus/MiX [**?**]. Dans la pratique, cela a consisté à spécifier et réaliser un service de migration de processus et d'équilibrage de charge.

Ces travaux ont été la base de travail de mes recherches durant les années suivantes. La réalisation de maquettes permettant d'évaluer la mise en pratique de tels services étant un travail très coûteux, nous nous sommes intéressé à la modélisation et à la simultation des environnements d'exécution. Ce travail a donné lieu à une coopération étroite avec H. Guyennet, B. Herrmann et F. Spies. D'autre part, nous avons continué à réaliser des maquettes pour valider les résultats simulés. Ce travail nous a également permis de participer à un projet nationnal (LHPC-C3) autour d'une machine parallèle commercialisée par la société Archipel. Le travail, mené sur une maquette de bibliothèque de répartition de charge pour la machine EVA, a donné lieu à l'encadrement d'un DEA. Comme précédemment, la base de travail pour la mise en oeuvre de cette répartition, était le processus qui modélise l'exécution d'un programme.

Les principaux résultats de ces travaux ont été, d'un point de vue scientifique, des résultats sur le comportement d'algorithmes de répartition face à différents types de charge d'exécution et différentes architectures. D'un point de vue personnel, ces travaux m'ont permis de mieux appréhender le comportement global d'un ensemble de processus. Ils m'ont aussi montré que le nombre de paramètres à utiliser pour simuler le comportement réel des applications dynamiques communiquantes est bien trop important pour se fier aux simulations et qu'il est nécessaire de valider les résultats avec des réalisations pratiques.

La difficulté de maîtrise de ces multicalculateurs, malgré tous les efforts menés pour en simplifier l'utilisation, ne leur ont pas encore permis d'être utilisés en dehors des niches du calcul très hautes performances. Ainsi, il ne reste que très peu de modèles de machines à mémoire distribuée au catalogue des fabricants. Nous avons donc fait évoluer nos recherches en cherchant d'autres champs d'applications aux techniques d'ordonnancement distribué et en analysant les évolutions des machines de calcul pour comprendre leurs besoins.

Gestion d'objets

Les systèmes répartis à objets sont des supports d'exécution destinés à faciliter la prise en compte de la distribution des objets tant au niveau du programmeur que de l'utilisateur. Ces systèmes fournissent les services de base pour le support d'une transparence simple : la localisation et l'invocation distante. La norme CORBA spécifie ainsi un ensemble de fonctionnalités minimum à mettre en oeuvre dans un ORB (Object Request Broker, bus logiciel de communication entre objets) et les interfaces des services qu'il est possible de lui adjoindre [**?**]. Etant donné la proximité des buts visés par ces plate-formes avec ceux que nous avions sur les mul-

ticalculateurs, il nous a paru intéressant d'appliquer nos connaissances à ces environnements pour les raisons suivantes :

- la modularité et la granularité du paradigme était plus adaptée : plus le grain est fin, plus l'équilibrage l'est. Il faut toutefois prendre en compte le fait qu'une granularité trop fine nuit aux performances puisque les entités, du fait de leur courte durée de vie, ne bénéficient que peu des placements tout en conservant un coût de traitement globalement équivalent. Or, la granularité moyenne préconisée par les systèmes répartis à objets en font des entités adaptées à la répartition de charge.
- L'utilisation des concepts communs entre la programmation de l'application et son support d'exécution permettent de mettre en place des interactions plus fortes et donc de mieux observer et comprendre le comportement de ces entités. En particulier, la notion d'interface permet de mettre en place une observation des communications qu'il était difficile de mettre en oeuvre dans des processus. Ceci permettant de tenir compte, dans les décisions de placement, d'un nombre plus important de paramètres.

Si la norme CORBA propose effectivement un support pour le placement d'objets, elle ne propose dans son modèle aucun outil pour la gérer en fonction des propriétés du système. Nous avons donc spécifié un nouveau service de gestion d'objets capable de prendre en charge la dynamicité du placement. Par la suite, nous avons réalisé une maquette de gestion d'objets, baptisée POM (pour Positionning Object Manager). L'intérêt de cette maquette réside dans la prise en compte d'un nombre plus important de données (telles que les communications) pour les choix de placement, dans l'application de techniques multicritères pour ces choix et dans l'utilisation d'un modèle de comportement pour les objets. Notre réalisation nous a permis d'étudier, dans les procédures de décision, l'influence du paramétrage sur la qualité du placement. Nous avons également pu mettre en évidence le fait que la qualité de ce paramétrage dépend du type de l'application. Ce travail a été le point de départ d'une collaboration de plusieurs années avec l'équipe travaillant sur les systèmes répartis à objets du SEPT de Caen. Il a donné lieu à la soutenance d'une thèse sur le thème de l'intégration de la procédure multicritères dans le placement dynamique et d'un DEA sur l'utilisation de plans de tests pour la mise en évidence des dépendances entre paramétrage et propriétés du placement.

Les résultats de cette étude sont liés à l'utilisation de procédures multicritères dans le cadre de décisions concernant un système dynamique et à la dépendance entre des caractéristiques internes de l'application, telles que le ratio calcul/communication, et le paramètrage optimum. Au titre des problèmes ouverts, force est de constater la complexité de la mise en oeuvre d'un lien entre le paramétrage et les applications en l'absence de caractéristiques stables.

Les composants et l'administration

Le cycle de vie traditionnel d'une application prévoit, une fois les développements et tests de validation réalisés, un déploiement en situation, l'exploitation de l'application en cours d'exécution puis, lorsque l'application devient obsolète, la migration vers une application plus récente et l'arrêt de l'ancienne. Dans le cas d'applications traditionnelles à gros grains, l'administration de l'application consiste principalement à définir le support matériel nécessaire au support de l'exécution, aux choix de déploiement et d'arrêt des serveurs. La diminution du grain des applications - ce qui se produit avec les composants - a des conséquances sur le comportement en temps d'exécution et sur les besoins d'administration. En effet, la modula-

rité inhérente au concept rend possible et souhaitable différents niveaux de dynamicité : de mise en oeuvre des services, de chargement des composants, des liens entre les composants, de comportement, etc. Or ces dynamicités entrainent un travail d'administration beaucoup plus important. En effet, il n'est plus question d'applications bien emballées mais d'ensemble d'entités indépendantes dont l'administrateur d'application doit s'occuper. Comment concevoir un déploiement si le comportement change, si les composants évoluent séparément ? Notons, de plus, que ce changement de conception logicielle va de paire avec une autre révolution qui est celle de l'Internet qui devient le support d'exécution privilégié de ces applications. La répartition des composants et les évolutions architecturales, avec un nombre de services externalisés croissant, ajoute à la dynamicité.

La gestion d'objets, telle qu'elle a été présentée précédemment, a volontairement été centrée sur les problèmes existants dans un réseau local. L'introduction de la notion de domaine a permis, par la suite, d'étendre le sujet en introduisant de nouvelles propriétés et contraintes, telles que l'hétérogénéité ou le respect de consignes de sécurité tout en poursuivant le même but d'optimiser l'utilisation des ressources. Ceci nous a donc permis de passer de travaux ayant des visées proches de la répartition de charge, puisque issus de celle-ci, à un environnement proposant des interactions avec le logiciel dans le but de permettre de l'administrer. De même, le modèle des entités gérées a évolué pour passer des objets aux composants. Ce travail nous a permis de définir un nouveau modèle de gestion des composants intégrant la possibilité d'administrer ceux-ci. Il a fait l'objet d'un marché avec les équipes du CNET de Caen, issues du SEPT. Il a donné lieu à la soutenance d'une thèse sur le thème de l'intégration de la notion de domaine dans la gestion dynamique de composants. Un stage de DEA a également été réalisé sur le thème de l'utilisation de la notion de besoins des applications dans la procédure de décision multicritères.

Les résultats de ces travaux concernent principalement l'utilisation de besoins applicatifs dans le placement de composants et la spécification d'un service d'administration automatique. De plus, nous avons proposé des améliorations à la procédure de décision permettant de l'auto adapter en fonction du contexte applicatif. A titre personnel, ces études m'ont persuadé du fait, pour l'étude des applications dynamiques (composées d'éléments dont la structuration n'est pas fixe et dont le comportement nous est mal connu) qu'un passage par l'expérimentation s'avère nécessaire même si seule la modélisation permettra de généraliser.

Ces travaux constituent le coeur de mon activité de recherche actuelle et la base de ce rapport. Je présenterai les perspectives et les orientations que je souhaite y donner dans la conclusion. Cependant, mes activités de recherche ne sont pas limitées à ces travaux. Je collabore avec d'autres collègues sur des sujets que je n'ai pas intégré dans le corps du document pour en assurer la cohérence. Ainsi je présente, dans la suite de ce chapitre, mes activités sur la mise en place, au sein de l'architecture CORBA, de services pour le parallélisme.

CORBA parallèle

L'apparition des réseaux à hauts débits a fait évoluer la recherche qui était centrée sur les multicalculateurs vers la gestion de clusters : machines composées de stations de travail (souvent de type PC) reliées par un réseau de ce type. Ces configurations de machines, permettent de réaliser des "serveurs de calcul" à moindre prix mais supposent une utilisation par des programmeurs spécialisés. Les résultats que nous avions obtenus pour la répartition de charge

pour les multicalculateurs restent valides pour ces configurations mais les lacunes des systèmes et des environnements de programmation le restent aussi. Or, la baisse des coûts des machines parallèles permet d'intéresser un public plus large qui n'est pas forcément expert en parallélisme. Je me suis donc interrogé sur la possibilité d'utiliser CORBA comme environnement d'exécution parallèle. L'idée est de faire profiter ces utilisateurs des facilités de l'utilisation des composants et de leur portabilité.

Les travaux que j'ai mené dans ce domaine ont entièrement été réalisés en coopération avec le groupe GRID du LIFC. Plusieurs voies ont été explorées :

- la mesure de performances de communication d'ORB. Les performances de communication sont toujours un aspect crucial du support d'exécution pour les applications parallèles. Les résultats ont montré que certains ORB savaient être, sur des schémas de communications synchrones, aussi rapides, voire plus, que des environnements d'exécution parallèles standards tels que PVM.
- la réalisation d'applications parallèles au dessus d'un ORB compatible CORBA. En particulier, nous avons réalisé l'implantation d'une factorisation WZ parallèle au dessus de l'ORB Cool. Ce travail nous a permis de mieux comprendre les lacunes et les difficultés du modèle de communication "tout synchrone" pour les applications parallèles.
- l'étude des types de communication et l'utilisation du paradigme client-serveur dans la programmation parallèle. A la suite de l'implantation précédente, le travail en cours vise à réaliser un support de communication basé sur un ORB et offrant un modèle de communication adapté.
- l'implantation d'un courtier d'accès à des composants parallèles dans un environnement hétérogène. L'idée est, à partir d'une console d'interprétation de commandes de calcul (calcul sur les matrices, par exemple) de retrouver le serveur adapté à l'exécution du calcul en tenant compte du contexte de l'exécution (commandes précédentes, placement des données, etc). Dans le cadre de ces travaux, je participe à l'action incitative INRIA OURAGAN.

Cette liste ne saurait être exhaustive si j'oubliais l'utilisation de techniques génétiques pour l'auto adaptation des procédures de décision en fonction du contexte applicatif. Néanmoins ma participation y est minime et les résultats ne semblent pas encore applicables.

Plan du document

Je présente maintenant le plan de mon document. J'ai voulu y décrir le cadre complet de l'environnement dans lequel je réalise mes travaux. Pour cela, j'ai choisi de le scinder en deux parties : une partie sur le contexte et l'état de l'art et une partie regroupant mes contributions. Dans la première partie, je présente un modèle des applications que nous souhaitons gérer, leurs supports d'exécution, les outils que nous avons utilisés pour les gérer et les travaux proches des nôtres. Cette partie est composée de trois chapitres.

Le premier donne une vue globale sur les environnements à objets les plus en vue. Nous nous sommes attachés à en décrire, en dehors des généralités indispensables, le modèle de composant proposé et les aspects fonctionnels de leur mise en oeuvre. Nous y avons réalisé une comparaison des services offerts et des performances de communication. Nous concluons ce chapitre par une description plus complète du contexte applicatif visé par notre travail.

Le second chapitre présente une synthèse du placement dynamique et de son optimisation.

Nous y décrivons les principales fonctionnalités nécessaires à la mise en oeuvre de la répartiton de charge et la problématique qui y est liée. Nous illustrons ce propos par quelques exemples de réalisations appliquées aux objets. La mise en place d'un service optimisant le placement serait incomplète sans l'étude des procédures d'optimisation classiques. Malheureusement, la plupart d'entre elles sont inutilisables dans notre cas car beaucoup trop statiques. Néanmoins, nous nous sommes intéressés aux procédures multicritères et au modèle relationnel que nous présentons.

Le troisième chapitre donne un aperçu de la modélisation d'applications réparties et des techniques existantes dans l'administration d'applications. Nous nous sommes ainsi intéressés à la norme RM-ODP qui, à partir de points de vue, permet de spécifier les fonctionnalités des applications distribuées. La présentation de différentes réalisations d'administration nous permettra de faire le point sur l'état de l'art de ce domaine.

Dans la seconde partie, de ce document, je présente mes contributions aux domaines de la gestion d'objets et de l'administration automatique. Le quatrième chapitre est donc consacré à la définition de l'administration de composants telle que nous l'avons réalisée. Nous y spécifions le modèle d'administration automatique que nous proposons et l'application de la théorie multicritères aux procédures de décisions de la gestion d'objets. Les chapitres V et VI décrivent respectivement l'implantation mono et multidomaine de cette proposition. Pour finir, le septième chapitre présente les tests qui ont été réalisés sur ces implantations. Nous y décrivons les procédures de tests, l'environnement et les applications utilisées avant d'analyser les résultats obtenus.

Dans la conclusion, nous commençons par rappeler les résultats obtenus et nous donnons les perspectives ouvertes par ce travail, puis nous présentons les nouvelles orientations prises par nos recherches, en particulier dans le cadre de la mise en place d'une collaboration avec la société StellarX.

Première partie

Contexte et état de l'art

Chapitre 2

Les plates-formes à objets répartis

Un des enjeux des prochaines années est la maîtrise des environnements distribués tels que nous les avons décrits : Internet, télécommunications ou parallélisme, tant au niveau de la programmation, du déploiement que de l'exploitation. L'approche basée sur les composants est une approche largement reconnue pour être adaptée à ce type d'environnement : la modularité inhérente au modèle, associée à des supports d'exécution qui garantissent la transparence des interactions entre composants assure un déploiement tirant parti des ressources malgré leur distribution. Plusieurs environnements sont candidats au support de ces applications. Dans ce chapitre, nous présentons la norme CORBA et les deux spécifications principales Java et DCOM.

De manière générale, et quelque soit l'environnement, les objets serveurs rendent leurs fonctions disponibles en les exportant via une interface. L'interface constitue la description fonctionnelle du serveur qui la porte. Elle est exprimée comme un ensemble de fonctions avec leur prototype. Les objets clients, en fonction de leurs besoins, contactent un ensemble de serveurs dont les fonctions leur sont utiles et communiquent avec eux via les mécanismes offerts par le système. Ces plates-formes proposent donc à la fois une structuration du développement autour des "points de rencontre fonctionnels" proposés par chaque composant et un support d'exécution pour les instances qui composent l'application répartie. Ainsi ces environnements ont en commun le fait qu'ils sont, en plus du support d'exécution qu'ils offrent, intégrés aux environnements de développement dans le sens où le choix de l'environnement se fait au moment du développement de l'application ou du composant. Cette intégration peut être effective à différents niveaux : elle est complète au niveau de l'environnement Java puisque l'accès à l'environnement ne se fait pas par directives ou appels de fonctions mais par instructions du langage, elle est partielle pour DCOM et CORBA puisque réalisée à partir d'appels de fonctions et de bibliothèques jointes à un langage de programmation.

Pour chacun de ces environnements, nous étudions le support d'exécution offert et le modèle de composant défini.

2.1 CORBA

Une des organisations principales dans le domaine des systèmes répartis à objets est l´Object Management Group (OMG). Il s´agit d´un consortium dédié à la promotion de la technologie

objet. Son objectif est de mettre en place une structure pour le développement et la promotion des différents aspects de la programmation objet. Pour satisfaire cet objectif, l'OMG définit un modèle à base de standards «de fait» [?, ?, ?] [1].

2.1.1 Architecture générale

L´OMA (Object Management Architecture) est un modèle de référence définissant un système à objets comme un fournisseur de services à des clients. Ces services sont implantés par des objets et offerts aux clients à travers une interface de requêtes. A une requête, sont associés : une opération, un objet cible, éventuellement des paramètres et un contexte optionnel. L´interface décrit l´ensemble des opérations qu´un client peut demander à un objet. Les interfaces sont définies grâce au langage de définition d´interface (IDL). Un objet satisfait une interface s´il est susceptible de rendre l´ensemble des services définis dans cette interface. L´OMA (figure **??**) définit également des entités spécifiques :

- l´Object Request Broker (ORB) est le bus de communication logiciel. Il fournit les mécanismes d´échange entre les objets répartis. Son rôle est d´assurer la transparence de l´accès aux objets présents sur le bus. Nous revenons sur l´ORB dans la description de l´architecture CORBA.
- Les services objets sont les services optionnels qui donnent accès à des fonctions systèmes de base telles que la persistance ou le nommage. En spécifiant les interfaces de ces services, l´OMA permet l´indépendance des applications vis a vis des mécanismes systèmes et de ce fait, la portabilité des applications les utilisant.
- Les utilitaires communs correspondent aux services tels que l´aide en ligne ou l'administration du système. Ils sont essentiellement dédiés à être intégrés dans les processus de dialogue avec l´utilisateur.
- Les objets applicatifs désignent les objets spécifiques des applications (des clients et des serveurs) ou objets métiers. C'est dans cette partie que peuvent être regroupés les composants dédiés à un type d'application.

L'OMG a fait de l'interopérabilité une de ses priorités, dans la norme CORBA 2. Grâce à cette fonctionnalité, des objets implémentés au dessus de différents ORB peuvent travailler ensemble. En particulier, IIOP (Internet Inter-ORB Protocol) permet à des objets de communiquer via Internet quelques soient leur localisation et l'ORB sous-jacent. Ceci est réalisé en généralisant la notion d'objet d'interface pour obtenir des références d'objets, et en définissant des formats d'échange de données.

2.1.2 Le bus ORB

L´architecture CORBA définit le modèle du bus de communication logiciel ORB (Object Request Broker). Ses composants sont : les souches client (stub), l´invocation dynamique, l´adapteur d´objet, l´interface ORB, le squelette d´implémentation et le noyau ORB sur lequel reposent les composants précédents. Une vue synthétique de cette architecture est donnée par la figure **??**. Ces composants ont les fonctions suivantes :

1. Le lecteur désireux de lire une présentation, simple et illustrée d´exemples, de CORBA, pourra se reporter à la thèse de Philippe Merle [?]

FIGURE 2.1 – Architecture de gestion des objets (OMA)

- l´Object Request Broker assure le transport et la transparence des invocations entre les objets. Ce transport est assuré via un mécanisme approprié à la localisation relative de l´émetteur et du récepteur. L´interface ORB permet aux applications de communiquer avec l´ORB. C´est par ce biais que sont réalisées l´initialisation de l´environnement CORBA et la gestion de la mémoire entre autres.
- les interfaces d´invocation statique résultent de la traduction des spécifications IDL des interfaces en un langage de programmation. C´est au travers de ces souches (stub en anglais) que les clients invoquent les méthodes des serveurs. Leur fonction est l´empaquetage des données transmises lors des invocations. Elles délivrent ensuite ces invocations transformées à l´ORB.
- l´interface d´invocation dynamique permet à un objet ne disposant pas de l´interface d´un serveur de bâtir dynamiquement une requête vers ce serveur. Il est nécessaire de connaître l´objet cible et d´obtenir les types des paramètres depuis le référentiel des interfaces.
- l´adaptateur d´objets a pour fonction d´isoler le mode d´implémentation des objets du support de communication. Il gère les problèmes d´activation et de contrôle d´accès. Il utilise le référentiel des implémentations pour déterminer les processus à activer en fonction des opérations et des objets mis en œuvre.
- les interfaces de squelettes statiques prennent en charge le désempaquetage des données transmises lors des invocations et l´empaquetage des résultats. Elles sont, comme les interfaces d´invocations statiques, le résultat de la traduction des spécifications IDL.
- l´interface de squelette dynamique est l´équivalent, pour les serveurs, de l´interface d´invocation dynamique, pour les clients. Elle permet de recevoir des invocations sans disposer d´un squelette statique pour réaliser le désempaquetage des données.

De façon générale, une instance de l'environnement est présente sur chaque site du réseau, sous la forme d'un processus ou d'une bibliothèque. La compatibilité à la norme CORBA se fait principalement sur la base des services offerts par l'ORB. Les services extérieurs, services

FIGURE 2.2 – Bus de communication logiciel (ORB)

objets, utilitaires communs ou interfaces de domaines, sont normalisés sur la base de leur interface. Ils restent cependant optionnels.

2.1.3 Les services objets

Les services objets peuvent être réalisés sous forme de bibliothèques, associées au code en temps de compilation et éventuellement chargées dynamiquement, sous forme de serveur indépendants ou comme une combinaison des deux. Le choix qui est fait dans les différentes implantations d'ORB est surtout dépendant du type de service rendu. Par exemple, le service de nommage est généralement réalisé sous forme d'un serveur indépendant alors que le service de cycle de vie est généralement réalisé sous forme de bibliothèques.

De nombreux services objets ont été définis par la norme ou sont en attente de définition. Parmi ceux-ci nous pouvons citer le service de nommage qui permet, dans une structuration en contextes, d'associer un nom à une référence d'objet, le service de courtage qui permet de classifier les services et d'effectuer une recherche intelligente de service en fonction d'attributs recherchés ou encore le service de cycle de vie qui prend en charge la création, le déplacement ou la destruction des objets qu'il gère. Une description exhaustive de l'ensemble des services normalisés n'a pas sa place ici. On pourra trouver une description plus détaillée dans [**?**].

L'accès à des services de base peut être automatique à travers la notion d'objet notoire. C'est, par exemple, le cas du service de nommage ; celui-ci devant être accessible directement par les objets puisqu'il constitue la base de tout lien dynamique entre un client et un serveur. Les objets notoires sont accessibles, sur une plate-forme ORB, à travers l'appel à la fonction *resolv_ intial_ references*.

2.1.4 Le modèle de composant de CORBA

Le modèle de composant n'est introduit en tant que tel seulement depuis la norme CORBA 3. Les normes précédentes définissaient principalement le modèle d'interaction entre les objets hétérogènes (CORBA 1) et d'interaction entre les ORB (CORBA 2). L'aspect purement com-

posant, qui suppose la prise en compte du déploiement et de l'exploitation, plus des facultés d'auto-description, est proche de la définition qui en est faite dans les EJB de Java et est compatible avec celle-ci.

Les composants, dans CORBA, sont vus comme une spécialisation du modèle plus général d'objet. Ainsi, le type composant est introduit dans l'IDL comme étant une extension du type interface. Ceci permet aux composants d'être enregistrés dans le référentiel d'interface au même titre que les autres objets. Par contre, le modèle offre un certain nombre d'extensions telles que le support d'interfaces multiples, les modes de connexion entre composants, etc. La spécification peut être vue sous les deux faces du développement et de l'exploitation.

Du point de vue du développement, le but recherché est de pouvoir inscrire dans les fichiers d'interface une sémantique de comportement qui permet de compléter la description fonctionnelle donnée par l'interface. Ainsi, les composants peuvent, d'une part, déclarer offrir un ensemble de facettes qui correspondent à autant d'interfaces qu'ils implantent et, d'autre part, exprimer le fait qu'ils utilisent d'autres composants pour rendre leur service. La lecture de l'interface permet alors de savoir quelle sera la structure de connexion dans laquelle sera inscrite le composant. Enfin les composants pourront exprimer leur participation à un ensemble d'événements, soit en tant qu'émetteur, soit en tant que récepteur. Ces fonctionnalités permettent donc de mieux afficher les interactions du composant avec l'extérieur. Le tout peut être décrit dans le langage CIDL (Component Implementation Description Language). L'encapsulation du composant est réalisée dans un *home* chargé de créer les instances de composants et de les localiser.

Pour l'exécution des composants, le modèle est issu du modèle des EJB proposé par Java et que nous décrivons en **??**. Il s'appuie sur différents types de *containers* chargés de l'exécution et de la gestion des composants. Ainsi, un client désirant accéder à un composant s'adresse d'abord au *home* qui y est associé pour obtenir une référence sur celui-ci. Cette référence fait obligatoirement le lien vers un container qui interagit avec un POA (Portable Object Adaptor) pour gérer le mode d'activation du composant. Ce même container permet, par la suite, de gérer de manière transparente le mode d'échange mis en place avec le client en fonction du type du composant.

2.1.5 Quelques exemples d'ORB

De nombreuses implémentations du standard CORBA existent. Elles sont la preuve d´une activité soutenue dans ce domaine, tant de la part des industriels que des universitaires. CORBA ne spécifie que les interfaces des services à offrir, aussi toute latitude est laissée dans les choix d´implantation. Cependant, il peut y avoir quelques divergences dans les implantations et les normes successives ont souvent pour but de limiter les dérives. C'est, par exemple, le cas de la norme CORBA 2 qui unifie les services d'adressages autour d'IIOP pour permettre l'interopérabilité entre ORB.

Une description exhaustive de l'ensemble des ORB existants est difficilement envisageable tant le nombre de projets autour de la norme de l'OMG est important. Nous avons sélectionné ceux qui nous ont paru essentiels. Ceux-ci sont disponibles sur de nombreuses plates-formes.

ORBIX est un produit de la société Iona [**?**]. Il est une des résultantes du projet ESPRIT COMMANDOS [**?**]. Il supporte plusieurs langages de programmation dont C++ et Java.

Leader des ORB "commerciaux" il a été détrôné depuis peu par VisiBroker [?].

ORBAcus est développé par la société OOC (Object Oriented Concepts). Il est accessible gratuitement aux universités et donc largement utilisé dans ce cadre. Il supporte les langages C++ et Java et offre un nombre important de services. La version actuelle implante la norme CORBA 2.3.

OmniOrb2 est développé par un laboratoire de recherche commun à Oracle et Olivetti. Il est certifié compatible à la norme CORBA 2.1. Il est réputé pour être rapide, léger et fiable. Il n'offre qu'un service de nommage.

TAO a été développé à l'Université Washingthon. C'est un ORB entièrement gratuit qui est réputé pour avoir une orientation temps-réel et qui repose sur une couche de communication nommée ACE. Il offre un nombre important de services en particulier dans le domaine des télécommunications. Malheureusement, il est aussi de taille conséquente, ce qui nuit à son utilisation.

COOL ORB était un produit de la société Chorus [?]. Il est disponible sur plusieurs plates-formes dont solaris, linux, windows et ultrix. Ce produit est le successeur de COOL v2. Il est pour sa part compatible CORBA 2.0. L´un des points forts de COOL-ORB est de proposer une bibliothèque de «monitoring». Grâce à cette bibliothèque, il est possible d´obtenir des informations sur les communications réalisées par les objets. Dans COOL-ORB, la collecte est réalisée, par le système, grâce aux objets d´interface. Dans le cadre de notre étude, cette approche est intéressante.

David a été développé au CNET. Il est basé sur une couche générique de communication entre objets appelée Jonathan. Cette couche supporte, pour le moment, deux environnements de communication : CORBA et Java-RMI. C'est un ORB entièrement gratuit qui est au coeur d'une intiative française.

2.2 JAVA

Il est difficile de parler du langage Java sans parler de son succès planétaire et fulgurant : en cinq ans ce langage a su s'imposer dans toutes les applications informatiques comme un langage à fort potentiel. Nous ne reviendrons pas sur ce succès et ses raisons, pas plus d'ailleurs, que sur le contenu du langage ou ses qualités. Il est cependant important de rappeler le modèle d'exécution du langage car c'est ce qui le différencie le plus des autres langages : au lieu d'être compilé en code binaire pour générer un exécutable, un programme java est traduit en code intermédiaire puis interprété par une machine virtuelle. Ceci rend les programmes portables et donc utilisables dans des environnements hétérogènes.

Le langage en tant que tel n'est pas à proprement parler un support d'exécution distribué. C'est donc la machine virtuelle qui retiendra plus particulièrement notre attention pour sa prise en compte des communications entre sites. Les primitives d'accès à cette machine virtuelle font partie du langage, en tant que paquetages additionels. Il est alors légitime de se poser la question : "où s'arrête le langage et où commence le support d'exécution ?". Pour cette raison, nous confondons les deux notions. Nous nous intéressons principalement aux aspects liés aux objets distribués et aux composants.

Le langage Java est un langage orienté objet qui propose des fonctions de communication entre ses objets. Il constitue donc un bon support pour la mise en oeuvre d'applications à objets

distribués. Dans le langage Java, il existe plusieurs solutions pour réaliser une communication entre différents programmes : communication par sockets ou par invocations distantes. Nous n'abordons pas les sockets car elles ne sont pas dédiées à la mise en place et à l'interaction des composants. Par contre, en créant dans la version 2 de son langage les Java Beans, JavaSoft à résolument décidé de s'intéresser aux objets distribués à travers deux concepts : les RMI et les Beans.

2.2.1 Les invocations distantes

Le modèle d'invocations distantes proposé par Java, les RMI (Remote Method Invocations), diffère très peu du modèle de base défini par CORBA. Chaque objet-serveur publie une interface en langage IDL. Cette interface est compilée pour générer des classes-souches servant au codage et à l'acheminnement transparent de requêtes. Notons cependant que cette implantation est destinée à la communication entre programmes Java. Elle prend donc en compte l'hétérogénéité des plate-formes d'exécution mais pas celle des langages.

De même que dans CORBA, le lien dynamique entre composants est réalisé à travers un serveur de noms. Le nommage utilisé par ce serveur repose sur le système des URL. Le serveur de noms ne propose cependant pas de structuration de l'espace de nommage, celui-ci est plat.

Java-RMI étend le mécanisme de ramasse-miettes aux invocations distribuées. Il assure la sécurité de la même manière qu'en local.

Java-RMI permet de nombreux scénarios de communication incluant le lancement dynamique d'un serveur avec une sécurité équivalente à celle qui est assurée pour les *applets*, le chargement dynamique de *stub*, etc. Par contre il est, dans son modèle, moins complet que CORBA.

Le modèle de composant de Java repose sur la notion de *Bean*[2]. La spécification des composants repose sur deux modèles. Les beans et les EJB (Enterprise Java Beans). Le modèle d'exécution des deux types de composants est différent, nous présentons ces concepts dans les parties suivantes.

2.2.2 Les beans

La spécification Java Beans a vu le jour avec la version 1.1 du langage. Elle définit des composants logiciels réutilisables et manipulables à partir de logiciels interactifs. Le but est de permettre le développement d'applications à base de composants logiciels en simplifiant l'assemblage et l'intégration de ces composants.

La mise en place des composants repose sur quatres mécanismes :

Introspection qui permet de découvrir dynamiquement les méthodes d'une classe, les paramètres associés à cette méthode et les données de retour. Ceci pour réaliser des invocations dynamiques "à la CORBA" sur les composants.

Spécialisation qui permet d'adapter le composant aux besoins d'une application. Ainsi un utilisateur peut, s'il en a le droit, modifier les propriétés associées au *Bean*.

2. Le *java bean* est littéralement le grain de café, élément de base entrant dans la composition et la réalisation du café en tant que boisson

Evénements qui constituent une base importante du modèle de synchronisation et d'échange. Dans le cadre d'une application, un *Bean* doit s'intégrer dans un schéma d'évenement.

Persistance qui est réalisée en sauvegardant les *Beans* (sérialisés) dans des fichiers spéciaux. Ils peuvent alors être à nouveau utilisés lors d'invocation. Pour cette raison, un *Bean* n'est plus créé par l'instruction *new* mais plutôt par une méthode *instanciate*.

Les fonctionnalités des *Beans* permettent surtout l'accès à des données paramètrables et la découverte dynamique de l'interface. Ils constituent donc la base d'une programmation par composants mais n'offrent pas de gestion particulière pour la mise en oeuvre de la répartition de ces composants.

2.2.3 les EJB

Les EJB (Enterprise Java Beans) ont été spécifiés au sein de la plate-forme J2EE (Java 2 Enterprise Edition). Leur but est de servir de base à la définition de serveurs d'application, accédés à partir de clients WEB. Il s'agit en fait d'un modèle applicatif composé de trois niveaux (3-tier application).

L'architecture

FIGURE 2.3 – Architecture de EJB

Comme le montre la figure **??**, l'architecture EJB est constituée de trois niveaux :

- Le serveur constitue un premier niveau d'environnement d'exécution pour l' *Enterprise Bean* dont il prend en charge l'interface applicative. Il est en contact direct avec le client.
- un ou plusieurs container(s) qui offre(nt) des fonctionnalités techniques comme la persistance ou encore la transaction et constituent un second niveau d'environnement d'exécution dédié au services de haut niveau. Il est chargé dans le serveur.
- des composants dont la tâche est limitée à l'application puisque les aspects de gestion de l'environnement ont été délégués au serveur et container.

L'objectif est de permettre de travailler avec un produit serveur et un ou plusieurs containers provenant de fournisseurs différents. De même, des bibliothèques de composants applicatifs et métiers pré-utilisables pourront être utilisés dans le cadre des EJB. Ainsi, obtient-on une indépendance vis-à-vis des systèmes, des plates-formes, des serveurs objets transactionnels et des middlewares.

Nous détaillons par la suite les fonctionnalités qui sont associées à chacun de ces trois niveaux.

Le serveur

Le serveur de composants offre les services suivants :

- Mise en oeuvre de l'identification. Il permet aux clients EJB de localiser les services à travers l'interface de nommage JNDI (Java Naming Directory Interface) et de savoir si un type de Bean est disponible.
- Mise en oeuvre de l'authentification des clients en leur demandant de se connecter et fournir un identificateur pour accéder aux services. Il établit la connexion entre le client et le Bean.
- Mise en oeuvre de l'invocation par les clients sur les objets gérés dans le serveur. En particulier, il s'occupe de la bonne montée en charge de l'applicatif en distribuant, si besoin est, les traitements sur plusieurs machines. Il peut également gérer des transactions distribuées sur plusieurs serveurs ou encore les mécanismes de validation à deux phases.

Le serveur EJB tient un rôle de serveur d'applications. C'est à dire qu'il prend en charge l'environnement d'exécution pour le rendre transparent au client et optimiser l'utilisation de ses ressources. Les serveurs EJB peuvent inclure plusieurs containers.

Le container

Le container EJB, quant à lui, tient le rôle de serveur de données. Il gère la création des Enterprise Beans, leur stockage transparent en base de données ainsi que leur récupération, et peut également fournir un mécanisme de cache. Il revient au container EBJ de s'assurer de la bonne montée en charge de l'accès aux données.

Le container est un deuxième niveau dans l'environnement d'exécution des EJB. Il en contrôle l'exécution et leur fourni les services de niveau système. L'idée est de limiter ce type de service au sein des EJB pour se concentrer sur les développements propres à l'application.

Les containers offrent les services suivants aux EJB :

- la gestion de transactions simples ou distribuées. Il est possible de fixer le type de transaction utilisée.
- la sécurité avec pour support les *deployement descriptor* dans lesquels le rôle de chaque client est fixé. Ce rôle permet de définir les méthodes accessibles.
- la communication distante de manière transparente.
- la gestion du cycle de vie.
- des pools de connexion aux bases de données. Ces connexions étant coûteuses, elles sont préallouées et maintenues au delà de la durée de validité de l'EJB. Lorsqu'il en a besoin, l'EJB utilise une de ces connexions préallouées. Ceci permet de réutiliser une connexion d'un EJB à l'autre sans la réallouer à chaque connexion.

Les enterprise beans

Il existe deux types d'enterprise bean. Les *session beans* et les *entity beans*. Les *session beans* ont pour rôle de représenter le client dans le serveur. Leur vie est d'ailleurs liée à la session du client : ils ne peuvent avoir qu'un seul client et sont détruits à la fin de la connexion du client. Les *entity beans* représentent les objets d'un mécanisme de stockage persitant tel qu'une base de données. Cela signifie que l'entity bean est porteur d'un état qui peut être sauvegardé dans

la base. La mise en oeuvre de cette sauvegarde doit être réalisée par le programmeur.

2.3 COM/DCOM

Le modèle COM/DCOM est la contribution de Microsoft aux environnements à objets distribués. Il est issu des besoins d'interaction entre applications pour la mise en place de documents composés. A l'origine, ces besoins ont été résolus par les technologies OLE, basées sur les librairies dynamiques (DLL). Ces technologies ont été étendues pour prendre en compte de manière générale toutes les interactions entre les applications sous Windows, qu'elles soient utilisées sous la forme de bibliothèques, de serveurs locaux ou de serveurs distants. L'ensemble de ces technologies est regroupé sous la dénomination ActiveX. Dans cette technologie, COM/DCOM représente la partie support de communication que l'on pourrait assimiler à l'ORB dans CORBA. La présentation que nous en donnons ici est technique dans la mesure où la définition de COM n'a pas été réalisée à partir d'un modèle mais c'est plutôt le modèle qui a été dérivé à partir de la réalisation technique.

2.3.1 La communication

La technologie COM/DCOM reconnaît trois types de serveurs : les librairies dynamiques, les objets locaux exécutés dans des contextes d'exécution séparés et les objets distants. Pour chacun de ces objets, la mise en oeuvre de la communication est différente. Le modèle COM/DCOM définit un schéma d'accès, basé sur des interfaces, commun au trois types de serveurs. Dans ce cadre, COM est l'implantation supportant les communications locales, DCOM permet l'accès à des serveurs distants. Le schéma COM définit le format binaire des bibliothèques d'accès au serveur pour assurer les possibilités d'interaction entre les objets.

Dans le modèle DCOM, un objet serveur doit au moins supporter l'interface *IUnKnown* qui est nécessaire pour mettre en oeuvre l'implémentation de COM. En plus de cette interface, dénominateur commun, un objet peut supporter plusieurs interfaces qu'il définit. A chaque interface d'objet est alors associé un identifiant, le *GUID*. Comme dans CORBA, le client possède une référence de l'objet distant, basée sur le *GUID*. Il utilise cette référence pour invoquer l'objet. Les *GUID* sont mémorisés dans une base appelée le registre, sorte de serveur de nom.

L'interface *IUnKnown* propose trois fonctions :

QueryInterface qui permet de faire découvrir au client les interfaces supportées par l'objet. Elle sert donc de point d'entrée lors d'invocations dynamiques.

AddRef permet au serveur de maintenir le compte de ses clients. Cette méthode est appelée par le client chaque fois qu'il crée une nouvelle instance de référence. Si le compte de référence est nul, alors l'objet peut être déchargé de la mémoire.

Release informe le serveur que son client a libéré une instance de référence.

Nous voyons ici que le modèle défini par COM/DCOM offre le support à l'invocation dynamique. Il permet également, dans son modèle d'exécution, de lancer dynamiquement le serveur à la demande du client en utilisant des usines d'objets : lorsqu'un objet connaît le *GUID* du serveur auquel il veut accéder, il peut en demander directement la création au bus COM.

Celui-ci, après avoir recherché dans le registre la référence de l'objet concerné, sous-traite sa création et son lancement à une usine d'objets. L'usine rend alors à l'utilisateur une référence sur l'interface *IUnKnown* au client.

Contrairement à CORBA, il n'y a pas d'héritage possible entre les interfaces. Pour remplacer, le modèle propose d'utiliser des mécanismes d'agrégation et de délégation qui sont liés à la définition du modèle de composants défini par COM/DCOM.

2.3.2 Le modèle de composants

Il faut reconnaître que Microsoft a été le premier à mettre en place un véritable modèle de composants à travers COM avec l'idée explicite de décomposer les applications en parties indépendantes qui interagissent. Le problème de ce modèle réside dans le fait qu'il n'était pas distribué et limité à l'environnement Windows, donc très propriétaire.

Le choix de refuser l'héritage d'interface, qui a été fait dès l'origine par Microsoft, est justifié par le fait que la composition d'objets est une technique qui respecte bien la notion d'encapsulation et de modularité alors que l'héritage complexifie grandement la réutilisation des objets. En effet, il est souvent nécessaire de connaître l'implantation d'une classe mère pour l'utiliser correctement, ce qui ne l'est pas lorsqu'on y accède par une interface. D'autre part, l'application qui utilise une classe mère ne supporte pas forcément une évolution de celle-ci, ce qui est possible si les implantations sont distinctes. Ces problèmes font qu'il n'est pas facile d'utiliser les classes comme des "boîtes noires" pour en hériter. Pour permettre une mise en oeuvre d'applications basées sur des composants réutilisables, COM parie sur l'agrégation et la délégation. La figure **??** montre ces deux schémas de composition entre objets.

FIGURE 2.4 – Agrégation et délégation dans DCOM

L'agrégation est une composition d'objets qui n'offre qu'une seule interface. C'est un ensemble d'objets réunis en un seul. Dans ce cas, l'un d'entre eux sert de point d'entrée à l'ensemble. C'est lui qui porte l'interface *IUnKnown* et tient le compte des références. De cette manière, le client ne voit qu'un seul objet.

La délégation est l'utilisation d'une même interface par des objets indépendants. Il n'y a pas de regroupement des objets qui sont indépendants. Par contre, pour une même interface, différents objets peuvent rendre le service.

Le modèle COM/DCOM est très lié à la plate-forme Windows puisqu'elle a longtemps été la seule supportée. Néanmoins, Microsoft ayant compris l'intérêt que pouvait présenter une acceptation plus large de son modèle, un portage a été entrepris vers les plates-formes Unix.

Nous donnons en **??** un jugement de valeur sur cette réalisation. Notons tout de même que, si le langage C++ a été le premier langage supporté par COM, le langage Java l'est maintenant aussi. Celui-ci apporte un certain nombre d'avantages, en particulier, le fait de ne plus avoir à gérer le nombre des références objets puisque celui-ci est pris en charge par le ramasse-miettes de Java.

2.4 Comparaison des environnements à objets

Dans un souci de veille technologique, nous avons cherché à analyser les différents environnements à objets distribués. Une étude exhaustive semblait coûteuse en temps et en moyen. D'autre part, un certain nombre de travaux similaires sont disponibles dans [**?**], il n'était donc pas nécessaire de recommencer ces travaux. Nous ne cherchons en aucun cas à dégager un gagnant mais à voir à quels environnements et type d'applications sont adaptés chacun des environnements.

2.4.1 Comparaison fonctionnelle

Cette comparaison fonctionnelle cherche à mettre en regard les différentes fonctionnalités des environnements que nous avons décrits précédemment.

La première constatation qui peut être faite, c'est que la concordance est forte, au moins en ce qui concerne le support à la communication. Chacun propose des communications transparentes entre le client et le serveur, qui se traduisent en pratique par l'invocation distante du serveur par le client à partir d'une référence d'objet et de bibliothèques de mise en forme. Quels sont alors les critères de différenciation entre les environnements ? Si la communication est centrale dans la définition de la norme CORBA, c'est plutôt le modèle de composants qui est mis en avant dans DCOM (la distribution est un ajout au modèle) et l'intégration de l'ensemble des services dans un même langage pour Java. La comparaison n'est donc pas aisée et il est évident que les points forts d'une spécialité deviennent des points faibles de la généralité. Nous avons donc essayé de dégager pour chacun ses points forts et ses points faibles en regard des autres.

Pour ce qui est de CORBA, le fait d'être une norme ne le place pas sur un pied d'égalité directe avec les deux autres. L'hétérogénéité et l'étendue des services définis sont assurement ses points forts. L'hétérogénéité est, d'une part, prise en compte dans la norme et, d'autre part, implicitement mise en oeuvre avec les différentes implantations d'ORB. CORBA constitue, de ce fait, un environnement adapté à la mise en oeuvre d'applications ayant une base applicative préexistante. L'ensemble des services défini par CORBA est le plus large et s'enrichit à mesure que la norme cherche à répondre à de nouveaux besoins. Un des points faibles de CORBA est son indépendance par rapport à des environnements de développement plus complets. Le modèle de composants de CORBA est en cours de définition et risque de manquer de maturité pendant quelques années.

Pour Java, sa principale qualité est la portablité. Comme CORBA, avec lequel il a échangé bon nombre de concepts : interfaces, composants, invocations distantes, etc. Java est un environnement très riche qui ne cesse d'évoluer. La spécification de l'interaction entre Java et activeX est en cours et elle est déjà réalisée avec CORBA puisque le schéma conceptuel des

EJB intègre l'accès à des composants CORBA. Par contre, les invocations à distance RMI sont moins complètes que celles de CORBA au niveau des invocations dynamiques mais est java mieux intégré dans le langage donc plus simple.

La plate-forme DCOM est plus éloignée des autres, c'est la plus fermée puisque, même si Microsoft tend à la rendre accessible, techniquement elle reste la plus complexe à appréhender. Nous avons essayer d'en évaluer les performances sur un système Linux, à partir du portage réalisé par la société Software AG. Malheureusement, la distribution ne permet pas une utilisation en distant et la programmation C++ est très difficile à maîtriser. Pour cette raison, nous n'avons pas inclu DCOM dans les comparaisons de performances que nous présentons dans la partie suivante.

2.4.2 Performances de différents ORB

Comme je l'ai dit dans l'introduction, nous avons cherché à utiliser les services d'un ORB pour la mise en oeuvre d'applications parallèles et d'un courtier de calcul haute performance. Dans le cadre de ces projets, il a été nécessaire de comparer les propriétés et les performances de différents ORB. Dans cette étude, nous n'avons retenu que des ORBs disponibles gratuitement car cette démarche était plus en accord avec la nôtre. Les tests de performances ont été réalisés en local et entre deux PC (Pentium II 450 Mhz) sous Linux. Il s'agit d'une invocation réalisant l'envoi d'une séquence de caractères dont nous faisons varier la taille :

temps en ms	10	100	1000	10000
Mico	0.37	0.39	0.41	0.67
OmniOrb2	0.2	0.2	0.26	0.51
OrbAcus C++	0.3	0.3	0.34	0.79
OrbAcus Java	1	1	5	60
Jonathan	0.88	0.98	2.3	16
JDK Sun	0.81	0.96	1.28	4.5

TABLE 2.1 – Performances de différents ORBs en local

temps en ms	10	100	1000	1000
Mico	0.65	0.73	1.55	9.83
OmniOrb2	0.56	0.68	1.4	9.81
OrbAcus C++	0.57	0.67	1.54	10.26
OrbAcus Java	1	2	6	47
Jonathan	1.06	1.23	3.37	24.3
JDK Sun	1	1.26	2.26	12.7

TABLE 2.2 – Performances de différents ORBs en distant

D'autre part, il existe des études telles que [**?**] comparant différents ORB. Ces études font état de rapports de performances, par rapport à OmniOrb2, de l'ordre de 5 pour Orbix et de 2 pour Visibroker, pour l'échange de messages courts. Pour les messages de taille plus importantes,

les rapports diminuent, toujours en faveur d'OmniOrb2 mais avec des rapports de 1,2 avec VisiBroker et 2,5 avec Orbix.

De ces différentes études, nous pouvons conclure au bon comportement d'OmniOrb2 et d'OrbAcus et, si on exclut Orbix, des ORB utilisant le C++ par rapport à ceux qui utilisent Java.

2.5 Les projets

Les plates-formes de composants que nous avons décrites précédemment ne sont pas complètement satisfaisantes du point du vue de la gestion de l'exécution des applications distribuées. Leurs fonctionnalités couvrent les communications entre les applications (CORBA, RMI) en rendant l'accès aux composants transparent et uniforme, le support d'exécution des composants distribués (EJB, CCM, DCOM) en prennant en charge la gestion de bas niveau, etc. Elles permettent d'améliorer la productivité de développement en soulageant le programmeur de leur implantation. Mais ces plates-formes proposent peu de support pour la phase d'exécution. En effet, les schémas d'exécution restent très statiques alors que les environnements eux-mêmes ne le sont plus. Pour ces raisons, plusieurs projets sont en cours pour étendre les plates-formes de composants à la prise en compte de leur besoins durant la phase d'exploitation : reconfiguratoin dynamique, répartition de charge, utilisation dans l'informatique mobile ou dans les cartes à puces, etc.

Le projet SIRAC s'intéresse à la construction d'applications réparties adaptables avec un lien privilégié vers les composants. Les sujets abordés concernent la migration d'objets, l'adaptation et la reconfiguration dynamique d'applications. Dans le cadre de ce projet, la définition de la plate-forme OLAN[**?**] a visé à la gestion des composants depuis la conception jusqu'à l'exploitation. Le développement est facilité à travers la mise en place d'outils d'aide à la conception, de compilateurs pour la génération automatique de code et d'outils de composition de composants. Le déploiement et la reconfiguration dynamique des applications sont facilités par l'utilisation d'une console graphique. De manière à garantir la cohérence de l'application pendant son déploiement et en cas de reconfiguration, une description ADL (Architecture Definition Langage) lui est associée.

Le projet GoodeWatch [**?**] propose une infrastructure pour la supervision d'applications CORBA. Le but est d'observer le comportement des applications, ou des composants, à partir d'informations collectées par l'ORB pour permettre aux utilisateurs de les contrôler. La mise en oeuvre repose sur la génération d'événements par le système. Ces événements servent ensuite de base pour déclencher les réactions nécessaires.

D'autres projets sont en cours à travers des collaborations entre différentes équipes. C'est le cas du projet SAMOA qui vise à la gestion de composants dans les environnements d'informatique mobile ou du projet ARCAD qui vise à la gestion de la reconfiguration de composants adaptables (la description de l'architecture est basée sur l'ADL) dans des environnements extensibles.

2.6 Conclusion

Les technologies à objets sont actuellement en plein essort car elles apportent une solution plus adaptée que la programmation classique aux problèmes de modularité, de réutilisation ou de maintenance. Leur utilisation dans des environnements distribués tels que l'internet à conduit les industriels et les organismes de normalisation à définir des supports de communication pour permettre une interaction entre objets distants. Une des évolutions de cette technologie est actuellement la définition de modèles de composants permettant le paramétrage et une grande réutilisabilité du code. Cette utilisation suppose la mise en place d'un support d'exécution adapté à ces composants.

Chapitre 3

Le placement dynamique et son optimisation

L'origine des travaux sur la répartition de charge se trouve dans l'utilisation des machines sous-chargées, lorsque les performances des ordinateurs étaient suffisamment faibles et leur prix suffisamment élevé pour essayer d'en maximiser l'utilisation. Des études telles que [**?**] montrent, par exemple, qu'environ un tiers des stations de travail d'un réseau sont pratiquement inactives aux heures où les machines sont le plus utilisées. La baisse du prix des ordinateurs et l'augmentation de leurs performances ont limité ce besoin de partage, d'autant plus que les utilisateurs étaient rétissants à partager leur bien. Les recherches se sont ensuite orientées vers les multiprocesseurs à mémoire distribuée mais, comme nous l'avons dit, celles-ci n'ont pas eu l'essort commercial qui était attendu et les programmeurs spécialisés préfèrent gérer manuellement - par répartition dynamique des données par exemple - la charge de la machine.

Cependant ces évolutions n'ont pas rendu obsolètes les efforts et les recherches réalisés en faveur de la répartition de charge. En effet, la baisse de prix des réseaux à haute vitesse permet aujourd'hui d'envisager des serveurs composés d'ensemble d'ordinateurs (clusters)à bas prix. Ces serveurs hébergent des applications sous forme de composants pour le compte des utilisateurs. La répartition de ces composants sur l'ensemble des machines étant une fonction dynamique, elle ne doit et ne peut pas être réalisée par le développeur, elle doit faire partie de l'environnement d'exécution. D'autre part, un service d'optimisation globale doit être implanté au niveau du système pour assurer l'optimisation des échanges entre les composants et éviter la surcharge sur le bus local.

Toutes ces raisons font que le placement dynamique est encore d'actualité et, s'il ne se fait plus sur des processus comme aux premiers jours, il est d'autant mieux adapté aux composants qui sont plus indépendants du système et d'une granularité plus fine. Ainsi la plupart des résultats obtenus restent valident et applicables au monde des composants logiciels.

Dans ce chapitre, nous donnons un aperçu des travaux existants dans le domaine du placement dynamique et des procédures utilisables pour optimiser les décisions de placement. Ce thème est le point de départ de nos travaux sur l'administration de composants. La vue en reste volontairement très synthétique. On trouvera des présentations plus complètes dans [**?**, **?**].

3.1 Le placement dynamique

La plupart des travaux proposés dans la littérature pour optimiser dynamiquement l'allocation des ressources dans les systèmes répartis s'inscrivent dans une thématique «équilibrage de charge», ou charge sous-entend charge des processeurs. Des travaux récents introduisent d'autres ressources, en particulier, la mémoire et le réseau de communication. Les mécanismes de gestion de ressources reposent sur un certain nombre de principes généraux. Nous étudions ces principes à la lumière de facteurs de performance et de propriétés requises pour un mécanisme d'allocation de ressources.

Dans cette partie, nous étudions les critères d'efficacité et les propriétés des systèmes de gestion des ressources. Ceci nous permet ensuite de présenter les principes des algorithmes sur lesquels ils sont basés et de comparer plusieurs approches proposées dans la littérature.

3.1.1 Les caractéristiques d'un système d'allocation de ressources

Le but d'un système d'allocation de ressources est double : permettre une exécution plus rapide des applications, par exemple en plaçant une entité sur un ordinateur peu occupé, et optimiser l'allocation des ressources d'une manière globale, par exemple en plaçant deux entités qui communiquent beaucoup sur le même site pour limiter leur utilisation du réseau. Ces deux buts concourent à fournir un meilleur confort à l'utilisateur qui se traduit, à son niveau, par plus d'efficacité. Il est donc envisageable de qualifier le bénéfice obtenu par l'allocation en ressources en terme d'efficacité.

Les critères d'efficacité

Pour définir l'efficacité, nous nous appuyons sur la notion de temps de réponse d'une application. Le temps de réponse d'une application est, du point de vue de l'utilisateur, le temps écoulé entre le lancement d'un traitement et l'obtention d'un résultat.

L'objectif de la gestion de ressources est d'améliorer, globalement, les temps de réponse des applications. Cette amélioration globale est difficile à évaluer. Il s'agit d'une minimisation de la moyenne et de la variance des temps de chaque application. Minimiser la moyenne signifie que globalement le temps de réponse d'une application est plus court. Minimiser la variance signifie que deux exécutions d'une même application auront sensiblement le même temps de réponse. Du point de vue de l'utilisateur, cela se traduit par une sensation de confort. Du point de vue du système, cela signifie une meilleure utilisation des ressources qui permet de mieux satisfaire les besoins des applications.

La mise en oeuvre de mécanismes de gestion de ressources est consommatrice de ressources, et est en ce sens antinomique à ses objectifs ! Il tombe sous le sens que la gestion de ressources ne doit pas pénaliser les applications s'exécutant sur le système. L'efficacité d'un tel mécanisme se mesure en fonction de ses coûts en terme d'échange de messages et d'utilisation des ressources vis à vis des optimisations qu´il permet de réaliser.

Les principes et propriétés d'un mécanisme de gestion de ressources

Un système d'allocation de ressources a pour objectif de prendre une décision de la forme : quelle(s) entité(s) faut-il placer ou déplacer vers quelle(s) ressource(s) pour augmenter les performances globales du système ? Ce processus de décision se décompose en un ensemble de services qui peuvent être abordés indépendamment les uns des autres [**?**]. Ces composants sont au nombre de trois : le module d'information, le module de décision et le module de transfert. Nous les détaillons dans les parties suivantes.

La politique de répartition de la charge définit les méthodes et les moyens utilisés pour optimiser les performances globales du système. Une première distinction apparaît entre le partage et l'équilibrage de charge. Le partage de charge vise à éviter d'avoir des sites surchargés sur le réseau. L'équilibrage de charge tente de maintenir une charge identique sur tous les sites du réseau. Cette dernière permet théoriquement une diminution de la variance des temps de réponse. Par contre les traitements qu'elle induit sont plus coûteux que ceux induits par le placement. En effet, le partage de charge ne repose que sur le placement d'entités lors de leur création alors que l'équilibrage nécessite le déplacement d'entités au cours de leur exécution.

Nous énumérons ci-dessous une liste non exhaustive des propriétés que peut ou doit avoir un mécanisme de gestion de ressources[**?**].

1. **la sûreté** : la prise en charge d'une application par le mécanisme de gestion de ressources ne doit pas avoir d'incidence sur son résultat.
2. **la transparence** : un utilisateur, qui le souhaite, ne doit pas voir de différence entre une exécution sans gestion des ressources et une exécution avec gestion des ressources.
3. **l'extensibilité** : le mécanisme de gestion de ressources doit pouvoir prendre en charge un système avec un grand nombre de sites, sans modification sensible de son comportement.
4. **la stabilité** : le système de gestion des ressources ne doit pas perpétuellement déplacer une entité sans lui laisser le temps de s'exécuter. Par extension, deux sites ne doivent pas s'envoyer réciproquement de la charge en même temps.
5. **l'équité** : il n'est pas souhaitable qu'une unique application monopolise une ou des ressources, empêchant ainsi les autres applications de s'exécuter.
6. **la tolérance aux pannes** : le système de gestion de ressources ne doit pas mettre en péril l'exécution des applications si des pannes apparaissent sur des sites ou des liens de communication.
7. **la sécurité** : les entités d'exécution, éventuellement déplacées, doivent conserver les mêmes droits et le même niveau de sécurité.
8. **l'hétérogénéité** : généralement, l'ensemble des ressources d'exécution d'un réseau n'est pas homogène. Le mécanisme de gestion des ressources doit permettre de manipuler de façon uniforme l'ensemble de ces ressources, permettant ainsi aux applications d'interopérer malgré l'hétérogénéité.

Il semble illusoire, en l'état, de vouloir bâtir un système d'allocation de ressources disposant de l'ensemble de ces propriétés. Par contre, il est possible d'évaluer les solutions disponibles en fonction du nombre de critères satisfaits et/ou du degré de satisfaction de chaque propriété.

3.1.2 La politique d'information

La politique d'information spécifie les informations utilisées pour déterminer un ou plusieurs critères locaux à un site, et comment, ce ou ces critères sont transmis aux autres sites pour définir une image plus ou moins précise, dans le temps et dans l'espace, de l'état du système. L'image du système la plus précise est l'état global : la construction à un instant T en un même endroit d'une représentation de l'état exact de toutes les machines du réseau. La moins précise est la prise en compte d'aucune information.

Choix des informations

L'objectif de cette partie du service est de fournir un indicateur décrivant l'état d'une machine (un site) à chaque instant. La plupart des algorithmes utilisent une seule variable, c'est à dire qu'ils tentent d'agréger en un seul critère l'ensemble des paramètres décrivant l'état local. D'après [?] un indicateur de charge doit effectuer un calcul fiable de la charge et ne pas dépendre d'épiphénomènes. Il doit de plus permettre une prédiction, au moins à court terme, de l'état du système. Certaines études [?, ?] insistent sur le fait que des techniques simples, à base de peu d'informations, donnent des résultats proches de techniques complexes. Notons cependant que les entités gérées (ou simulées) sont généralement des processus simples qui ne communiquent pas entre eux. Il n'est pas vérifié que ces mesures puissent également s'appliquer aux objets.

La majorité des travaux estiment la charge d'une machine en fonction de la disponibilité du processeur, soit en se basant sur la longueur des files d'attente du système, soit en mesurant l'accessibilité au processeur via une «sonde»[?], processus qui calcule l'intervalle de temps entre deux accès au processeur : plus cet intervalle est grand, plus la machine est chargée. Notons que, dans un réseau de machines hétérogènes, la charge d'un processeur ne peut être comparée directement à celle d'un autre. Il convient donc d'appliquer une transformation tenant compte des puissances relatives de ces deux machines [?].

D'autres données peuvent être utilisées telles que la disponibilité mémoire ou la charge du réseau. Cependant, il n´y a pas de corrélation entre le temps d´utilisation du processeur par un processus et son encombrement mémoire [?]. En ce sens, la place mémoire est moins une qualification de la charge qu´un veto à opposer à une migration potentielle. D'autre part, la charge du réseau de communication est rarement utilisée dans les mécanismes d´équilibrage de charge car il est difficile d´extraire de la charge du réseau la part jouée par le site local.

La distribution des informations

Le temps a une grande influence sur la validité d´une information. Un critère est obsolète dès qu´il est connu, puisque l´état du système a évolué pendant sa détermination. Ceci implique que les critères doivent être, soit calculés souvent, en accord avec la variabilité du système qu'ils décrivent, soit être calculés à la demande, c´est-à-dire au moment où ils vont être utilisés.

De nombreuses stratégies ont été proposées pour constituer un ensemble d´informations susceptibles d´aider à la prise de décision [?], depuis la prise en compte d´aucune information sur les sites distants (méthode autonome), jusqu´à la constitution d´un état global centralisé. Nous présentons ce mécanisme à partir de quatre couples de propriétés qui permettent de

décrire la majorité des approches. L'acquisition des informations peut être :

distribuée / centralisée : une collecte centralisée hypothèque grandement l´extensibilité du mécanisme car elle est peu résistante aux pannes et génératrice de goulets d'étranglement. Par contre, la centralisation de toutes les informations permet une prise de décision quasi optimale. Notons qu'il existe des solutions alternatives, à base de gérants de sous-réseaux qui centralisent la collecte, sur des structures de petite taille et utilisent une méthode distribuée entre eux [**?**, **?**].

totale / partielle : les informations distantes sont issues de tous les sites ou seulement d´un sous-ensemble des sites. Les méthodes à acquisition totale sont fortement dépendantes de la taille du système et, de ce fait, sont peu extensibles. L´acquisition partielle nécessite, pour chaque site, la définition d´un sous-ensemble des sites du système, chargé de lui fournir des informations.

à la demande / périodique : l´acquisition des informations distantes est coûteuse en communications. Il convient de minimiser le nombre de ces échanges tout en maintenant la pertinence des informations. L´acquisition pourra se faire sur demande, à l´initiative du récepteur ; de façon périodique ou sur événement [**?**], à l´initiative de l´émetteur.

directe / indirecte : les informations peuvent être obtenues directement depuis la machine concernée ou via une machine tierce. L´intérêt de l´information indirecte repose sur la diminution du nombre de messages émis [**?**] mais s'accompagne de son vieillissement.

3.1.3 La politique de décision

La politique de décision détermine si le site est surchargé, quelle entité doit être déplacée et vers quel site. La réponse à l´une de ces questions influence, voire détermine, les réponses à apporter aux autres.

Initialisation de la procédure

L'initialisation de la procédure de décision est, généralement, basée sur l'utilisation d'un ou deux seuils. Dans le cas d'un seuil unique [**?**], un placement ou déplacement est envisagé lorsque la charge locale d'un site dépasse ce seuil. Une méthode utilisant deux seuils [**?**, **?**] offre un meilleure stabilité en évitant de passer sans transition d'un état où le site accepte de la charge, à un état où il désire en donner. La définition de ces seuils peut alors être réalisée statiquement de façon empirique ou dynamiquement en fonction de la charge moyenne du système. La première méthode a pour avantage sa simplicité, mais n'est réellement adaptée qu'à un régime spécifique du système. La deuxième méthode, plus souple que la première, implique de déterminer une valeur approchant de la charge moyenne du système.

Choix de l'entité

Le choix de l´entité à déplacer repose sur le site émetteur. [**?**] définit le résidu d´exécution comme le temps nécessaire à une entité pour terminer la tâche qui lui incombe. Les entités qui profiteront le plus d´un déplacement, sont celles dont le résidu d´exécution est grand devant le temps de réalisation de ce déplacement [**?**]. Le paradoxe de la vie résiduelle indique qu´il

faut choisir les entités les plus vieilles : un processus ayant vécu T unités de temps a 40% de chance d´avoir une durée de vie de $2T$ unités temps. Le choix de l´entité à déplacer peut aussi être fait en accord avec des informations concernant le site récepteur et le comportement de l´entité. En particulier, l´optimisation des communications est importante [**?**, **?**, **?**]. Par exemple, pour un système réparti à objets, les objets serveurs sont de bons candidats pour un éventuel déplacement. En effet, théoriquement, ils ne se termineront qu´au moment de l´arrêt du système. Ils disposent donc d´un résidu d´exécution important qui doit permettre l´amortissement du coût du déplacement.

Sélection du destinataire

Dans le cas des politiques autonomes, un site s´estimant surchargé sélectionne un site pour l´exécution de son excédent de charge sans tenir compte de son état passé ou courant. C'est le cas du placement aléatoire définit dans [**?**, **?**]. Cette politique, efficace pour des systèmes faiblement chargés, devient instable et coûteuse lorsque le système monte en charge. Pour le placement cyclique, utilisé par exemple dans PVM [**?**], le choix de la destination n´est plus aléatoire : les nouvelles tâches sont attribuées aux sites du système dans un ordre déterminé.

La constitution, sur un ou plusieurs sites, d´une représentation, globale ou partielle, de l´état du système, a pour rôle de permettre une prise de décision raisonnable. Elle est à la base des algorithmes de placement coopératifs. L´initiation du processus de décision repose sur la détection d´une optimisation possible de l´environnement d´exécution. Cela peut être une différence de charge importante entre deux sites, ou la consommation par une entité d´une ressource distante. La structuration du réseau d'exécution en groupe de processeurs permet, en entrant dans le processus de décision, de prendre en compte des informations sur l'architecture du réseau en même temps que sur la charge [**?**].

Le premier type d'algorithme obtient généralement de bonnes performances en regard de la simplicité de la mise en oeuvre. Notons cependant qu'il ne parvient pas à l'optimum alors que les méthodes coopératives s'en approchent.

Conduite de décision

La décision de déplacement peut être conduite par un site ou par une des entités génératrices de la charge.

Lorsque qu'un site, du fait des informations dont il dispose, constate qu´il est sur ou sous-chargé, il initie la recherche d´un autre site pouvant céder ou recevoir de la charge. La littérature distingue généralement le cas du site initiateur sous-chargé, où la décision est initiée par le récepteur, du site initiateur surchargé, où la décision est initiée par l´émetteur. Le premier étant plus adapté aux environnements chargés et le second aux environnements sous-chargés [**?**].

Une décision de placement ou de déplacement peut également être initiée par une entité, soit lors de son lancement, soit si le système dispose d´un mécanisme permettant d´observer son comportement. Dans le cadre du placement, à chaque arrivée de tâche, il faut définir le site où cette dernière doit être placée. Des informations peuvent éventuellement être maintenues sur les précédentes exécutions de ce type de tâche pour en déduire un placement approprié. Dans le cas où le système a la possibilité de détecter qu´une tâche profiterait d´une migration, par

observation des communications par exemple [?, ?, ?, ?], le site cible est implicitement celui avec lequel le flux de communication est le plus intense, et le site source est celui sur lequel l´entité est exécutée.

3.1.4 La politique de transfert

La politique de transfert a pour rôle de mettre en oeuvre les mécanismes appropriés de transmission de la charge en fonction des décisions issues de la politique de localisation.

Le placement

Le placement consiste, lors de l´arrivée (création) d´une tâche, à l´affecter à un site. Cette fonction est disponible dans la plupart des systèmes, sa mise en oeuvre est simple et son coût est faible. Par contre, le placement ne permet plus, une fois la tâche affectée, de modifier son site d'exécution. Cette solution est actuellement souvent employée en conjonction avec la migration. Réaliser un bon placement initial permet d´éviter de devoir trop recourir à la migration.

La migration

La migration permet de déplacer une tâche au cours de son exécution. Cette fonction, plus souple que la précédente, induit plusieurs problèmes : sa disponibilité, le sur-coût qu´elle engendre et la complexité de sa mise en oeuvre. La migration n´est disponible, d'origine, que sur très peu de systèmes, et la construction d´un mécanisme de migration reste un travail complexe [?]. Le sur-coût induit par l´utilisation de la migration et son impact sur les performances ont été largement étudiés [?, ?, ?]. La migration d´objets est facilitée par la nature même des objets. L´état est clairement définit par les valeurs de l´ensemble des attributs déclarés dans la classe d´un objet. Aussi, la sauvegarde d´un état est facilitée. Le gel de l´exécution d'un objet reste identique à celui d´un processus. Le service de mobilité (migration) proposé par la norme CORBA n´est pour l´instant applicable que pendant les phases d´inactivité des objets.

La duplication

La duplication ne permet pas, à proprement parler, d´économiser des ressources. Mais, dans le contexte d´une approche client/serveur, elle permet de multiplier les ressources logicielles. Considérons le cas d´un serveur dont le temps de traitement d´une requête est important. La multiplication du nombre de requêtes émises par des clients peut dépasser ses capacités de traitement, ce qui allonge la file d'attente des requêtes en attente de traitement. Dans le cas d´un protocole de communication synchrone, cela peut être dommageable pour les performances. Une solution consiste à dupliquer ce serveur, sur un autre site, et à partager les invocations entre les deux instances du même service. Cette solution ne va pas sans soulever un certain nombre de problèmes : comment rediriger les requêtes en souffrance, comment assurer la cohérence des deux ou N serveurs et à quel prix, dans quelles conditions supprimer des réplicats pour minimiser la consommation de ressources.

Le problème de la duplication est étudié dans le domaine de la tolérance aux fautes. Les problèmes de redirection et de cohérence trouvent des solutions. Par contre, la suppression des serveurs devenus inutiles du fait de la baisse de régime du système, qui est une contrainte propre à l´optimisation de l´allocation de ressources, n´est que peu développée. Une solution consiste à bâtir chaque service comme un groupe [?], même s´il ne contient au départ qu´une seule instance de serveur. Lorsqu´un serveur est surchargé, il demande la création d´une nouvelle instance dans le groupe, puis sort du groupe. Lorsque sa charge atteint un seuil suffisamment bas, il peut, soit réintégrer le groupe, soit terminer son travail puis s´arrêter. L'étude [?] est faite avec l´hypothèse de serveurs ne traitant que des requêtes indépendantes, sans maintien d'état.

3.1.5 Des exemples de systèmes de gestion de ressources

Le début de cette partie présente thématiquement les différents éléments constitutifs des processus d´optimisation de l´allocation de ressources. Cette troisième partie est dévolue à une étude de cas. Nous y présentons plusieurs études que nous avons sélectionnées pour leur connexité avec notre problématique. En particulier, nous nous sommes limités aux études basées sur des systèmes répartis à objets. Les systèmes répartis à objets, plus «jeunes» que les systèmes à base de processus, n´ont pas été très étudiés du point de vue de la gestion de l´allocation des ressources. Nous présentons, par la suite, les principales études dans ce domaine. Les trois différences principales avec les approches à base de processus se situent au niveau de la taille des entités, de leur nombre et de l´importance des communications. Les objets sont des entités de petites tailles comparés à des processus. Les fonctions, mises en oeuvre dans un processus, sont réparties entre plusieurs objets, et ils utilisent énormément les communications pour collaborer à la réalisation d´une tâche globale définie au niveau de l´application.

Les hypothèses faites dans le domaine des processus ne sont pas satisfaisantes dans le cadre des applications objets. En effet, à un instant donné, il y a une multitude d'objets qui forment un certain nombre d'applications indépendantes. Bien qu'indépendantes, ces applications se partagent certains des objets présents. Le coût d'une application est directement en relation avec la distribution des objets qui la composent. De plus le modèle d'exécution utilisé, que ce soit dans Java, CORBA ou Dcom suppose des objets serveurs dont la durée de vie ne peut être comparée avec celle de processus principalement destinés à exécuter des commandes. Un objet serveur est mis en place en tant qu'application sur un site et est activé à chaque invocation de la part des clients.

L'équilibrage dans Guide2

ELVIS est le nom du projet mené dans l´équipe Bull/IMAG pour réaliser un gestionnaire de charge [?] dans l'environnement Guide2 [?]. Ces travaux s´appuient d'une part sur les résultats d´une étude sur la granularité des objets [?], et, d'autre part, sur des études menées dans le cadre de l´équilibrage de processus.

Dans Guide, l´équilibrage est fait au niveau des clusters. Le cluster est le grain juste au-dessus de l´objet. Il est constitué d´un ensemble d´objets fortement corrélés. Son grain reste sensiblement inférieur à celui d´un processus. L´évaluation du coût d´exécution est basée sur

la charge des processeurs. Chaque site dispose d'informations concernant la charge sur tous les sites du réseau. Cette approche est très peu différente de ce qui est réalisé dans les systèmes à base de processus.

Bellorophon

L'approche décrite dans [?] consiste en un équilibrage de charge dynamique, au niveau système, indépendant de la topologie et permettant d'éviter les gros déséquilibres de charge. Le modèle objet présenté est délibérément simple, il assure que la stratégie proposée est largement applicable. Cette étude s´articule plus particulièrement autour des trois points suivants : l'utilisation de plusieurs mesures locales, la restriction du domaine de comparaison à un sous-ensemble (budy) et la détermination de *clump* d'objets (bouquet d'objets) candidats au transfert.

Il y a trois types de mesures : celles qui sont en relation avec l'espace (nombre d'objets sur un noeud), celles qui sont en relation avec le temps (nombre de messages reçus ou envoyés depuis un noeud en une seconde), et la troisième catégorie qui contient les informations liées au réseau, qui varient dans le temps et dans l'espace (contentions au niveau de certaines interconnections). La comparaison de ces mesures de charge reste difficile du fait de l'hétérogénéité des ressources présentes sur les différents sites.

Dans le cas où des comparaisons sont réalisées, l'auteur propose un système de voisinage qu'il nomme "buddy set" (ensemble de copains) de diamètre faible. Un certain nombre de membres du voisinage doivent être choisis parmi les sites ayant beaucoup de communications avec le site de base de l'ensemble.

L'un des problèmes majeurs, dans l'équilibrage de charge basé sur les objets, est de trouver un ensemble d'objets (clump) à déplacer pour équilibrer la charge. En effet, il faut construire ces ensembles pour obtenir une granularité qui soit en relation avec le coût de l'opération de migration. Il faut s'assurer que les objets regroupés puis déplacés ne vont pas réduire les performances du système en engendrant des sur-coûts de communication. L'auteur propose de calculer un ratio entre le nombre de communications réalisées à l'intérieur du clump et le nombre de communications qui franchissent la frontière du clump (communication depuis l'extérieur ou vers l'extérieur). Ce ratio peut être vu comme un coefficient de cohésion du clump. Le système Bellerophon introduit également la notion de «colocators» et de «contralocators». Deux objets qui contiennent un colocator l'un pour l'autre, doivent être toujours sur le même site. Par opposition, deux objets possédant un contralocator l'un pour l'autre ne peuvent pas être sur le même site.

Computational Field Model (CFM)

Le «Computational Field Model» [?] est le résultat d´une analogie entre les forces d´interactions d´un espace physique réel et l´utilisation de ressources dans un environnement informatique. Les unes comme les autres peuvent être modélisées par des attractions et des répulsions. Le principe du CFM est donc d´associer, à chaque consommation de ressource, une force d´attraction ou de répulsion. Ces forces permettent de calculer pour chacun des objets du système où se trouve sa position d´équilibre. En discrétisant les résultats, il est possible de déduire un placement.

Le système Muse a été développé, sur ce principe, dans les laboratoires Sony. Les informations utilisées sont les débits de communication et l'activité des objets. Des travaux sont menés pour tenter de définir une métrique pour l'espace dans lequel évoluent les objets.

L´équilibrage dans Gothic

Gothic [?, ?] est un projet réalisé dans le cadre d´une collaboration INRIA/BULL. La spécificité de ce système est de distinguer le site de stockage d´un objet et le site d´exécution de ses méthodes. Le placement dans Gothic [?] doit donc résoudre un double problème : où placer les fichiers décrivant les objets et où placer les exécutions de ses méthodes. L´étude propose un placement initial des objets sur le site de création en accord avec les résultats obtenus dans [?]. Le positionnement de l´objet est ensuite adapté en fonction des interactions avec les autres objets présents dans le système.

L'équilibrage de requête

Une des solutions les plus employées dans les systèmes répartis à objets pour éviter la surcharge des serveurs est l'équilibrage de requêtes. Nous avons déjà évoqué cette possibilité en ?? et les travaux évoqués en [?] proposent une architecture pour la gestion de la charge des serveurs. Les infrastructures normalisées évoquées au chapitre ?? proposent également la mise en oeuvre de cette fonctionnalité. La centralisation des requêtes en un point (le serveur d'application) permet de répartir - au mieux puisque la connaissance est totale - les requêtes entre les différentes instances d'objets ou de composants. La répartition peut alors aussi bien concerner une seule machine qu'un cluster.

3.2 L'optimisation

Pour représenter la distribution des applications telles que nous les avons définies au chapitre ?? et la politique de mobilité qui doit leur être appliquée afin d'obtenir une exécution optimale, plusieurs modèles mathématiques peuvent être appliqués. Un modèle est défini comme une représentation du réel pour se rapprocher le mieux possible du problème posé.

3.2.1 Un modèle d'environnement

Si la modélisation mathématique peut apporter une aide à la formalisation de notre problème, elle reste insuffisante pour en tirer des résultats. En effet, la part due au hasard et le nombre de paramètres qui régit les systèmes répartis est bien trop important pour tirer une solution analytique du problème. De même, la modélisation par files d'attente permet, comme nous l'avons déjà pratiqué, de simuler une partie du système mais jamais celui-ci dans son intégralité. La voie de l'expérience est donc la seule qu'il nous reste pour étudier le comportement d'un algorithme et la définition d'une architecture prennant en charge les composants.

Le modèle qui convient le mieux au type des applications que nous visons est un graphe où les sommets sont des composants et les arcs les interactions. Chaque sommet est annoté pour traduire des propriétés particulières du composant. De même, les arcs sont orientés et

valués en fonction du débit associé à l'invocation. Ce graphe ne présente pas, à priori, de propriété particulière. Nous présentons en **??** une proposition pour trouver dynamiquement des propriétés dans ce graphe.

D'autre part, le plan matériel peut être modélisé par un graphe où les noeuds représentent les machines et les arcs les liens du réseau. Les noeuds sont également annotés pour traduire les propriétés de chaque site. Les arcs ne sont pas orientés (les réseaux sont généralement bi-directionnels) mais valués par le débit maximum du lien. Notons que cette représentation ne permet pas une mise en valeur d'un type très répandu de réseau : le bus. Une solution serait de représenter chaque brin de réseau comme un sommet et de typer les sommets en composant et réseau. Néanmoins, étant donné la difficulté de prise en compte d'informations dynamiques sur le comportement d'un réseau de type bus, cette modélisation est suffisante.

Avec ce modèle, le placement d'une application consiste à définir le plongement du graphe applicatif A dans le graphe physique P de manière à optimiser trois valeurs : la charge des sites qui doit être équilibrée, l'utilisation du réseau qui doit être minimale et la mise en concordance des propriétés des composants avec celles des sites qui doit être maximisé. Ainsi, la décision de placer une application afin d'en améliorer les performances représente une classe importante de problèmes d'optimisation combinatoire. En général, trouver une solution optimale est un problème NP-complet, ce qui a été prouvé dans les articles [**?**, **?**, **?**]. Les solutions proposées sont donc basées sur des heuristiques qui tentent d'optimiser "au mieux" mais engendrent souvent des coûts de calcul très importants. Parmi celles-ci, nous pouvons citer la théorie des graphes, la théorie des files d'attente, les algorithmes heuristiques, tels que le recuit simulé [**?**], les algorithmes de recherche tabou [**?**] ou encore les algorithmes génétiques [**?**].

Nous nous trouvons ici dans un cadre totalement dynamique où les données ont une durée de validité qui peut varier de la micro-seconde, par exemple pour la charge, au mois, par exemple pour les propriétés des sites. De même, la structuration du graphe est, elle aussi, dynamique ; un composant pouvant à tout moment décider d'invoquer un autre serveur. Il n'est évidemment pas envisageable de réaliser un placement toutes les micro-secondes avec un coût tel que celui d'un algorithme d'optimisation classique. D'autant plus que nous sommes confrontés aux problèmes classiques de l'équilibrage de charge : diffusion restreinte des informations, distribution des procédures de décision, etc. Par conséquent, la décision de mobilité devra être dynamique tout en gardant un temps de décision assez court. Pour cette raison, nous avons cherché des solutions sous-optimales avec une bonne évaluation heuristique.

Notre travail repose prinicipalement sur deux supports d'optimisation que sont la théorie d'aide à la décision multicritère [**?**] qui nous permet d'aggréger des informations de plusieurs natures et la dérive des connaissances [**?**]. Nous présentons ces deux modèles dans les parties suivantes.

3.2.2 La théorie multicritère

La théorie multicritère repose sur l'élaboration de critères à partir de l'information qui est collectée. Un critère traduit une vision (ou une représentation) du système qu´il qualifie. Un ensemble de critères doit passer par une phase d´agrégation afin de générer un indicateur qui servira de base à une décision. Or, prendre une décision à partir de plusieurs critères, implique que ceux-ci aient certaines propriétés. Nous présentons dans cette partie les bases du processus d'agrégration. Nous en déduisons les propriétés requises pour chacun des critères de façon à

pouvoir définir les transformations à mener sur les informations pour en extraire des critères.

Les actions et leurs conséquences

Un processus de décision est un choix d'une, ou d'un sous-ensemble d'actions, parmi un ensemble d'actions réalisables. Le choix de cette ou ces actions repose sur la possibilité d'exprimer une préférence, c'est-à-dire, la possibilité d'établir une relation d'ordre entre les différentes actions. Soit $A = a_1, a_2..., a_n$ l´ensemble des actions à considérer, il doit être muni d´une relation S telle que $a_i S a_j$ signifie que l´action a_i est préférable à l´action a_j.

Cette relation d'ordre doit s'appuyer sur les conséquence des actions. Plus les conséquences d'une action sont bénéfiques, plus il est souhaitable de la choisir. Les conséquences d'une action s'expriment selon les variations qu'elles induisent sur un ensemble d'indicateurs représentant chacun un point de vue de la situation résultante de l'action évaluée. La notion de point de vue est introduite pour traduire une observation objective, mais réduite, de l'espace modifié par les actions. Un point de vue restreint la vision de l'espace à un sous ensemble d'indicateurs homogènes.

La notion de critère

Un critère, ou plus généralement une fonction critère, est l'estimation des conséquences d'une action selon un certain point de vue. Il s´agit d´une fonction c à valeurs réelles, définie sur l´ensemble A des actions. La fonction c est telle que quelques soient deux actions a_i et a_j il est possible d´écrire

$$c(a_i) \geq c(a_j) \Rightarrow a_i \; S \; a_j$$

où S est une modélisation de la préférence restreinte au point de vue que qualifie c. Autrement dit, si a_i est préférable à a_j alors $c(a_i)$ est supérieur ou égal à $c(a_j)$.

La confiance dans une fonction critère n'est pas forcément absolue. Cela peut être dû à des incertitudes dans la modélisation, des problèmes de précision ou être inhérent au point de vue choisi. Il est possible d'introduire un seuil d'indifférence traduisant le fait qu'une action ne sera préférée à une autre, que si leur évaluation respectives diffèrent d'une valeur supérieure à ce seuil. Soit I_c le seuil d'indifférence attaché à la fonction critère c. Il n'est possible de choisir entre les actions a_i et a_j que si :

$$|c(a_i) - c(a_j)| > I_c$$

Lorsque toutes les conséquences d'une action peuvent être exprimées suivant un seul point de vue, il est possible d'utiliser un critère unique. Dans de nombreux cas, cette simplification n'est pas possible. Dans le cas du déplacement d'un objet A, l'optimisation de ses temps de communication avec un objet B et l'optimisation de son temps d'exécution sont deux critères traduisant des conséquences du déplacement de l'objet A. Ces deux critères sont attachés à deux points de vue différents, et sont difficilement résumables sous la forme d'un critère unique comme nous l'avons montré dans [**?**]. Dans ce genre de cas, il est nécessaire de recourir à un mécanisme d'agrégation complexe pour aboutir à une recommandation.

L´agrégation

L'agrégation multicritère est une méthode permettant, à partir d'un ensemble de critères qualifiant différents points de vue d'une situation, d'obtenir une recommandation permettant de choisir une action parmi un ensemble d'actions réalisables. L'action recommandée est celle qui maximise globalement la valeur de l'ensemble des critères. Dans la plupart des cas, un problème multicritère n´a pas de solution optimale. Il n´existe pas de solution meilleure que toutes les autres pour l´ensemble des critères. Résoudre un problème de cette nature consiste à trouver un compromis.

La recherche de la ou des actions à mettre en oeuvre parmi un ensemble $A = a_1, a_2..., a_n$ est faite en accord avec une fonction d´utilité U, également appelée critère de synthèse. U est une fonction de l´ensemble des critères c_i définis pour un problème.

$$U(a) = F(c_1(a), c_2(a)..., c_m(a))$$

L´action choisie est :
- soit l´action maximisant la valeur de U. Si cette action existe, elle est la meilleure solution au problème posé en fonction du point de vue choisi.
- soit une action a dont l´évaluation via U est supérieure à l´évaluation d´une action de référence r.

$$U(a) > U(r)$$

Toutes les actions répondant à cette condition sont meilleures que l'action de référence r au sens du point de vue choisi. L'objet ici n'est pas de choisir une action optimale, mais plutôt d'en choisir une suffisamment bonne.

La formule de U n'est pas définie à priori. Elle se définit en fonction du type d'agrégation désirée. Il existe plusieurs types d'agrégations simples que nous définissons ci-dessous :

la procédure lexicographique repose sur la définition d'un ordre d'importance au sein des critères, permettant de les ordonner du plus important au moins important. Si le critère le plus important permet de choisir une action, cette action est choisie. Dans le cas contraire, il faut évaluer les actions en fonction du critère d'importance juste inférieur. Ce processus est itéré jusqu'à l'élection d'une action ou épuisement des critères. Il s'agit d'un mode de choix hiérarchique. Cette solution est utilisée dans [**?**]. La simplicité du processus est attrayante. Pourtant, il nécessite le classement des critères et ne tient pas compte des compensations éventuelles d'un critère par d'autres.

la procédure sommative avec pondération consiste à définir pour chaque critère un coefficient d'importance et à réaliser une somme pondérée des critères par les coefficients. Si k_i est le coefficient attaché au critère c_i, et n le nombre de critères, la fonction d'utilité U s'écrit :

$$U(a) = \sum_{i=1}^{n} k_i c_i(a)$$

en posant $\sum_{i=1}^{n} k_i = 1$, U devient la moyenne pondérée des valeurs des critères. Ce processus permet de faire jouer le principe de compensation des critères. Par contre, il

implique de pouvoir exprimer l'ensemble des critères dans une gamme de valeurs et de grandeurs homogènes. La détermination des coefficients d'importance est également un problème complexe, qui influence grandement le résultat de l'agrégation.

Il est nécessaire que la contribution de chacun des critères à la fonction d'utilité U soit indépendante des valeurs des autres critères pour pouvoir utiliser une procédure sommative [?].

la procédure multiplicative similaire dans sa forme à la procédure sommative, consiste en un produit pondéré des critères. Si k_i est le coefficient attaché au critère c_i, et n le nombre de critères, la fonction d'utilité U s'écrit :

$$U(a) = \prod_{i=1}^{n} k_i c_i(a)$$

Cette méthode est adaptée aux critères suivant une distribution exponentielle, c'est-à-dire aux critères à valeurs comprises entre 0 et $+\infty$, et telles qu'une valeur supérieure à 1 décrive une évaluation positive et une valeur inférieure à 1 décrive une évaluation négative.

D'autres formes de processus d'agrégation existent. Ils permettent de traiter des problèmes complexes, mais leur mise en oeuvre implique des coûts en temps particulièrement importants à chaque prise de décision ce qui n'est pas souhaitable dans le cadre de la gestion dynamique du placement.

Les veto

Les veto constituent un complément aux processus d'agrégation fortement compensatoire. Ils permettent d'introduire des limitations. Cette méthode permet d'utiliser les processus d'agrégation compensatoires sans risquer de prendre des décisions inconsidérées grâce à des veto sur les critères critiques. Par exemple, il est possible de fixer un seuil de veto sur le volume minimum échangé entre deux objets pour favoriser un déplacement. Dans le processus d'agrégation sommative, qui est fortement compensatoire, l'importance d'un critère peut être traduite par un coefficient. Les seuils de veto sont une autre méthode pour traduire l'importance. Leur manipulation doit être particulièrement réfléchie car ils peuvent bloquer complètement le processus de décision.

3.2.3 La dérive des connaissances

Dans les parties précédentes, nous avons présenté deux types de procédures de décision qui permettent de gérer l'allocation des ressources dans un système. Ces procédures s'intéressent principalement à une vision extérieure du système, en se reposant sur des indicateurs d'état du système, pour élaborer leurs décisions. Dans le contexte des applications, telles que nous les avons définies au chapitre 1, il est cependant nécessaire d'observer le comportement et les interactions des composants pour optimiser leur accès aux ressources. La théorie que nous présentons ici, repose sur les échanges entre composants et propose un ensemble de règles pour la structuration des applications. Dans cette partie, nous donnons un aperçu de la dérive des connaissances.

FIGURE 3.1 – L´espace relationnel

Présentation de la dérive des connaissances

La dérive des connaissances [**?**, **?**] est une théorie issue de réflexions sur l'évolution des systèmes informatiques susceptibles de supporter la gestion et l´utilisation de systèmes d´information d´entreprises. La multiplicité des ordinateurs et des réseaux les reliant augmente dans des proportions vertigineuses les échanges de données électroniques. Cela pose le problème du rapport entre la notion de connaissance et celle d´information. Dans ce contexte, l´étude part de l´hypothèse selon laquelle, pour construire de la connaissance, il faut savoir oublier de l´information. Autrement dit, la connaissance est obtenue en restructurant les informations échangées. Se pose alors le problème de la définition de critères pertinents pour oublier ou restructurer l´information afin de bâtir des connaissances.

Le modèle relationnel, l´espace d´observation des connaissances, considère une information comme une entité évolutive qui, au-delà du sens véhiculé, possède certaines caractéristiques qui rendent compte de l´évolution de ses interactions avec son milieu. Cela part de l´hypothèse selon laquelle toute entité individuelle existe surtout à travers le rôle qu´elle joue au sein de la collectivité. Ce rôle se définit suivant plusieurs facettes dont celle des interactions qu´entretient cette entité avec les autres. Ces interactions et une représentation des différents contextes qui les ont produites, deviennent alors une partie de ce qui définit ces entités. C´est à partir de cette facette relationnelle représentant les conditions d´évolution des interactions des entités entre elles, que le modèle relationnel propose des critères de pertinence pour transformer de l´information en connaissance.

Le modèle relationnel est applicable aux systèmes qui fondent leur auto-adaptation sur les phénomènes stables (reproductibles dans le temps et dans l´espace) qu´ils produisent ; cela exclut donc la prise en compte systématique d´éventuels épiphénomènes. Ces contraintes permettent au modèle relationnel d´être un modèle descriptif, car basé sur une auto-observation continue du fonctionnement réel du système, et capable de prédire l´évolution probable à court terme des interactions entre les entités du système. Ces prédictions s´adaptent dynamiquement aux évolutions du système.

Le modèle relationnel est constitué d´un espace (voir figure : **??**) dont la description se fait selon trois points de vue (local, collectif et global). Dans cet espace est plongé un graphe orienté

et non nécessairement connexe, où les noeuds représentant les informations en interaction, sont reliés entre eux par des liens valués. L´évolution des relations est assurée par deux mécanismes antagonistes : le flux excitatoire et le flux entropique.

Le point de vue local consiste en la vision de l´espace depuis un noeud. Chaque noeud possède une vision de l´espace au travers des liens qu´il entretient avec d´autres noeuds. Ces liens sont de deux types : ceux modélisant des sollicitations vers ce noeud et ceux modélisant des sollicitations depuis ce noeud. Plusieurs paramètres décrivent chaque noeud de l´espace. La masse d´un noeud est une mesure instantanée de son activité, elle est une combinaison linéaire des interactions de ce noeud. L´altitude d´un noeud est la distance entre le noeud et une origine arbitraire de l´espace, elle est représentative de l´histoire du noeud. Les variations de l´altitude sont proportionnelles à l´activité de l´objet et inversement proportionnelles à sa masse. L´apparition d´un lien entre deux noeuds s´interprète comme la découverte d´une entité par une autre, la disparition de ce lien est une forme d´oubli.

Le point de vue collectif décrit l´émergence de groupes de noeuds en relation de façon stable. Cette structuration est une conséquence des modes d´interaction des entités observées. L´apparition d´un groupe stable appelée «forme relationnelle collective» [**?**] traduit la réalisation d´une fonction complexe par coopération d´entités élémentaires. Une forme collective est un sous-graphe connexe dont l´activation d´un des noeuds implique l´activation de tous les autres dans des proportions stables. La détection de la création d´une forme collective est basée sur l´observation des variations des relations liant ses membres. Si ces variations sont homogènes (proportionnelles) dans le temps, la forme perdure. La perte d´homogénéité d´une forme relationnelle collective, entraînée par une variation non proportionnelle d´un de ses liens, se traduit par sa désagrégation en deux voir N autres formes. La création d´un lien stable entre deux formes collectives entraîne leur fusion en une seule.

Les formes collectives sont particulièrement intéressantes. Elles sont à rattacher au courant de recherche sur l´émergence de comportements collectifs chers aux entomologistes. Par contre leur détection et leur observation hors d´un contexte global est particulièrement complexe.

Le point de vue global est une vision macroscopique de l´espace, via une observation statistique de l´activité de ses éléments. C´est à ce niveau que sont déterminés les paramètres de l´évolution du modèle. En fonction du nombre d´éléments, de la masse globale et des tendances du modèle, le paramétrage pourra, soit augmenter les découvertes, soit augmenter l´oubli. L´oubli, d´un point de vue global se traduit par la disparition d´un noeud de l´espace. Cette disparition est déclenchée lorsque l´altitude d´un objet dépasse un seuil. Les règles globales de gestion doivent traduire, dans le modèle d´observation, la sémantique du système observé.

L´intensité des liens, qui relient les noeuds du modèle, dépend principalement des activations réellement faites entre ces noeuds dans l´espace observé. C´est le domaine d´application qui induit la notion d´activation. Le flux excitatoire définit la façon dont naît et croît la valuation d´un lien du graphe ainsi que les variations d´altitude des noeuds. Le flux entropique est un opérateur périodique diminuant les valeurs des liens. L´entropie modélise la désagrégation due au temps.

Globalement, le modèle relationnel est un espace dans lequel est plongé un graphe. Les noeuds du graphe ayant des interactions sont liés par des arcs. Les noeuds interagissant de façon intense, se rapprochent et progressent vers l´origine de l´espace. Si les interactions ne sont pas

maintenues, sous l´action du temps les liens entre deux noeuds se distendent, voir disparaissent, et les noeuds s´éloignent de l´origine du repère.

Ce modèle est une approche originale de la modélisation des interactions entre entités. Il permet une prédiction à court terme des tendances du système observé. Il est applicable à l´observation des systèmes répartis à objets et permet de déduire des regroupements d´objets (forme relationnelle). Il est nécessaire d´étudier dans quelle mesure la distribution des traitements sur l´espace modifie le comportement global du modèle, et en particulier, vérifier si ses propriétés sont conservées. Nous abordons ceci dans le chapitre **??**.

3.3 Synthèse

La présentation, dans un même chapitre, de la répartition de charge et de l'optimisation n'est pas dénuée d'intentions. En effet, elle nous permet de mettre en regard les besoins de la première par rapport à la seconde. Elle sert également à rapprocher deux domaines qui ne le sont pas habituellement : celui du système et celui de la décision voir de la cognition. Nous avons vu que la part de décision dans la répartition de charge est cruciale. Elle repose sur l'antagonisme suivant : plus la décision est rapide, meilleure elle est puisqu'elle suppose une meilleure réactivité, mais, pour être correcte, la décision a besoin d'un nombre important de données qu'il est coûteux de collecter. Ceci met évidemment en lumière le besoin important de ce type de service quant à la mise à disposition d'informations, ce qui devrait être possible avec les systèmes et le support de langages reflexifs.

Des résultats relativements anciens [**?**] montraient que la mise en oeuvre d'une procédure de décision complexe pouvait nuire à la qualité de l'équilibrage de charge du fait de sa lourdeur. Aucune mesure récente n'a été fournie à ce sujet mais il est probable que la rapidité des ordinateurs actuels feraient mentir ces résultats, à condition que le coût de collecte des informations reste faible. Cela nous conduit à penser que des procédures de décision de type "intelligence artificielles" pourraient un jour avoir leur place dans un cadre très dynamique tel que le nôtre.

Ce problème de décision dynamique n'est pas seulement lié à la réparition de charge et il s'applique à tous les cas où une décision doit être prise dans un temps limité et à partir de données multiples : c'est le cas des applications dynamiques ayant un comportement non décidable à priori. Or ce type d'applications et donc de situations de décision va se généraliser avec l'introduction de composants informatiques embarqués et mobiles.

3.4 Conclusion

Un travail important a été déjà réalisé autour de l'équilibrage de charge ou le placement dynamique. Les principaux résultats concernent aussi bien les performances et les gains possibles que l'architecture du système décomposé en modules fonctionnels. Il en résulte que la mise en oeuvre d'un tel service est un travail complexe et que, si les modèles ou les études théoriques sont nombreux, les réalisations abouties sont rares. Dans le cadre du module de décision, nous nous sommes intéressés aux modèles permettant de l'optimisation de la procédure de décision et, en particulier, à la théorie multicritère qui permet d'agréger différents indicateurs et à la dérive des connaissances qui permet de modéliser les interactions entre les composants.

Chapitre 4

La modélisation et l'administration d'applications

Notre dernière partie d'état de l'art est consacrée à la modélisation des applications réparties et aux travaux existants dans le domaine de l'administration. L'étude de ces travaux nous permet de mieux comprendre comment sont gérés à l'heure actuelle les systèmes d'information. La première partie de ce chapitre donne un aperçu de la norme RM-ODP, introduite pour donner un cadre formel à la conception d'applications réparties. Ce cadre nous concerne puisque, d'une part, les composants que nous gérons en sont issus et, d'autre part, les réalisations que nous présentons au chapitre **??** sont elles-mêmes des applications réparties. La seconde partie présente différentes normes et réalisations autour de l'administration des systèmes et des applications. Parmi les travaux les plus récents, certains sont en relation avec une norme de modélisation qui est utilisée au moment du développement de l'application. Ceci montre l'intérêt grandissant pour l'interaction entre les connaissances du concepteur et le travail de l'administrateur.

4.1 La modélisation

Notre travail s'inscrit dans le contexte des systèmes répartis à objets (SRO), support des applications que nous avons décrites précédemment. De nombreux efforts ont été déployés pour étudier les technologies et les concepts liés à de tels systèmes. La norme RM-ODP (*Reference Model Open Distributed Processing*) [**?**] a donné un cadre pour la spécification de tels systèmes.

Le modèle de référence RM-ODP

Le modèle de référence RM-ODP définit un cadre architectural. Il présente notamment le concept de point de vue. Un point de vue est une perspective que l'on peut avoir d'un système permettant d'en occulter d'autres. RM-ODP définit cinq points de vue :

Le point de vue entreprise définit les rôles des différents intervenants du système, les objectifs, et les politiques globales. Le concept principal de ce modèle est un contrat qui

exprime les obligations entre les différents participants dans une entreprise. Les concepts importants incluent *agents*, *rôles*, *communautés* et *fédérations*. Actuellement, le but est de progresser vers un langage plus spécifique [**?**].

Le point de vue information considère les sources et les destinations de l'information véhiculée. Il fournit une interprétation commune qui assure qu'un concept identifié dans une interface est le même qu'un concept référencé dans une invocation distante. Par exemple, on peut utiliser les spécifications formelles (tel que le langage Z et le langage B) ou UML *Unified Modelling Language* pour décrire ce point de vue.

Le point de vue traitement détermine la décomposition fonctionnelle du système en un ensemble d'objets qui interagissent via des interfaces permettant ainsi leur répartition. Pour permettre les interactions entre objets via ces interfaces, une *liaison* est créée. Nous décrirons cette dernière ci-dessous.

Le point de vue ingénierie qui explicite l'infrastructure nécessaire pour réaliser les interactions entre les objets. Il définit un ensemble de concepts abstraits pour modéliser les communications et les ressources systèmes. Sur le plan des communications, le modèle définit le concept d'un *canal* qui consiste en une configuration avec des *objets talons* (représentants locaux de l'objet distant), des *objets de liaisons* (assurant l'intégrité du canal de communication), des *objets de protocoles* (fournissant une abstraction du protocole de communication) et un *intercepteur*. La figure **??** décrit l'établissement d'un canal par un processus liaison. Si un système est fédéré avec d'autres qui supportent différentes technologies, il est nécessaire d'adapter les communications grâce à une conversion de protocole ou une procédure autorisant le passage entre domaines. Il est souvent réalisé par un intercepteur qui permet la conversion des messages d'un protocole vers un autre.

FIGURE 4.1 – Le processus liaison (ORB)

Sur le plan des ressources systèmes, le modèle définit le concept de *nœuds*, *capsules*, et *grappes*. Un critère important de système distribué ouvert est la transparence pour résoudre un nom avec sa référence.

ODP définit également un ensemble de transparences permettant d'abstraire la complexité des SRO : transparence d'accès, transparence de la persistance, transparence à la localisation, transparence à la migration, transparence à la relocalisation, transparence aux défaillances, transparence des transactions, transparence à la réplication.

Le point de vue technologie qui considère les solutions techniques, les matériels et logiciels pour la réalisation.

Pour les utilisateurs de l'industrie, RM-ODP est trop abstrait. En revanche, pour les éditeurs de logiciels et outils middleware, RM-ODP peut servir de modèle de référence. Un de ses intérêts est de bien mettre en évidence les différents points de vue. Ceci permet de formaliser et structurer le travail d'administration qui cherche des critères explicites pour réaliser une optimisation. Notons que la norme de composants de CORBA est un support évident à la mise en pratique de ce modèle à travers la définition d'interfaces multiples que sont les différentes facettes du composant.

4.2 L'administration

L'administration est un champ d'étude très large. De nombreux outils sont déjà disponibles pour gérer le matériel comme les réseaux, les ordinateurs ou tout autre périphérique ainsi que tout ce qui concerne la gestion d'un parc informatisé. Tout nouvel ajout, modification et suppression d'une composante nécessite une prise en compte du ou des administrateurs. Il existe deux protocoles largement utilisés qui permettent de gérer ces évolutions matérielles : SNMP *Simple Network Management Protocol* et CMIP *Common Management Information Protocol* [**?**].

Dans les systèmes distribués ouverts, les protocoles tels que SNMP et CMIP qui utilisent un paradigme d'agent-gestionnaire (centralisé au niveau du gestionnaire) ne suffisent pas. Tout d'abord, la quantité de données opérationnelles qui doivent être surveillées et traitées en temps réel augmente considérablement. Ceci peut poser un problème de congestion au niveau du gestionnaire. Les agents traitent les informations et envoient seulement les notifications afin de décharger le gestionnaire de la masse d'informations. Cependant, cette technique complexifie et surcharge l'agent et pose le problème de son extensibilité. La complexité croissante des systèmes implique la nécessité d'observer une quantité innombrable de paramètres et fournit des fonctionnalités évolutives. Le traitement centralisé ne convient plus et le gestionnaire doit pouvoir déléguer le travail à des agents et à d'autres gestionnaires.

De plus, l'administration doit également se faire au niveau de la sécurité pour prendre en compte les restrictions d'accès et d'authentification des profils des utilisateurs. De nombreux logiciels commerciaux tentent d'apporter une solution à ces problèmes comme NIS+ [**?**] de SUN Microsystems, ou encore Systems Management Server (SMS) de Microsoft. La tendance actuelle s'oriente vers l'administration au niveau des applications pour une meilleure gestion des installations de logiciels, des licences et des droits d'accès utilisateurs. Naturellement, l'évolution rapide et complexe de l'informatique engendre de nouveaux challenges pour l'administrateur. En effet, avec le poids croissant de l'Internet, gérer des applications devient de plus en plus compliqué. Le modèle Client/Serveur permet de découper les applications en objets plus ou moins indépendants qui peuvent se trouver sur des systèmes différents. Cette avancée technologique implique une totale redéfinition du rôle de l'administrateur ainsi qu'une masse,

trop souvent excessive, de contraintes à prendre en compte. Ces contraintes sont souvent liées au trafic Internet croissant et aux problèmes de sécurité imposés par l'ouverture et la banalisation des autoroutes de l'information. Afin de simplifier et d'obtenir une administration globale et standardisée, des projets sont menés dans des sociétés commerciales (Computer Associates Unicenter TNG, TIVOLI de IBM), des laboratoires de recherche et des groupes de recherche comme le DMTF (Desktop Management Task Force). De plus, les utilisateurs exigent une qualité de service de plus en plus élevée qui nécessite une étude complète du problème en intégrant, par exemple, la gestion des ressources dans ce nouveau type d'environnement.

4.2.1 DME

Distributed Management Environment (DME) de Open Software Foundation (OSF) propose un cadre pour supporter la gestion intégrée de réseaux, de systèmes et d'applications. Il existe trois importants principes pour le développement de DME :

- la consistance : les interfaces homme-machine doivent être consistantes pour gérer les réseaux, les systèmes et les applications.
- l'interopérabilité : il est nécessaire de fournir une interopérabilité entre les systèmes de gestion DME, via un support des différents protocoles industriels standards tels que SNMP et CMIP, un support pour OSI SMI (Structure of Management Information) et la compréhension commune des définitions d'objets.
- la capacité a s'étendre : il est nécessaire de pouvoir s'étendre d'une seule machine à toute une entreprise et de fournir une flexibilité pour la gestion de différents modèles géographiques, topographiques et organisationnels.

FIGURE 4.2 – La structure de DME

La figure **??** montre la structure de DME. DME utilise les technologies objets pour la gestion de réseaux. Il utilise les RPC (Remote Procedure Call) de DCE (Distributed Computing Environment) pour invoquer des autres objets. XMP, protocole de haut niveau pour la gestion des réseaux permet d'accéder aux protocoles de plus bas niveau comme SNMP et CMIP. Le MRB (Management Request Broker) sert d'interface aux opérations sur les périphériques de

réseaux distants et à toutes les applications de gestion. Il y a un adapteur pour traduire les opérations SNMP et les méthodes DME. En dehors des cadres de gestion, quatre applications importantes sont fournies avec DME :

- la gestion d'événements qui permet de répertorier, filtrer, et retransmettre les événements à l'environnement distribué. L'administrateur système peut être informé sur les problèmes ou les changements dans un système. DME fournit un haut niveau de langage ¨template¨ pour la définition des événements.
- le service d'imprimantes qui fournit un service d'imprimantes distribuées via les réseaux.
- le service de distribution de logiciels qui aide à l'installation et au déploiement d'applications distribuées.
- le service de licences de réseaux qui fournit un mécanisme indépendant de l'éditeur pour créer, distribuer et utiliser les licences de logiciels. L'administrateur système peut décider quand, où et à qui les licences doivent être allouées.

4.2.2 GRM+

[?] propose un modèle et un formalisme pour décrire le comportement des relations entre objets gérés dans des buts de supervision et de contrôle. Son travail est basé sur un modèle général de relation de l'approche issue de l'OSI : le GRM (General Relationship Model) [?]. Cependant le GRM manque de directives permettant son exploitation dans un environnement de gestion de réseaux. Pour cette raison, GRM+ propose une architecture logicielle qui a été implantée en Java. Ce travail se situe dans le cadre spécifique de la modélisation des relations entre les objets de gestion de réseaux. Pour les cas plus généraux, l'étude s'appuie sur le service Relations défini par l'OMG. Ce service gère des associations dynamiques reliant des objets sur le bus. Ces relations sont encapsulées dans des objets ce qui permet de parcourir les graphes d'objets et de valider les contraintes d'intégrité référentielle entre ces objets. Ce service Relations fournit une palette très variée de types de relations : l'appartenance, l'inclusion, la référence, l'auteur et l'emploi.

4.2.3 TINA

Dans le monde des télécommunications, il existe beaucoup de standards et de consortiums. Ceci complique les intégrations entre les différents produits et les interopérations entre les différents protocoles. Le consortium TINA-C a été créé pour définir une architecture d'information en télécommucation (*Telecommunication Information Networking Architecture*). Le modèle propose des spécifications et des interfaces interopérables afin de traiter des informations multimédia et de fournir des services de communication [?]. TINA-C définit un modèle d'entreprise avec trois rôles (figure **??**) :

Domaine Utilisateur fournit les informations utilisateur via un terminal réseau tel qu'un ordinateur personnel ou un téléphone mobile.

Domaine Fournisseur de Services représente les services de courtiers d'informations et les fournisseurs de contenu d'informations.

Domaine Fournisseur de Réseaux inclut les organisations qui opèrent sur les réseaux et fournissent les interconnexions entre les domaines.

Les fondations technologiques qui supportent TINA sont :

FIGURE 4.3 – Modèle entreprise et ressource TINA

Orientation Objet Architecture logicielle basée sur des standards préexistants et qui suit le modèle de référence RM-ODP ainsi que celui de CORBA de l'OMG. Elle définit l'ODL (*Object Definition Language*) qui est un super ensemble d'IDL de l'OMG.

DPE (Distributed Processing Environment) Ce sont des systèmes utilisés pour construire des composants logiciels distribués qui peuvent s'exécuter sur différentes machines d'un réseau. Ils cachent la complexité de la distribution en fournissant un bus logiciel. Ces systèmes permettent la construction de systèmes logiciels gérables et extensibles. La figure **??** explicite les composantes du système DPE.

FIGURE 4.4 – Les composantes de l'architecture DPE

Agents Ce sont des composants logiciels qui représentent leurs clients. La caractéristique des agents est leur faculté d'isoler leurs clients des services et des ressources réseaux par leur

haut degré d'autonomie et de mobilité.

Réseaux multiservices TINA supporte plusieurs types de réseaux comme ATM ou RNIS grâce à son indépendance vis à vis de la technologie.

TINA utilise les technologies existantes pour établir une structure permettant aux entreprises d'interopérer entre elles. Les standards usuels des télécommunications sont, par exemple : TMN(*Telecommunicaiton Management Networks*), IN (*Intelligent Networks*), ODP, OMG, etc.

4.2.4 Le modèle d'applications hautement administrables

Pour bien administrer les applications réparties, [**?**] propose une architecture, des mécanismes et des outils pour la conception d'applications réparties. Cinq propriétés sont associées à la gestion d'une application répartie ¨hautement administrable¨ pour faciliter à la fois le travail du concepteur et de l'administrateur. Ces propriétés sont la fiabilité, la transparence à la localisation, l'homogénéité aux activités d'administration, l'indépendance de l'environnement sous-jacent et la traçabilité. Chaque application contient des composants d'administration communs à toutes les applications ainsi que des composants d'administration spécifiques à cette application. Ces composants disposent de services offerts pour mettre en œuvre les actions d'administration, ainsi que des services requis de la part d'autres composants d'administration de l'application ou d'autres applications. Pour permettre l'observation des actions d'administration, certains composants prennent automatiquement des actions de gestion de leurs ressources. La coopération entre ces composants ainsi que les appels de services se font à l'aide d'un système de communication basé sur la diffusion. Un moteur générique d'administration permet de gérer les services offerts et les résultats de ces services à l'aide d'un langage de script DCML (Dynamic Control Management Language). La définition des composants d'administration est pris en compte dès la phase de conception de l'application en intégrant les différents points de vue de la norme ODP. Cette architecture définit un modèle pour gérer les applications distribuées. Une application de cache WEB réparti illustre la conception d'administration. Cependant la définition du modèle semble abstraite et les applications peu aisées à implanter. De plus, il semble que le modèle ne soit pas extensible car il nécessite une adaptation pour chaque application. Ceci nous montre que le modèle manque de transparence dans l'administration des ressources telles que la charge du processeur.

4.3 Conclusion

Le domaine de l'administration en informatique est en plein essort. De nombreuses normes et réalisations existent dans le domaine des réseaux et du déploiement d'applications. Dans ce cadre, la prise en compte des informations est principalement basée sur des données statiques ou évoluant peu et l'utilisation d'informations dynamiques semble difficilement réalisable car trop coûteuse. Leur utilisation n'est donc pas complètement adaptée à l'administration de l'exécution de composants. Cependant, la convergence entre les deux champs d'action semble inéluctable. En effet, la mise en services de composants ayant un comportement dynamique ne permettra plus une gestion simple et manuelle des applications. Ceci suppose un effort pour la collecte d'informations dynamiques et la mise en place de procédures de décision.

Deuxième partie

Contribution

Le but de la première partie a été de mettre en évidence le besoin d'un service d'administration d'exécution pour les environnements de composants. En effet, ces systèmes vont vers plus d'intégration entre le développement et l'exploitation avec des évolutions qui se généralisent au cours du cycle de vie d'une application. Ainsi le schéma traditionnel : analyse, conception développement, déploiement puis exploitation se voit bouleversé par ce besoin d'évolution et les possibilités de dynamisme apportées par les objets. L'utilisation de composants "sur étagère" permet d'envisager des phases de conception et de déploiement qui se passeraient en même temps que l'exploitation de l'application. C'est la granularité plus fine des composants et la modularité du paradigme qui permet d'envisager ces évolutions dynamiques des applications.

Comme toute évolution, celle qui est liée aux composants entraîne de nouveaux besoins. Ces besoins se situent au niveau de la spécification des composants, au niveau de leur archivage ou au niveau de leur exécution. Parmi les problèmes posés, nous avons mis en évidence le besoin d'administrer automatiquement ces composants, leur granularité étant trop fine et leur dynamisme trop grand pour envisager une prise en charge manuelle. Nous proposons dans cette partie un modèle d'administration des composants et sa mise en oeuvre. Ceci nous permet, par la suite, de le valider et de démontrer son intérêt.

Dans le chapitre **??**, nous formalisons un modèle d'administration de composants et nous décrivons l'application de la théorie d'aide à la décision multicritère aux procédures de décision de la gestion d'objet. La spécification et l'implantation du service sont données au chapitre **??**, d'abord pour les environnements intradomaine - c'est à dire limités aux réseaux locaux - puis nous proposons une extension aux réseaux interdomaines ou à longue distance. Les maquettes réalisées ont été évaluées et nous présentons les procédures de test et les résultats obtenus au chapitre **??**.

Chapitre 5

Positionnement

Avant d'aborder le contexte technique de mon travail, je souhaite montrer son positionnement par rapport à d'autres projets. Le but de ce chapitre est donc de constituer une transition entre les aspects d'état de l'art que nous avons développés dans la première partie et les apects de modélisation et de technologie que nous développons dans cette partie.

5.1 L'évolution de l'administration

L'activité traditionnelle de l'administration consiste, à partir d'informations sur l'état du système, à réaliser les évolutions nécessaires à une bonne exécution des applications. Un administrateur est également chargé de réagir en cas de panne en adaptant la configuration des systèmes et des applications de manière à permettre un fonctionnement dégradé pendant le temps d'indisponibilité d'une ressource. Pour finir, une mise en oeuvre correcte de l'administration ne va pas un minimum de prévisions à court et à long terme de l'évolution des systèmes et des applications.

Comme je l'ai déjà évoqué précédemment, le besoin de l'administration de composants est engendré par la dynamicité du modèle. Cette dynamicité ne permet plus de gérer, d'administrer, les composants applicatifs manuellement. En effet, la fréquence des évolutions du logiciel, aussi bien en terme de placement ou de version que de reconfiguration, est telle qu'un opérateur humain suffit plus. Dans ce cadre, quels sont donc les besoins en terme d'administration de l'exécution ? Ils sont constitués, à mon avis, de tous les cas où la structure de l'application ou les interactions entre les composants changent et nécessitent donc une adaptation. Ils recouvrent également la gestion de l'accès et de l'utilisation des ressources et doivent donc prendre en compte leur évolution.

L'évolution des plates-formes matérielles d'exécution est également à la base de nouveaux besoins d'adminitration. L'introduction du réseau n'est pas un fait nouveau au niveau des applications, par contre, l'utilisation généralisée de l'Internet l'est. Or cette utilisation repose sur une ressource difficilement maîtrisable puisque partagée et externe à l'entreprise. Ainsi la mise en place de réseaux virtuels, utilisant le support de l'Internet pour la mise en oeuvre d'applications ne permet pas, par exemple, de se protéger contre les congestions de trafic. L'administrateur peut donc avoir à intervenir pour pallier ces difficultés.

Si, sans chercher à prédire l'avenir, je cherche à projeter ces évolutions technologiques sur le futur, il semble assez naturel de supposer que nous aurons bientôt des provider de services gérant l'exécution d'applications auxquelles les clients payeront pour accéder (le minitel ?). Le service rendu sera alors satisfait par un ensemble de composants qui ne se trouveront pas nécessairement sur le même site ou chez le même provider. Ceci créera des chaînes d'invocations proches de celles qui sont données par les formes relationnelles du modéle défini par F. Bourdon.

La mise en place de ce point de vue me permet, dans la partie suivante, de décrire ma vision des activités liées à l'administration.

5.2 Les activités d'administration

L'évolution du modèle applicatif entraine évidemment une évolution des fonctions liées à l'administration. Dans cette partie, je propose mon point de vue sur les activités d'administration qui servira de support à la définition de notre service au chapitre **??**.

L'administration peut être définie comme la gestion du système d'information. Elle peut être réalisée au niveau applicatif, au niveau système ou au niveau matériel. Elle suppose la gestion des ressources matérielles pour permettre l'exécution "au mieux" des applications. Le critère à optimiser pour permettre l'exécution "au mieux" est la satisfaction des utilisateurs qui passe évidemment par la performance du système. Cette performance est, elle-même, dépendante de la qualité de gestion des ressources matérielles, système et applicatives. Or, pour cela, le service d'administration doit disposer d'informations issues du système d'information.

Les informations considérées peuvent être statiques (structure ou graphe applicatif, topologie du réseau, propriétés matérielles, etc.) ou dynamiques (position des composants, charge et disponnibilité des machines, intégration de matériel mobile, etc.).

Nous l'avons vu au chapitre **??**, les architectures d'administration sont basées sur trois fonctions principales : la collecte des informations, la prise et la mise en oeuvre des décisions. Ces deux fonctions peuvent être décomposées en activités :

1. La collecte d'informations concerne toutes les informations disponibles par observation de l'exécution des applications, telles que le matériel, le système, les applications ou les utilisateurs. Cette activité suppose d'avoir, au plus près de chaque composant, logiciel ou matériel, des fonctions d'observation. Cette fonction est traditionnellement réalisée par des agents d'observations.
2. Une fois ces informations collectées, il est nécessaire de les stocker en vue de leur exploitation. La fonction de stockage est également présente dans les systèmes d'administration classiques et prise en charge par des gestionnaires.
3. L'analyse des informations doit permettre d'élaborer des décisions. Celles-ci peuvent être prises soit manuellement, ce qui est généralement le cas des systèmes classiques, soit avec le support de procédures d'aide à la décision ou encore automatiquement avec des procédures de décisions indépendantes. Le système que nous proposons intègre de tels services qui sont naturellement hérités de ma culture de répartition de la charge. C'est, je le crois, une direction indispensable pour la gestion des environnements à base de composants.

4. La mise en oeuvre des décisions suppose de disposer de moyens d'action sur les entités gérées. Pour ce qui est du matériel, l'intervention humaine est indispensable. Par contre, pour l'aspect logiciel, qu'il soit système ou applicatif, il est possible d'inclure les fonctions nécessaires. Pour la gestion des composants, nous avons parlé au chapitre **??** de services de reconfiguration et de composants adaptables.

La comparaison de cette liste avec celle qui est établie en **??** pour les caractéristiques du placement dynamique met en évidence des concordences fortes tant du point de vue architectural (collecte d'informations, prise de décisions et actions sur les composants) que du point de vue fonctionnel. C'est la raison de notre approche de l'administration depuis la répartition de charge et de notre proposition de regroupement des deux fonctions.

S'il était nécessaire de porter un jugement de valeur sur ce travail, j'accorderais facilement que la part faite à la gestion de charge est sûrement trop importante pour une mise en oeuvre satisfaisant les clients actuels d'un service d'administration. Par contre, je pense que l'utilisation de procédures automatiques pour gérer les ressources, dont la charge du processeur, est une voie d'avenir pour l'administration. Notre modèle constitue, à ce titre, une première pierre à un édifice dont nous donnons l'ébauche par la suite.

5.3 Première ébauche d'un service

La définition intuitive d'un service d'administration se fait sur la base des activités que nous avons identifiés précédemment. Or la modularité du concept de composant doit évidemment être appliquée, en premier, à la conception de l'environnement d'exécution de ceux-ci. Chacune des activités doit donc faire l'objet de composants différents.

La première partie du service d'administration est celle qui est en charge de la collecte des informations. Evidement, cette partie est elle-même divisée en composants. Comme dans le modèle classique, nous y trouvons les fonctions d'observation attachées aux différents éléments du système d'information. Nous y trouvons également les composants en charge de la collecte de ces informations et de leur gestion.

La seconde partie concerne la prise de décision. Puisque différentes politiques d'administration, éventuellement liées à des points de vue d'optimisation différents, peuvent exister à l'intérieur d'un même système, il semble nécessaire de proposer une mise en oeuvre modulaire. Il est possible de tenir compte de différents niveaux de décision. Avec, par exemple, un niveau de décision concernant le placement, la duplication ou la reconfiguration dynamique et un niveau plus général donnant les grands axes de la politique d'administration. L'utilisation des données gérées par la première partie peut être faite par consultation ou sur le mode événementiel. Le positionnement de mon travail se situe clairement dans cette partie en traitant les problèmes liés aux procédures de décisions dynamiques et à l'intégration d'informations pertinentes dans ces procédures.

La troisième partie concerne la mise en oeuvre des décisions. Il s'agit de services qui agissent sur les composants pour améliorer le rendement du système. Nous pouvons citer, à titre d'exemple, les services de migration, de reconfiguration, etc. Ces services nécessitent également la mise en place d'une infrastructure complète qui intervient dès la conception et le déploiement de l'application pour garantir une bonne intégration entre le service et le composant. L'étude de ces infrastructures et de leur mise en oeuvre constitue un sujet en soi.

Il est nécessaire de faire remarquer que nous avons, dans cette partie, complètement passé sous silence la fonction de sécurité. Bien sûr cette fonction est une partie importante et cruciale de l'administration. A tel point qu'il vaut mieux la traiter de manière indépendante comme cela est recommandé dans la norme ODP.

Une fois ces activités spécifiées, il semble évident que l'activité d'administration ne peut être vu seule, indépendemment des autres activités de conception, de déploiement et de support à l'exécution. Ce travail possède donc des liens forts avec d'autres domaines de recherche qui, comme nous l'avons fait avec le placement dynamique, doivent d'être intégrés dans un schéma global d'administration.

Chapitre 6

L'administration de composants

L'administration automatique de composants, telle que nous l'entendons, suppose la prise en charge de ces composants pour leur exécution. Cette prise en charge doit tenir compte des besoins des composants pour positionner chacun d'eux au mieux en fonction des caractéristiques du système et de leur évolution.

Quel est le schéma classique de déploiement d'une application ? Il est généralement basé sur une étude préalable permettant de mettre en évidence les besoins en matériel. La mise en place d'un nouvel applicatif étant généralement l'occasion d'ajouter un nouveau serveur adapté, en terme de performances, à l'application. Le positionnement de l'application ne pose, de ce fait, pas de problème car sa structure monolithique s'adapte à son exécution sur le serveur. Ce travail était jusqu'alors réalisé par les administrateurs du système lors de la mise en place des nouvelles applications. Dans le cas des composants, le support n'est plus le même : on cherche à les faire interagir et à les répartir sur le réseau au plus proche des utilisateurs (ou composants) qui les utilisent. De plus, le contexte de travail qui était jusqu'alors statique (et obligé de l'être) prend, avec les composants, une dimension dynamique : un client n'invoquera pas toujours le même composant car, un jour, il en trouvera un qui lui offira un service plus adapté à ses besoins qui auront évolué. Dans ce cadre en perpétuelle évolution, la gestion manuelle des composants n'est plus envisageable.

La gestion automatique des composants suppose l'utilisation d'informations conjoncturelles sur le système afin d'élaborer une décision adaptée. Ceci suppose la prise en compte de données hétérogènes comme la charge des machines, les interactions entre composants ou les propriétés des sites. Dans ce chapitre, après avoir synthétisé notre contribution et son contexte, nous présentons donc notre modèle d'administration des composants et l'application de la théorie multicritère aux problèmes décisionnels qui en découlent.

6.1 Synthèse de la contribution

Dans cette partie, je présente mon travail d'un point de vue général qui doit permettre de comprendre comment se situe ma contribution et le contexte de mes travaux. Le point de vue technique, qui en explique le contenu, est décrit dans les parties suivantes.

Le point de départ de ce travail est, comme nous l'avons évoqué dans l'introdution, la mise en

oeuvre d'algorithmes de répartition de charge pour les systèmes à objets répartis. L'évolution de ces plates-formes du concept d'objet vers celui de composants, nous a amené à nous poser des questions en terme d'interactivité entre notre plate-forme, celle qui supporte l'exécution des composants et les composants eux-mêmes. Ainsi, la grande facilité d'intervention au niveau des "containers" de composants, support évident et dédié à leur administration, acrédite cette interactivité. Restait donc à définir le cadre de cette intervention à travers un modèle architectural, ce que nous avons fait.

Ce travail a été réalisé dans le cadre de ma participation à l'encadrement de la thèse de Huah Yong Chan, étudiant de l'Université des Sciences de Malaisie, et du DEA de Vincent Portigliatti. Nous le poursuivons actuel en développant la prise en compte d'informations exprimées par l'applications dans des procédures de décisions orientées vers l'analyse et la satisfaction de contraintes.

Un second point important de notre intervention se situe au niveau de l'utilisation d'une procédure de décision élaborée au niveau de la gestion des composants. L'application d'une procédure de décision multicritère a nécessité une étude poussée des possibilités offertes par la théorie multicritère dans des environnements dynamiques.

Ce travail a été réalisé dans le cadre de ma participation à l'encadrement de la thèse de Pascal Chatonnay et en collaboration avec le SEPT à Caen, centre d'étude des postes et télécommunications sur les problème de circulation de documents et de courrier éléctronique.

6.2 Le modèle d'administration automatique

Dans ce modèle, nous avons utilisé une conception modulaire et orientée objet pour qu'il puisse intégrer de nouveaux modules ou permettre le remplacement d'un module existant par un autre. La figure **??** donne un aperçu de l'architecture de notre modèle. Cette partie présente une description détaillée de chacun des composants et de leurs fonctions.

6.2.1 Interagir avec les applications : la réflexion

Pour assurer à la fois l'intégration et la transparence dans notre modèle, nous utilisons le concept de réflexion [**?**] pour interagir avec les applications. La réflexion permet d'unir deux niveaux d'information : un niveau bas, qui est le niveau de l'application et un méta-niveau qui est le niveau de la réflexion. Les informations du méta-niveau proviennent de l'observation du comportement de l'application. Elles sont donc dépendantes de l'application elle-même et de l'environnement d'exécution de l'application. Si une information du méta niveau change, cela entraîne une modification au niveau bas et réciproquement de manière à assurer la cohérence entre les deux niveaux.

L'avantage de la réflexion est que la syntaxe d'un programme peut rester la même quand la sémantique change. Ceci permet donc de changer le comportement d'une application sans en réécrire le code. Cette propriété est intéressante pour implanter un nouveau service système tel qu'un service de sécurité, de tolérance aux pannes ou d'équilibrage de charge sans changer une application existante.

Comment exploite-t-on ce concept dans un environnement de type CORBA ? Nous utilisons l'*Interface Definition Language* (IDL) pour définir les classes ou les services d'une application

FIGURE 6.1 – Architecture d'administration automatique de composants

aussi bien que les services systèmes. Grâce aux fonctionnalités de CORBA, les applications peuvent activer dynamiquement les services dont ils ont besoin. La modification de certains programmes n'entraîne pas la recompilation d'autres programmes qui les invoquent tant que leur interface ne change pas. Il est donc possible de changer le comportement des objets serveurs sans en informer leurs clients. Cette possibilité simplifie l'implantation de services systèmes tels que la migration, la duplication ou l'agrégation.

6.2.2 Positionnement de composants

Pour mettre en œuvre le positionnement automatique des composants, nous proposons de disposer de quatre composants principaux :

- les **proxies** qui sont attachés aux objets et permettent d'intervenir ou d'analyser leur comportement.
- les **moniteurs d'information** pour collecter les informations sur les applications, leur comportement et leur environnement.
- les **modules de décision** qui prennent les décisions de modification du placement des

applications sur la base des événements en cours.
– les **services systèmes** qui mettent en œuvre les décisions prises par les modules de décision.

Les proxies

Pour la mise en œuvre de la réflexion, chaque service système et chaque application publie ses interfaces. Un générateur de proxies prend en entrée ces interfaces et le code des applications pour générer l'interface des proxies et les proxies eux-mêmes. Le proxy se comporte comme le représentant de l'application. Le client invoque donc le proxy (méta objet) plutôt que directement l'objet applicatif lui même. Nous utilisons le concept d'intercepteur [**?**] dans le proxy. Un intercepteur est chargé d'invoquer les services systèmes et les moniteurs d'information avant ou après certains événements tels que l'initialisation, l'invocation et l'arrêt. Les intercepteurs sont des objets qui sont appelés à des points spécifiques des invocations entre objets. En définissant des sous-classes des classes d'intercepteurs et en les enregistrant auprès de l'ORB, il est possible "d'insérer" du code dans le déroulement de l'invocation. Pour pouvoir accéder aux proxies, les clients doivent inclure l'interface des proxies générée par le générateur de proxies.

Le proxy propose la même interface que l'application. Il y ajoute des exceptions systèmes, par exemple l'exception *MOVED* pour signaler que l'objet a été migré, et des interfaces d'administration, par exemple pour activer ou désactiver un service système, changer des paramètres ou des propriétés pour l'application. Dans ce cas, les interfaces servent de deux manières : elles décrivent les fonctionnalités implantées par une application et le comportement de cette application dans son environnement.

Des propriétés décrivent le comportement du proxy, la nature et les besoins particuliers de l'objet. Par exemple, la fonction d'un service système peut être autorisée ou interdite pour un objet particulier ou la propriété de taille d'un objet peut être à grain fin, moyen ou gros.

Les moniteurs d'information

Un moniteur d'information mémorise les informations sur l'état et le comportement du système, telles que le volume de communication entre un client et un serveur, la charge des machines ou la charge des composants. Il y a plusieurs moniteurs d'information qui se partagent la collecte des données. Par exemple, le moniteur de communications et de relations, le moniteur de charge des processeurs ou le moniteur d'utilisation du réseau.

Normalement, un moniteur d'information possède plusieurs interfaces, une pour chacun des composants avec lesquels il coopère : les modules de décision, les proxies, l'administrateur, les services système et d'autres moniteurs d'information comme cela est montré sur la figure **??**.

Les informations servent de paramètres aux modules de décision pour leur prise de décision. Pour obtenir les informations dont ils ont besoin, les modules de décision peuvent consulter les moniteurs d'information périodiquement. Les moniteurs d'information peuvent également envoyer des alarmes aux modules de décision pour leur signaler des événements importants lorsqu'ils ont lieu. Par exemple, lorsque le moniteur d'information qui collecte la charge détecte un déséquilibre important entre deux sites, il peut prévenir le module de décision gérant la charge afin que celui-ci résolve le problème.

Certains de ces moniteurs, tel que le moniteur de communication, interviennent au niveau des

FIGURE 6.2 – Interactions avec les moniteurs d'information

proxies. Dans ce cas, chaque invocation du proxy entraîne une invocation au moniteur.

Un administrateur peut activer, configurer ou désactiver un moniteur d'information s'il estime que son rôle doit être modifié pour permettre au système d'évoluer correctement. Dans ce cas, il utilise l'interface d'administration du moniteur.

Les moniteurs d'information peuvent également échanger de l'information entre eux. Cet échange a généralement lieu entre deux moniteurs du même type. Par exemple, les moniteurs d'information qui gèrent la charge consultent les autres sites pour connaître leur charge et ainsi vérifier qu'elle est bien équilibrée.

Il existe déjà des moniteurs d'informations comme les agents SNMP qui surveillent les réseaux. Des travaux sont réalisés pour effectuer la traduction des protocoles tels que SNMP, CMIP vers CORBA [**?**, **?**].

Les modules de décision

Les modules de décision implantent une politique d'allocation des ressources en collectant les informations auprès des moniteurs d'information, en analysant ces données puis en faisant appel aux services systèmes pour mettre en œuvre leurs décisions. Un module de décision peut donc utiliser plusieurs moniteurs d'information (de même qu'un moniteur d'information peut diffuser ses informations à plusieurs modules de décision) et plusieurs services systèmes. Par exemple, un module de décision de migration statuera sur quand migrer, quel objet migrer, et d'où migrer. Un autre module de décision peut choisir entre une migration et une duplication.

Pour analyser des informations sur le comportement du système et pour prendre des décisions, plusieurs types de procédures peuvent être utilisées. Parmi celles-ci nous pouvons citer des procédures basées sur la théorie des graphes, des files d'attente, de l'ordonnancement stochastique ou l'analyse multicritère. La méthode dépend de la nature et des besoins des applications. Cependant, toutes ces procédures dépendent de la qualité des informations qui sont maintenues par les moniteurs d'information.

Les services systèmes

Les services systèmes sont les actions réalisables sur les objets pour conduire l'optimisation de l'allocation. Les principaux services que nous utilisons sont des services de migration, de

duplication ou d'agrégation. Il est cependant possible d'en ajouter d'autres s'il est nécessaire, à la condition que ces services soient utilisés par un module de décision. Ces services peuvent eux-mêmes reposer sur d'autres services. Par exemple, dans le cas de CORBA, nous utilisons pour la migration les services de *cycle de vie*, de *nommage* et de *fabrique*.

Généralement, chaque service système a trois types d'interfaces : le premier pour les proxies, le second pour permettre aux modules de décision de donner leurs ordres et le dernier pour l'administration du système.

Les services systèmes reposent sur la mise en place de proxies au niveau des objets. Ces proxies agissent comme des souches ou des squelettes dans CORBA, mais leur rôle est de rediriger certains appels à l'objet, ou émanant de l'objet, vers le service système et d'aider à réaliser certaines opérations de contrôle telles que la migration, la duplication, etc. Toutes les applications qui doivent être gérées par les services système doivent utiliser les proxies correspondants. Pour garantir la transparence et l'indépendance des applications aux services systèmes, l'héritage entre les objets proxies et les objets d'application est largement utilisé. De plus, les proxies sont générés automatiquement pour éviter une programmation fastidieuse.

6.2.3 Administration auto-adaptative

Nous avons réalisé une étude qui montre qu'un même jeu de paramètres ne peut convenir à l'ensemble des applications [**?**]. Or, la structure modulaire des applications, le découpage en composants, favorise la dynamicité et donc le changement de comportement de celles-ci. Il est donc nécessaire d'ajouter à notre modèle un mécanisme permettant de prendre en compte ces évolutions et d'adapter les paramètres des différents serveurs. Ce mécanisme est composé d'un module d'administration auto-adaptif et d'un coordinateur.

6.2.4 Le coordinateur

Les modules de décision, lorsqu'ils constatent un déséquilibre dans la projection du graphe applicatif sur le graphe physique, cherchent à effectuer une action, réalisée par les services systèmes, pour rétablir l'équilibre. Nous supposons que chaque action peut être demandée par un module de décision qui détermine quand et comment elle doit être menée à bien. Cependant ce module doit consulter un coordinateur assurant la gestion des conflits entre les actions qui peuvent être réalisées. Si, dans une période de temps, il n'y a qu'une action envisageable pour un objet, le coordinateur autorise cette action. Par contre, s'il y a plusieurs actions qui sont demandées pour un objet, alors le coordinateur a pour fonction de choisir la "meilleure".

Le coordinateur possède deux interfaces : l'une pour les services systèmes, l'autre pour l'administration. Si le coordinateur utilise une procédure de type Electre qui réalise une décision à partir de valeurs étalon, l'interface d'administration peut être utilisée pour modifier dynamiquement sa table de performance, par exemple, pour ajouter une action ou un critère. Chaque service système, avant de prendre une décision pour un objet, doit consulter le coordinateur pour vérifier qu'une action, ordonnée par un autre service système, n'est pas en cours sur cet objet. Si c'est le cas, il doit laisser au coordinateur le choix du service système réalisant l'action. Un coordinateur peut donc être vu comme un module de décision de haut niveau.

Il y a des avantages et des inconvénients à avoir plusieurs services de décision. En effet, cela suppose la mise en place d'un coordinateur, ce qui peut avoir une incidence sur l'efficacité

et la qualité d'une décision. Cependant, il est difficile de prendre en compte tous les critères significatifs pour tous les services systèmes dans un seul module de décision ce qui rend ses choix discutables.

6.2.5 Le module d'administration

Certains paramètres des services systèmes peuvent être modifiés pour tenir compte du comportement du système. Ceci dans le but d'avoir une meilleure flexibilité dans la prise et la mise en œuvre des décisions. Il est donc possible à un administrateur du système d'intervenir pour donner de nouvelles valeurs.

Le module d'administration système peut être pris en charge soit par un intervenant humain soit par un composant qui fait évoluer les paramètres pour mieux adapter les procédures de décision et pour obtenir de meilleures performances. Même dans le cas où cette fonction est tenue par un composant, il est prévu de donner la possibilité à un intervenant humain de pouvoir agir sur le système de décision. Dans ce cas, le composant joue le rôle d'un pilote automatique qui peut seconder ou simplifier les traitements à réaliser par l'intervenant mais qui lui laisse le choix de la politique de base. Ceci suppose aussi que l'intervenant dispose de certains outils tels qu'un visualisateur de performances ou du comportement des différentes applications composant le système.

Les principaux paramètres sur lesquels intervient le module d'administration sont ceux des modules de décision et ceux des proxies. Les paramètres des modules de décision peuvent être ajustés en fonction de l'environnement (nombre d'utilisateurs, domaine d'application, etc). Ces paramètres sont, par exemple, le poids associé à certaines informations plutôt qu'à d'autres, le poids associé au passé, la longueur des files d'attente, la valeur des seuils, les périodes d'observation et de décision, etc. Lorsque ces paramètres sont modifiés, le comportement des modules de décision concernés est affecté et ceci a une influence sur tous les objets gérés par ces modules de décision. Les paramètres des proxies sur lesquels nous intervenons à ce niveau sont les propriétés associées à l'objet. Le module d'administration peut également dynamiquement activer ou désactiver les services systèmes.

6.3 Application de la théorie multicritère à l'administration

Dans l'état actuel de nos connaissances sur les composants, les actions envisageables pour l'administration de ceux-ci sont principalement leur placement sur les sites d'un réseau de manière à optimiser l'allocation des ressources présentes dans ce réseau.

Les actions que nous considérons dans le cadre de la gestion d'objets, sont des déplacements d'un composant d'un site vers un autre. Pour évaluer l'ensemble des actions, nous devons envisager, pour chacun des objets, son déplacement éventuel vers chacun des sites du système, et comparer chacune de ces actions entre elles pour choisir la meilleure. Il semble évident que cette solution, de par son coût, ne peut être envisagée. Il convient donc de limiter l'ensemble des actions potentielles aux actions améliorant les performances.

Pour l'intégration de procédures d'aide à la décision multicritère dans notre gestionnaire de placement, nous avons relevé, dans un premier temps, deux points de vue pour caractériser l'exécution d'une entité. Ce sont la charge des sites et les relations entre objets. Bâtir des

critères, à partir de chacun de ces points de vue, consiste à trouver une combinaison d'informations caractérisant les conséquences (l'optimisation des performances) du déplacement d'une entité. C'est-à-dire, qu'il faut, pour un déplacement d'un site A vers un site B, pouvoir estimer le gain de performance sur le site B par rapport au site A. Ce gain se traduit par la différence d'accessibilité aux ressources depuis le site B par rapport au site A. Chacun des critères s´exprime donc par la différence de disponibilité des ressources depuis le site B et depuis le site A. Dans ce cas, l´action de référence, consistant à ne pas déplacer d´entité, obtient une évaluation de 0 pour chacun des critères. Le coût de l´opération peut être introduit grâce à un seuil d´indifférence entre cette action et l´action de référence. Les actions recommandables sont celles dont l´évaluation est supérieure à ce seuil.

6.3.1 L'indécision due à l'antagonisme

Les deux points de vue, que nous avons mis en évidence, posent le problème de leur antagonisme : l'optimisation de la charge tend à répartir les entités sur tous les sites, alors que l'optimisation des relations tend à concentrer les parties connexes du graphe applicatif sur un même site. Les critères que nous définissons étant antagonistes, il nous faut utiliser une méthode d´agrégation compensatoire, c´est-à-dire une méthode qui permette de compenser les désavantages sur un critère, par les avantages sur l´autre. Ce choix induit l´utilisation de critères prenant leurs valeurs dans une gamme homogène et la détermination de facteurs d´importance attachés à chaque critère. Il est également nécessaire de déterminer des seuils de veto pour les deux critères afin que la compensation ne conduise pas à des situations trop désavantageuses pour l´un d´eux.

Même en prenant ces précautions, le processus de choix peut être inopérant. En effet, nous n´avons mis en évidence que deux points de vue, qui sont de plus antagonistes. La marge de manoeuvre entre l´effet de la compensation et les limites définies par les veto est faible. Deux alternatives permettent d´augmenter la capacité de décision du processus :

1. la prépondérance d´un critère sur un autre, permet de privilégier une direction, en utilisant le deuxième critère comme discriminant entre deux situations égales selon le premier point de vue, ou pour aboutir à une situation légèrement sous-optimale selon le premier critère, mais très intéressante selon le second. Le choix du critère à favoriser doit s´appuyer sur le caractère bénéfique de l´effet attendu. Un critère caractérisant un point de vue dont l´optimisation entraîne d´importants bénéfices doit être privilégié vis à vis des autres critères.

   Marquer une prépondérance pour un critère s´exprime, dans un processus compensatoire, par le choix d´un coefficient fort devant les coefficients des autres critères.

2. l´ajout d´un critère supplémentaire permet d´introduire un discriminant dans les cas d´indécision. Cette solution n´est pas idéale car, si elle permet de discriminer dans certains cas, elle peut aussi conduire à des cas d´indécisions dans des situations préalablement décidables. Cette conséquence indésirable est due à l´effet de coalition.

   L´effet de coalition, au sens intuitif, est la conjonction de deux critères face à un troisième. Soit trois critères, c_1, c_2, c_3, un opérateur d´agrégation $\otimes$ et un seuil d´indifférence S, la coalition de c_2 et c_3 face à c_1 se traduit comme suit :

$$c_1(a) \otimes c_2(a) \geq S,\ c_1(a) \otimes c_3(a) \geq S$$

$$c_1(a) \otimes c_2(a) \otimes c_3(a) < S$$

Dans le cadre de notre étude, les deux solutions sont envisageables. La première est la plus aisée à réaliser : en privilégiant le point de vue de la charge des machines, elle permet de garantir des résultats proches de ceux des travaux faits sur l´optimisation de la charge. La seconde solution, plus complexe, nécessite de définir un nouveau point de vue sur les conséquences d´un déplacement, et de bâtir un critère homogène aux deux précédents basé sur un système de contraintes applicatives. Cette seconde solution a l'avantage de permettre la prise en compte de données supplémentaires et d'établir une interaction avec les différents intervenants dans le cycle de vie d'une application.

6.3.2 Les besoins applicatifs vus comme aide à la décision

L´adjonction d´un troisième critère permet d´ajouter un facteur de discrimination, mais il doit aussi venir compléter intelligemment le processus de décision en traduisant un autre point de vue sur l´optimisation de l´exécution des applications réparties. Pour ce faire, nous proposons d´introduire la notion de besoin applicatif.

Un besoin applicatif est une contrainte posée sur l´environnement de l´application, définissant à un instant donné une propriété nécessaire de cet environnement pour assurer convenablement l´exécution de l´application. Le nombre de contraintes satisfaites par un site constitue un indicateur de l´aptitude de ce site à recevoir l´application ou le composant d´application ayant exprimé les contraintes.

6.3.3 La construction des critères

Les processus de décision présentés au chapitre **??** reposent sur la combinaison de critères décrivant l'état d'un ou de plusieurs sites du système. La volonté de bâtir un processus de décision multicritère induit des contraintes sur la nature des informations prises en compte et le mode de construction des critères agrégeant ces informations. Nous présentons dans cette partie les différents critères considérés, les informations qu'ils agrègent, et la façon dont ils sont calculés.

Les actions, parmi lesquelles nous devons faire un choix, sont l'ensemble des déplacements possibles d'un objet d'un site sur lequel il s'exécute, vers un autre site où il devra poursuivre son exécution. Nous considérons ces actions par rapport à une action de référence consistant à ne pas déplacer l'objet. Une action peut être mise en oeuvre, si son évaluation via le critère de synthèse, est supérieure à l'évaluation de l'action de référence additionnée d'un seuil. Le critère de synthèse traduisant une différence d'accessibilité, l'évaluation de l'action de référence est 0. Le seuil permet de traduire le coût d'un déplacement ainsi que les incertitudes sur la précision de la synthèse.

Nous traitons en premier lieu de la charge des sites, nous abordons ensuite les relations entre objets, pour finir par les contraintes applicatives. Chacune de ces parties explicite la signification précise du critère, le mode de recueil des informations et la formule de calcul du critère. Nous mettons également en évidence les paramètres qui influent sur le calcul de chaque critère. Les valeurs données à ces paramètres déterminent la sensibilité et/ou la réactivité du mécanisme d'allocation.

6.3.4 La charge des sites

Le mécanisme de détermination de la charge fournit à intervalles de temps plus ou moins réguliers, une estimation de la disponibilité de la machine. Cette information, servant à choisir une action à venir, doit avoir un caractère prédictif. La prédiction doit être valide sur l´intervalle de temps séparant deux déterminations de charge.

La charge locale

Ce travail est caractérisé par la liste des files d'exécution (threads) présentes sur la machine. Chacune des threads est dans un état plus ou moins consommateur de la ressource processeur. Nous utilisons les trois états principaux : actif, en attente d'entrées/sorties et endormi. Un indicateur de charge intéressant est constitué d'une combinaison linéaire du nombre de threads actives et du nombre de threads en attente. En effet, en introduisant dans le calcul une part des threads en attente, l'indicateur devient prédictif (il présume du retour, dans la file d´attente du processeur, des threads en attente d'entrées/sorties). L'indicateur de *charge locale* est donc de la forme :

$$chargeLocale = ImpActive \times nbThreadActive + ImpAttente \times nbThreadAttente$$

Les deux paramètres $ImpActive$ et $ImpAttente$ traduisent l´impact de chacun des deux types de thread sur l´estimation de la charge de la machine, c´est-à-dire sur le nombre de threads en concurrence pour obtenir la ressource processeur. A priori, le paramètre $ImpActive$ doit être égal à 1, signifiant que toutes les threads actives sont en concurrence vis à vis de la ressource processeur. Le paramètre $ImpAttente$ traduit le pourcentage de threads actuellement dans la file d´attente des entrées/sorties qui seront à cours terme dans la file d´attente du processeur. L´estimation de $ImpAttente$ est complexe, elle est dépendante de la nature des tâches en cours de traitement et de la base de temps du mécanisme de détermination de la charge.

Notre volonté de comparer deux indicateurs issus de deux sites différents, implique la définition d'une unité dans laquelle ces indicateurs doivent être exprimés. Nous déterminons une quantité que nous nommons *coefficient de puissance relative (CPR)*, en appliquant à l'ensemble des machines participant à l'allocation de ressources, un programme d'évaluation (bench). La quantité rendue par cette évaluation est comparable entre les machines et donc utilisable pour calculer une valeur de charge significative du site, par rapport aux autres sites. Cette quantité sera égale à :

$$chargeAbsolue = \frac{chargeLocale}{CPR}$$

Le critère

Le critère caractérisant la différence de disponibilité de la ressource processeur entre le site local et un site distant est simple dans sa forme. Il consiste en la différence de la charge absolue locale et de la charge absolue distante. Pour un site distant S, il s'écrit :

$$Critere_{charge}(S) = chargeAbsolue_{locale} - chargeAbsolue_{S}$$

Cette quantité est positive si la $chargeAbsolue_{locale}$ est supérieure à la $chargeAbsolue_S$, donc si le processeur du site S est moins chargé. Dans ce cas, le critère favorise le déplacement d'une entité d'exécution depuis le site local, vers le site S. Le critère ainsi défini prend ses valeurs dans l'intervalle :

$$[-chargeAbsolueMaximale_S, +chargeAbsolueMaximale_{locale}]$$

Ceci n'est pas conforme avec l'intégration de ce critère dans une fonction d'agrégation additive. Il est nécessaire de normaliser ce critère pour qu'il prenne ses valeurs dans un intervalle standard, conforme à celui des autres critères.

La normalisation du critère

La normalisation d'une variable sur l'intervalle $[-borne, +borne]$ consiste à ramener cette variable sur l'intervalle $[0, 1]$ (intervalle unitaire), puis à lui appliquer un simple changement d'échelle. Pour ramener une variable sur l'intervalle $[0, 1]$ il est nécessaire de connaître la borne supérieure et la borne inférieure de l'intervalle dans lequel elle prend ses valeurs.

La charge d´une machine est une quantité bornée variant entre 0 et la charge maximale admissible. Cette charge maximale admissible est connue par le système sous la forme d'une constante décrivant le nombre maximal de threads gérables concurremment. C'est cette constante qui se traduit parfois par l'affichage intempestif du message «no more processes» au lieu du résultat tant attendu de la commande que nous venons de lancer (cela se produit lorsque l'on n'utilise pas d'équilibrage de charge !). Cette constante est souvent largement supérieure aux capacités effectives de la machine, elle prend en compte toutes les threads quelque soit leur état. Nous utilisons le nombre de threads maximal de la machine la moins puissante du réseau pour borner la gamme de valeurs du critère de charge.

En effet, nous avons expliqué ci-dessus que la *charge absolue* d'une machine est une représentation de sa charge locale sous la forme du nombre de threads nécessaires pour produire la même charge sur la machine la plus faible du réseau. Aussi, la *charge absolue* de chaque machine est bornée par le nombre maximal de threads de la machine la plus faible (nbThreadMax). Donc le critère prend ses valeurs dans l'intervalle $[-nbThreadMax, +nbThreadMax]$. La transformation suivante ramène le critère sur l'intervalle $[0, 1]$:

$$Critere_{0,1} = \frac{Critere_{charge} + nbThreadMax}{2 \times nbThreadMax}$$

Pour exprimer le critère sur l'intervalle $[-borne, +borne]$, nous utilisons un changement d'échelle et une translation :

$$Critere_{normal} = (Critere_{0,1} \times 2 \times borne) - borne$$

en regroupant ces deux formules et en simplifiant, nous obtenons :

$$Critere_{normal} = \frac{Critere_{charge} \times borne}{nbThreadMax}$$

D'après le mode de calcul du critère de charge, il est très peut probable qu'il atteigne la borne supérieure ($nbThreadMax$). Cela signifierait que l'ensemble des threads d'un site sont actives simultanément. De façon pratique, cette borne supérieure pourra être abaissée pour être plus proche de la réalité du système. L'abaissement de cette valeur permet d'avoir une meilleure répartition des valeurs du critère de charge sur l'intervalle $[-borne, +borne]$.

6.3.5 Les relations entre objets

Comme pour le critère de charge des sites, le critère décrivant les relations entre objets doit caractériser, pour une action donnée, le bénéfice escompté. Dans le cadre des relations entre objets, plus précisément des communications entre objets, un bénéfice est constitué par l´augmentation du débit de communication. Le débit est le nombre d'informations échangées entre les deux extrémités d'une relation durant une unité de temps. L´augmentation du débit de communication entre deux objets peut être obtenue par différents moyens :

- par le choix d´un chemin de communication plus rapide. Dans un réseau homogène, un chemin de communication plus rapide est un chemin plus court. En particulier, nous avons montré qu'une communication locale est plus rapide qu'une communication distante.
- en minimisant, sur chaque chemin, le nombre de communications les traversant. Les relations, perdurant sur ces chemins, disposent de plus de bande passante, plus de débit potentiel.

Les actions de déplacement des objets doivent être évaluées en fonction de la diminution de la longueur des chemins de communications qu'elles engendrent, pondérée par les volumes de communications échangés sur chaque lien. Pour simplifier l'évaluation, nous faisons l'hypothèse que l'optimisation liée à la communication rendue locale par le déplacement est grande devant les diminutions ou les allongements de chemin éventuellement obtenus lors de ce déplacement, hormis pour les relations issues du site d'origine de l'objet. A condition de caractériser concrètement la notion intuitive d'intensité d'une relation, la différence des intensités forme la base d'un critère intéressant pour le choix d'une action de déplacement.

L'intensité des relations

L'intensité d'une relation prend son sens intuitif dans le débit de communication associé à cette relation. Si les deux entités de part et d'autre de la relation communiquent de façon stable et continue, celui-ci est une bonne représentation de l'intensité. Par contre, si la communication n'est pas continue, mais comporte des arrêts ou s'effectue par rafales, le débit, sensible aux épiphénomènes, représente l'intensité instantanée. Pour introduire la notion de durée, il est nécessaire de lisser la valeur du débit en tenant compte de valeurs antérieures. Soit $t-1$ et t deux instants successifs, soit d_t le volume d'informations échangées sur l'intervalle $[t-1, t]$ alors :

$$V_t = \alpha V_{t-1} + \beta d_t \text{ avec } alpha = \frac{a}{a+b} \text{ et } beta = \frac{b}{a+b}$$

Où V_t est une estimation lissée de la relation, V_{t-1} est cette même estimation à l'instant $t-1$. Les paramètres α et β ont donc un sens intuitif fort, respectivement, l'importance du passé et l'importance du présent. Pourtant, nous leurs préférons la notion de *durée d'amortissement* que

nous définissons comme suit : le temps T nécessaire pour que l'intensité (V) passe de la valeur C_0 à la valeur du débit maintenu C à 1% près. La notion de durée d'amortissement permet de mieux appréhender le comportement de l'intensité que ne le permettent les paramètres α et β. En effet, elle introduit explicitement la notion de temps qui est primordiale lorsque l'on traite de communications. Nous donnons ci-dessous la formule liant α et T. Le calcul de la formule est donnée dans [**?**].

$$\alpha = \sqrt[\frac{T}{B}]{0.01\frac{C}{|C - C_0|}}$$

Où B est l´unité de temps de recueil du débit. La durée d'amortissement T est alors le temps nécessaire pour que l'intensité d'une relation, partant de 0, atteigne la valeur maintenue d'un signal observé sur cette relation, à 1% près.

L'unité de localisation des objets est le site. Aussi, l'étude de l'action de déplacer un objet, d'un site vers un autre, doit prendre en compte l'ensemble des relations d'un objet avec les objets du site source et du site cible. Il suffit pour cela de considérer comme débit la somme des volumes d'informations issus d'un objet vers un site. Cette méta-relation (objet-site) à une intensité au même titre qu'une relation standard (objet-objet). Cette nouvelle intensité traduit la dépendance, en terme de communication, d'un objet avec un site. C'est sur cette base que nous bâtissons le critère traitant des relations.

Le critère

Le critère traitant des relations décrit le bénéfice d'une action de déplacement d'un objet d'un site vers un autre. Ce bénéfice peut être estimé par une différence de dépendances. Un objet est plus dépendant d'un site que d'un autre lorsque l'intensité des relations avec ce site est supérieure à l'intensité des relations avec l'autre site. Soit deux sites m_i et m_j et un objet O, notons $I_{O:X}$ l'intensité de la relation liant l'objet O au site X. Alors le critère caractérisant le déplacement de O de m_i vers m_j s'écrit :

$$Critere_{(O,m_i \mapsto m_j)} = I_{O:m_j} - I_{O:m_i}$$

Ce critère, de par sa construction, varie entre la valeur négative de l'intensité maximale entre O et m_i et la valeur positive de l'intensité maximale entre O et m_j. Pour utiliser ce critère dans le processus d'agrégation, il est nécessaire, comme pour le précédent, d'opérer des transformations afin qu'il prenne ses valeurs dans l'intervalle $[-borne, +borne]$. Nous obtenons alors le critère normalisé par :

$$Critere_{nomal(O,m_i \mapsto m_j)} = Critere_{(O,m_i \mapsto m_j)} \frac{borne}{IntensiteMaximale \times BaseDeTemps}$$

La constante $IntensiteMaximale$ introduite dans cette formule doit constituer une majoration du critère non normalisé. Lorsqu'une dépendance existe à travers le réseau de communication (dépendance avec un site distant), il est possible d'en majorer l'intensité par le débit maximal théorique du réseau multiplié par la base de temps de recueil. Cette majoration n'est pas satisfaisante dans le cas d'une dépendance au sein d'un même site. En effet, les communications

dites locales ne dépendant pas du réseau de communication, leur intensité maximale n'est pas majorée par une constante issue de ce dernier. Pourtant, pour simplifier le traitement, nous admettrons que l'intensité maximale d'une dépendance locale peut être majorée par le débit maximal théorique du réseau multiplié par la base de temps. Les cas où cette majoration n'est pas valide se produisent lors de communications particulièrement importantes en volume entre deux objets co-localisés. Dans cette configuration, le critère caractérisant les dépendances peut prendre des valeurs largement inférieures à $-borne$. Cette particularité est admissible dans le cas d'une agrégation sommative, il en résulte une grande résistance au déplacement des entités ayant un fort débit de communication locale.

6.3.6 Les contraintes applicatives

La notion de contrainte applicative fédère le principe de méta-programmation [**?**] et les travaux sur la co-allocation [**?**]. L´idée directrice est que, à chaque «portion» de code d´une application répartie, correspond un environnement matériel et logiciel optimal. De façon plus pragmatique, il est possible de décomposer un programme informatique en plusieurs phases ayant chacune une tâche spécifique. Considérons une application de calcul devant produire un résultat sous forme graphique à partir de données présentes sur disque. Cette application se divise intuitivement en trois phases élémentaires :

1. Une phase d´initialisation chargée de la lecture des informations sur le ou les disques
2. Une phase de calcul portant sur tout ou partie des données préalablement chargées.
3. Une phase de collecte et de restitution des résultats sous forme graphique sur un périphérique spécifique.

Il apparaît clairement que, à chacune de ces phases, correspond un environnement d´exécution spécifique. Cet environnement spécifique est très difficile à détecter en observant le ou les processus lors de leur exécution. Par contre le programmeur d´application dispose d´informations très précises sur le début et la fin de chaque phase, ainsi que sur les ressources qui leur sont nécessaires. Le principe des besoins applicatifs est de fournir au programmeur une interface pour qu´il exprime les contraintes de ses applications. Nous proposons d´étendre la possibilité d´exprimer un besoin à l´utilisateur et à l´administrateur. En effet, le programmeur, l'utilisateur et l'administrateur sont les trois acteurs susceptibles de définir des préférences ou des contraintes sur le mode d´exécution d´une application.

Avant de bâtir explicitement le critère traduisant la notion de besoin, nous présentons de façon plus générale et plus complète ce que sont les besoins applicatifs. Nous traitons de la motivation conduisant à l´utilisation de ces besoins et nous étudions leur nature, ce qu´il est possible et ce qu´il est souhaitable d´exprimer via un système de besoins et de propriétés.

Les motivations

L´objet de ce critère est de profiter de la compétence et des connaissances des différents intervenants autour d´une application pour réaliser le placement le plus adapté possible. Le principe est d´obtenir des annotations sur la sémantique et le comportement de l´application. Ces annotations doivent être, au maximum, indépendantes du système auquel est destiné l´application. En particulier, ces annotations ne doivent pas être dépendantes d´une vision du

système issue du programmeur. Cette vision étant souvent parcellaire, voire fausse, est source de difficultés de gestion pour le système.

Cette approche a aussi pour fonction de réinvestir chacun des intervenants de la fonction d´optimisation de l´exécution. La transparence totale du placement «bute» sur plusieurs problèmes, par contre quelques informations issues du programmeur permettent de faire de bien meilleurs choix de placement [?]. Nous pensons qu´il n´est pas du ressort du programmeur de maîtriser les principes de la gestion de composants. Aussi il n´est pas souhaitable de lui demander des informations sur l'heuristique de décision que le système utilise, ou sur la valeur du seuil de communication pour déclencher une migration. Par contre, le programmeur est capable d´annoter son programme avec des «remarques» sur le rôle de chaque partie. Le programmeur d´applications réparties est en mesure de dire pour telle ou telle partie de code, quel est l´environnement logiciel le mieux adapté, dans le cadre de son application et indépendamment des autres applications pouvant être présentes concurremment.

L´administrateur système doit conserver, à tous les niveaux, son rôle de grand ordonnateur ! Aussi, il est souhaitable qu´il puisse exprimer un certain nombre d´attributs, en particulier pour qualifier les sites. Il doit pouvoir traduire sous forme de propriétés les configurations de chacun des sites, mais aussi des règles de gestion ou de sécurité. La définition de la configuration d'un site permet au programmeur d'exprimer des besoins sous la forme, par exemple, d'une recherche d'un périphérique spécifique.

Les utilisateurs disposent déjà de moyens pour spécifier au système dans quelles conditions ils désirent utiliser une application à partir des fonctions de positionnement ou de synchronisation explicites. En fournissant aux utilisateurs le moyen d´exprimer des besoins sur l´exécution des applications qu´ils lancent, nous élargissons le contrôle de l´exécution. En particulier, nous permettons aux utilisateurs d´intervenir en complément du travail du programmeur.

L´expression d´une propriété

Une propriété qualifie la disponibilité ou la présence, dans le temps ou dans l´espace, d´une ressource ou d´un traitement attaché à un site. L´expression d´une propriété est dynamique et locale à un site, ce qui permet de prendre en compte dynamiquement l'ajout de nouveaux services. Une évaluation peut être associée à une propriété. Cette évaluation permet de quantifier, si besoin est, la présence de la ressource.

Les propriétés s'appliquent aisément à la description de ressources matérielles : elles permettent, par exemple, de recenser les périphériques spécifiques présents sur le site et leur capacité éventuelle. Les propriétés doivent également servir à la description de ressources logicielles. La présence d'un composant sur le site peut être vu comme une propriété du site. Ceci donne aux composants la possibilité d'exprimer leur dépendance à un service logiciel particulier et ainsi d'essayer de s'en trouver proche pour optimiser les échanges avec celui-ci.

L'expression des propriétés peut être réalisée par l'administrateur du site à partir d'un fichier de configuration ou manuellement à partir d'une console d'administration. Elle peut également être gérée automatiquement à partir d'auto-déclaration de modules de traitement logiciels : pilote de périphérique, système ou composant. Pour les propriétés attachées à un composant, le déplacement de ce dernier entraîne le déplacement des propriétés qui y sont attachées.

Les contraintes et les préférences

La recherche ou la fuite d'une propriété ne peuvent être exprimées que par un composant d'application. Ces deux notions ont trait au déplacement, aussi ne peuvent-elles pas être utilisées par l'administrateur dans la définition d'un site. Notons que la recherche et la fuite sont deux facettes d'un même problème. La fuite d'un attribut correspond, exactement, à la recherche d'un site où cet attribut n'est pas exprimé. Les deux approches sont identiques du point de vue de l'allocation et l'utilisation d'opérateurs permet de les grouper en un seul traitement.

La propriété exprime une présence locale. La recherche ou la fuite, par une entité d'exécution, de cet attribut exprime un besoin sur la localisation de cette entité. Peter Dickman, dans ses travaux [**?**], introduit la notion de «coallocation» ou de «contreallocation» de plusieurs entités d´exécution. Dans notre cas, nous pouvons parler de «colocalisation» ou de «contrelocalisation». En effet, ce n'est plus seulement une allocation en fonction des autres entités d´exécution, mais une localisation des composants d´application en fonction des autres composants, des ressources matérielles et des conditions de gestion.

Pour différencier la recherche stricte d'une propriété d'une recommandation, nous introduisons les notions de préférences et de contraintes. Une contrainte, exprimée par un composant, est un besoin incontournable d'un attribut qui limite le nombre des sites sur lequel le composant peut être placé à ceux qui disposent de cette propriété. Elle se traduit naturellement dans la procédure de décision multicritère par un veto. L'expression dynamique de contraintes de la part des composants oblige à mettre en place une procédure d'alarme qui réagisse rapidement à l'affirmation d'une contrainte incompatible avec les propriétés du site courant. Elle est alors un moyen de forcer le déplacement d'un objet de la part d'un des intervenants sur le cycle de vie du composant.

Les préférences exprimées ne sont pas des veto, elles constituent des recommandations. Cela signifie qu'une entité placée sur un site ne disposant pas de toutes les propriétés qu'elle recherche pourra tout de même s'exécuter. Il en est de même pour une entité placée sur un site exprimant un attribut qu'elle fuit. La concordance entre les préférences d'un composant et les propriétés d'un site donne lieu à l'élaboration d'une note qui sera le critère de besoin applicatif. Les contraintes ne participent pas à l'élaboration de cette note.

Les propriétés des sites étant valuées, il est nécessaire d'associer également des niveaux aux besoins exprimés par les préférences. De même, des opérateurs sont associés à ces expressions pour introduire la notion de minimum nécessaire. Ainsi, le formalisme utilisé permet, de la part d'un des intervenants de l'application, de rechercher un site ayant au moins 256 Mo. de mémoire centrale si cette propriété est déclarée par un site. La détermination de la note sur la base d'une expression contenant un opérateur de comparaison est présentée en **??**. Dans la perspective du placement, un site est acceptable pour un objet si le nombre des attributs recherchés par l'objet sur ce site est au moins équivalent à ceux dont il disposait avant de se déplacer.

Le dénombrement des contraintes satisfaites ne peut être réalisé que sur le site exprimant les attributs à étudier. En effet, l'ensemble des propriétés exprimées sur un site peut devenir important, aussi il n'est pas souhaitable que chaque site dispose d'informations de ce genre sur tous les sites du réseau. Ceci implique que lors de l'évaluation du déplacement d'un objet O du site S_1 vers un site S_2, les besoins exprimés par O doivent être étudiées sur S_1 et sur S_2.

S_2 doit fournir un résultat à S_1, pour que ce dernier puisse construire le critère concernant les besoins.

Nous abordons ici la définition des trois niveaux d'intervention. Ces trois niveaux sont définis par les trois intervenants que nous avons identifiés : l'administrateur, l'utilisateur et le programmeur. Chacun d'eux a une tâche spécifique.

Les besoins liés à l'administration

L'utilisation des besoins applicatifs repose pour une grande part sur le travail de l'administrateur. Ce dernier qualifie le système en déclarant les propriétés des sites et des réseaux : il est responsable, sur chacun des sites du système, de fournir une description sous forme de propriétés de la configuration matérielle et logicielle du site. La description de la configuration doit contenir une liste de tous les périphériques connectés au site, le système utilisé, le type de processeur et toute autre information susceptible d'être recherchée par un composant. Cette tâche repose sur l'administrateur mais pourrait être assurée par le système : par exemple dans le répertoire *proc* du système Linux sont stockées un grand nombre d'informations matérielles sous une forme «relativement aisément» exploitable. De façon à être utilisable, les propriétés doivent être connues de tous. Aussi doivent-elles être choisies dans une liste spécifique. Nous appelons cette liste : *liste des propriétés du système.*

L'administrateur peut également exprimer sur chaque site des propriétés propres à l'organisation du système local. Il est responsable de la publication de cette liste auprès des utilisateurs du système. Nous appelons cette liste : *liste des propriétés locales.* Des attributs tels que l'appartenance à un domaine au sein d'une université font partie de cette liste. La liste des propriétés locales peut alors contenir : recherche, enseignement ou équipe1, équipe2, équipe3, enseignement. Les propriétés locales ne sont, à priori, pas connues des programmeurs, elles sont destinées aux utilisateurs.

Les besoins liés à l'utilisation

L'utilisation d'un composant suppose la recherche d'un service donc sous-entend l'expression d'un besoin. Si ce besoin est clairement identifié, il est intéressant d'en faire part au système qui peut aider à le satisfaire au mieux. De même, un utilisateur peut avoir des exigences sur la localisation de l'application qu'il exécute en fonction du contexte. Pour toutes ces raisons et puisque les utilisateurs changent, il doit être possible de définir dynamiquement les besoins d'une application. L'utilisateur n'étant, à priori, pas à même de modifier une application cela suppose que le composant dispose d'une interface dédiée à l'utilisateur pour l'expression de besoins qui deviennent autant de paramètres d'exécution pour le composant. Cette fonctionnalité s'inscrit entièrement dans la logique des composants disposant de plusieurs interfaces, spécialisées en fonction de l'intervenant qui les utilise.

On peut également se poser la question de l'intérêt de laisser l'utilisateur définir ses propres propriétés pour qualifier les sites qu'il utilise, si elles sont compatibles avec les choix de l'administrateur. Dans la mesure où celles-ci correspondent à une connaissance de l'utilisateur sur son environnement applicatif, il semble logique de le laisser faire. Il peut utiliser ses propres propriétés, qui correspondent à un code qu'il définit et qui sont en rapport avec des besoins exprimés dans une application. L'utilisateur ayant besoin de plusieurs machines simultanément

peut, grâce à des propriétés, définir un environnement propice au travail qu'il désire réaliser. Il peut, par exemple, exprimer son nom sur chacun des sites qu'il utilise. C'est une façon de marquer son territoire !

Lors du lancement d'une application, l'utilisateur peut spécifier la recherche ou la fuite de différentes propriétés. Par exemple, en indiquant la recherche de son nom, il privilégiera l'allocation des applications qu'il exécute sur les sites qu'il occupe. Les besoins exprimés par l'utilisateur peuvent faire référence à des attributs appartenant à la liste des attributs locaux. Par ce biais, il tient compte du mode de gestion du système qu'il utilise. L'idée est alors de laisser à l'utilisateur la possiblité de diriger en partie son application dans son exécution.

Les besoins posés par l'utilisateur viennent s'ajouter aux besoins posés par le programmeur.

Les besoins liés à l'implémentation

Lors de l'implantation d'un programme, la tâche du programmeur est double du point de vue des besoins applicatifs. Sa première tâche est de définir pour chaque phase du programme un ou plusieurs besoins qualifiant le travail en court. Sa seconde tâche, de nouveau pour chaque phase du programme, est de définir le contexte matériel et logiciel souhaitable, et d'exprimer ce contexte sous forme de contraintes et de préférences. Le but de ce travail est de donner un maximum d'informations à l'environnement d'exécution pour réaliser le travail de positionnement des composants.

Les besoins exprimés peuvent être de deux natures différentes. Soit ils sont liés à une propriété du système, soit ils expriment une dépendance avec un autre composant :

1. les besoins liés à une propriété du système sont exprimés sous la forme de contraintes ou de préférences. Généralement, les besoins spécifiés par les programmeurs font référence à des propriétés reconnues exprimées dans le système, telles que la taille de la mémoire ou le type de processeur. Ces besoins peuvent donc porter sur la liste des attributs du système. Par contre, ils ne peuvent pas porter sur la liste des attributs locaux ni sur la liste des attributs de l'utilisateur, car le programmeur n'est pas censé avoir connaissance de ces dernières.
2. les besoins faisant référence à un lien avec un autre composant sont dépendants de l'application. Ils sont proches des "coallocateurs" définis dans [**?**]. Comme les autres besoins, il est possible de leur associer une note pour traduire le degré de dépendance. Ceci peut servir de support au programmeur pour définir les positions relatives des différents constituants de son application. Les liens entre composants ne peuvent pas faire partie des contraintes car le risque d'instabilité entraîné par une "grappe" de composants est grand.

L'expression d'un besoin doit correspondre à un souci, pour le programmeur, d'optimisation de l'environnement matériel et logiciel des différents composants de son application. En effet, ces fonctions sont fournies au programmeur pour lui permettre d'aider à la prise de décision du système de gestion de composants. Deux règles minimales posent les bases de l'utilisation des besoins applicatifs :

1. moins il y a de besoins posés, plus ceux qui le sont prennent de l'importance dans le processus de décision. Nous le montrons ultérieurement, dans le paragraphe **??** traitant

de la construction effective du critère.

2. pour un composant d'application, poser le besoin d'une propriété n'est pas une garantie d'être, à terme, colocalisé avec un autre composant ayant exprimé ce besoin. Il s'agit d'une aide au processus de décision, qui peut satisfaire ou ne pas satisfaire le besoin.

L'expression des propriétés, et la pose de contraintes sur ces derniers, peuvent mener à des contradictions. Nous étudions ce problème dans la partie suivante.

Les contradictions

A chacune des propriétés et à chacun des besoins est attaché une sémantique qui, si elle est mal maîtrisée, conduit à l'expression de combinaisons de propriétés et de contraintes contradictoires.

La sémantique d'une contrainte est simple. Elle consiste en la recherche d'un site disposant ou ne disposant pas d'une propriété spécifique. Une contradiction évidente est réalisée lorsque le système de contraintes attaché à une entité recherche un site disposant **et** ne disposant pas d'une propriété. La solution à ce problème est simple. Il faut supprimer les deux contraintes contradictoires du système et continuer l'exécution. Le problème doit être signalé à l'utilisateur (en espérant que ce soit encore le programmeur).

Notons également que nous n'avons pas défini les propriétés ni les besoins qui peuvent être exprimés par les différents intervenants du système. L'idée est simplement d'offrir un mécanisme capable de les manipuler sans connaissance particulière sur leur signification. Dans ce cadre, il est évident que le mode de traitement, si il reste simple, ne permet de recherches élaborées. Des contradictions peuvent ainsi facilement survenir comme nous l'avons vu. L'ajout d'analyseurs lexicaux et syntaxiques permettront d'améliorer le support mais là n'est pas notre but.

Le critère

La notion de besoin applicatif étant défini, nous étudions dans cette partie la constitution du critère traduisant l'intérêt, du point de vue des besoins, du déplacement d'un objet. Nous présentons tout d'abord l'évaluation sur un site des préférences exprimées par un objet. Nous traitons ensuite de la constitution du critère à partir des deux évaluations faites sur le site source et le site cible du déplacement potentiel.

Un objet exprime deux sortes de besoins, les contraintes et les préférences. Nous avons vu que les contraintes ne sont pas prises en compte dans le calcul du critère mais qu'elles sont utilisées comme des veto. Parmis les préférences exprimées par un objet, nous pouvons différencier les préférences en terme de propriété du système et les préférences en terme de coallocation. Pour chacune des préférences de propriété une note est attribuée en regard de la valeur exprimée par le système. Pour chacune des préférences de coallocation, une note est attribuée en fonction de l'importance associée à cette préférence (la valeur associée à la préférence exprime l'importance attachée celle-ci). Ainsi, un site satisfait plus ou moins un système de besoins en fonction du nombre de propriétés satisfaites. Nous proposons un indicateur simple constitué de la moyenne des notes obtenues à chaque préférence exprimée :

$$ind(O,S) = \frac{\sum_{i=0}^{i=n} f_i(Preference_i) + \sum_{j=0}^{j=m} f_i(Coallocation_i)}{n+m}$$

Si NBP est le nombre de préférences exprimées dans le système de contrainte d'un objet. D'après sa construction, l'indicateur que nous définissons prend ses valeurs dans l'intervalle $[-NBP, +NBP]$. Cet indicateur exprime l'intérêt d'un site pour un objet du point de vue des propriétés qu'il a déclaré.

Considérons l'action de déplacement d'un objet O d'un site S_1 vers un site S_2. Le critère jugeant cette action doit exprimer la préférence ou non de cette action vis à vis de l'action consistant à ne pas déplacer l'objet. Nous proposons de bâtir le critère à partir de la différence des indicateurs concernant le site d'origine (S_1) et le site éventuellement destination (S_2) de l'objet.

$$Critere_{besoin}(O; S_1 \mapsto S_2) = ind(O, S_2) - ind(O, S_1)$$

Ce critère donne la valeur 0 pour l'action ne rien faire. Il est positif si le site S_2 est préférable au site S_1. Il est négatif dans le cas contraire.

La normalisation de ce critère est relativement aisée, au sens où elle est fonction du nombre de besoins exprimés par l'objet considéré. En effet l'indicateur $ind(O, X)$ prend ses valeurs dans $[-NBP, +NBP]$ donc le critère prend ses valeurs dans l'intervalle $[-2NBP, +2NBP]$. Nous en déduisons la formule de normalisation permettant d'obtenir un critère variant dans l'intervalle $[-borne, +borne]$:

$$Critere_{besoin,normal}(O; S_1 \mapsto S_2) = Critere_{besoin}(O; S_1 \mapsto S_2)\frac{borne}{2NBC}$$

Il faut noter que le critère sur les besoins ne dépend d'aucun paramètre hormis la borne qui est commune à tous les critères. Nous avons ci-dessus énoncé une règle disant que, moins de besoins sont exprimés, plus ceux qui le sont, ont d'importance dans le processus de décision. Cela tient à la technique de normalisation utilisée. L'utilisation de la valeur NBP, décrivant le nombre de besoins exprimés, à la place d'une borne globale unique comme pour les autres critères, permet de maximiser l'impact de chaque besoin.

6.3.7 Les veto

Le processus d'agrégation menant au critère de synthèse étant compensatoire, du fait de l'antagonisme des points de vue choisis, il peut mener à un choix particulièrement mauvais selon l'un des points de vue. Pour pallier à ce défaut nous avons la possibilité de définir des veto. Les veto sont constitués par des seuils sur la différence d'accessibilité entraînée par une action. Le franchissement d'un des seuils signifie que l'action qui en est responsable ne peut être choisie.

Considérons un critère normalisé C parmi ceux définis ci-dessus et une action consistant à déplacer l'objet O du site S_i vers le site S_j. Assurer que cette action est viable vis à vis d'un veto V posé sur C, est équivalent à vérifier que l'inégalité suivante est vrai :

$$C(O, S_i \mapsto S_j) > V$$

Le choix de veto trop restrictifs peut rendre le système de décision inopérant. C'est-à-dire que quelque soit l'action évaluée, elle ne peut satisfaire tous les veto simultanément. Il est donc souhaitable de s'attacher à choisir les veto de manière à conserver un espace d'évolution au processus de décision.

Les veto utilisés sont posés :

sur la charge : définir un veto pour la charge consiste à définir la différence minimale de charge acceptable entre le site d'origine et le site de destination d'un objet. Dans le cadre d'un choix multicritère, le déplacement d'un objet sur un site plus chargé que son site d'origine, peut être souhaitable, du fait d'une optimisation importante sur l'un des autres critères. Le problème consiste à déterminer la perte maximale à autoriser sur la charge. Nous estimons qu'il est intéressant d'étudier la réponse du système pour des valeurs de V_{charge} comprises entre $-0.1borne$ et $+borne$.

sur les relations : la réflexion, autour d'un veto mis sur les relations, pose le problème des objets communiquant faiblement ou pas du tout. En effet, un objet ne communiquant pas ou peu engendre, pour toute action, une évaluation du critère sur les relations égales ou légèrement supérieure à 0. Or, ces mêmes objets, surtout ceux n'ayant aucune relation, sont intéressants pour équilibrer la charge. Ils peuvent être placés sur n'importe quel site sans induire de pertes en terme de communications. Cette remarque plaide pour un veto sur les communications interdisant les grosses pertes, mais n'obligeant pas un gain pour justifier un déplacement. Cela signifie que le veto ne doit pas être supérieur à 0. Les valeurs comprises entre $-borne$ et 0 équivalent alors à admettre une perte, plus ou moins grande, du point de vue des relations, si elle est compensée par une amélioration sur un autre critère.

sur les contraintes : nous l'avons expliqué dans le paragraphe traitant du critère sur les besoins applicatifs, les besoins sont de deux sortes : les contraintes qui entraînent naturellement des veto et les préférences qui ne sont que des recommandations. Il n'y a donc pas lieu de s'interroger sur des valeurs à associer à un veto sur les besoins. Par contre, il est nécessaire, lors de la déclaration des contraintes d'une application, de ne pas risquer de bloquer celle-ci en définissant un système trop restrictif.

A ce stade, nous disposons de trois critères, chacun exprimant une préférence selon un point de vue et de trois veto attachés chacun à un des points de vue précités.

6.3.8 Choix des facteurs

Considérons une action A, la forme du critère de synthèse U est la suivante :

$$U(A) = k_1 \times Critere_{charge}(A) + k_2 \times Critere_{relation}(A) + k_3 \times Critere_{besoin}(A)$$

Les constantes k_1, k_2 et k_3 sont les facteurs d´importance portant respectivement sur la charge, les relations et les contraintes. Ce sont les valeurs données aux différents k_i qui vont définir le comportement de la stratégie d´allocation. Chaque k_i se définit, non dans l'absolu, mais par

rapport aux autres facteurs. Ils représentent les importances relatives des critères. La valeur associée à chacun de ces facteurs est, comme nous l'avons déjà dit, dépendante des directions d'optimisation fixées par la gestion d'objet. Le choix des facteurs dépend également du type des applications gérées.

Il semble évident que le critère considéré comme prépondérant doit avoir le facteur de plus grande valeur. Notons également que ne pas privilégier un des critères et attribuer des facteurs équivalents à tous peut aboutir à un lissage de la procédure de décision. En effet, l'inertie liée aux critères de même importance entraîne la nécessité d'avoir une valeur très importante sur l'un d'eux pour faire pencher la procédure de décision en sa faveur.

Nous étudions au chapitre **??** l'influence des facteurs sur les performances du service de placement.

6.3.9 La prise de décision

Le critère de synthèse est le point central du processus de décision. Mais il n´en est pas le seul constituant. Il est également nécessaire de définir une stratégie pour fournir au critère de synthèse des actions à évaluer.

Le schéma général d´une prise de décision, à une période donnée, est le suivant :

- construire un sous-ensemble d´actions susceptibles d´être mises en œuvre. La construction de ce sous-ensemble s'appuie sur le calcul de la part relationnelle que nous avons développé dans [**?**]. En effet, le rapport des parts relationnelles permet de mettre en évidence les relations qui prennent de l'importance, celles que nous souhaitons privilégier. En combinant ce rapport avec l'intensité relationnelle d'une relation, nous trions les listes de relations des objets. Nous contruisons alors le sous-ensemble en limitant les relations évaluées aux relations les plus importantes. Les objets sont ensuite eux mêmes triés de manière à les ordonner par importance de relation. Les informations traduites par ce tri ne sont pas certaines, elles constituent uniquement une heuristique permettant de réduire le nombre d'actions à évaluer.
- extraire une à une les actions de ce sous-ensemble et les proposer à l´évaluation. C'est à dire, déterminer, pour l´action proposée, chacun des critères et les examiner au regard des veto. Puis, si l´action satisfait tous les veto, il faut construire le critère de synthèse. La construction du critère de synthèse suppose d'utiliser des informations sur les sites avec lesquels l'objet est en relation. Pour favoriser la répartition de charge au sein du réseau, nous ajoutons à cette liste le site le moins chargé, à la connaissance du service local.
- Si l´action satisfait au critère de synthèse il faut la mettre en oeuvre, sinon il faut considérer l´action suivante dans le sous-ensemble.

L'approche multicritère que nous avons définie suppose plusieurs paramètres tels que les facteurs associés aux différents critères, la base de temps de calcul des débits, etc. Elle est capable de prendre en compte un nombre d'information supérieur à celui qui est généralement utilisé dans l'implémentation du placement. Notons également que rien ne s´oppose à l´ajout de nouveaux points de vue, donc de nouveaux critères, au sein du processus de décision. La méthode de constitution d´un critère que nous mettons en oeuvre peut s´appliquer à de nombreuses autres informations, comme par exemple : les notions d´occupation de la mémoire et de dépendance avec des fichiers.

6.4 Conclusion

Dans ce chapitre, nous avons proposé une architecture pour la mise en place d'un service d'administration automatique de l'exécution de composants et l'intégration d'une procédure de décision multicritère dans les modules de décision. La définition de notre architecture repose sur les principes de base qui régissent les objets : modularité et spécialisation des modules. Le but de cette architecture est de pouvoir permettre à la fois la transparence vis-à-vis des applications et la possibilité pour un composant de fournir lui-même des informations pour faciliter le travail de la procédure de décision. L'adaptation de la théorie multicritère à nos besoin permet une approche orginale du placement dynamique et la prise en compte d'un nombre important d'informations.

Chapitre 7

Le service d'administration

Le nombre de paramètres à prendre en compte pour représenter le comportement de notre système est trop important pour pouvoir envisager une modélisation mathématique ou même une simulation. Pour cette raison, nous réalisons une maquette offrant le service de placement. Cette approche nous permet, d'une part de valider la faisabilité d'un tel service, et d'autre part de pouvoir l'analyser en supportant l'exécution d'applications de tests.

Le travail qui nous a conduit à mettre en place une maquette d'administration de composants a été réalisé en trois étapes. Comme nous l'avons expliqué précédemment, le point de départ de ce travail est la mise en place d'un système d'allocation de ressources pour les systèmes à objets répartis. Ce travail a donné lieu à la réalisation d'un marché pour le SEPT et à la soutenance de la thèse de Pascal Chatonnay.

La volonté d'élargir notre recherche en intégrant un maximum d'informations nous a conduit à définir un cadre plus large de gestion des objets. La seconde étape a donc consisté en l'introduction de la notion de domaine dans le système d'allocation de ressources et de la possibilité de définir des contraintes applicatives. L'ajout de la gestion des contraintes applicatives nous a permis d'offrir un premier support à l'administration des objets entre les domaines. Cette seconde partie de notre travail a donné lieu à la signature d'un marché avec le CNET et à la soutenance de la thèse de Huah Yong Chan.

L'utilisation de ces contraintes à tous les niveaux, intra et interdomaine, nous a permis de définir le cadre général de notre service d'administration automatique de l'éxécution de composants. Le travail est en cours avec la thèse de Vincent Portigliatti ayant pour thème l'étude de l'impact des besoins applicatifs sur l'administration dynamique des composants.

Dans ce chapitre, nous présentons la réalisation d'un service d'allocation de ressources destiné à administrer l'exécution des objets ou des composants. Dans une première partie, nous décrivons le gestionnaire de réseau local qui permet d'obtenir une gestion globale dans un domaine identifié. Dans la seconde partie, après avoir mis en évidence les spécificités de la gestion d'objets entre différents domaines, nous détaillons notre mise en oeuvre.

7.1 La gestion des composants

Nous présentons ci-après, l'architecture du mécanisme d'allocation des ressources aux objets et ses algorithmes principaux. Nous définissons également les protocoles de collaboration permettant aux différents services de coopérer pour obtenir la prise d'une décision.

7.1.1 L´architecture du mécanisme

L'architecture de la gestion d'objets dans un domaine est décomposée en services respectant le modèle défini au chapitre **??**. Le premier développement de ce service n'incluait que deux services d'information : le service de gestion de la charge et le service de gestion des relations. L'adjonction d'un service de gestion des besoins applicatifs nous a permis de valider l'intérêt de la conception modulaire de notre modèle. Nous n'avons pas utilisé plus d'un service de décision dans cette implantation. Le principal, celui qui a fait l'objet des premiers développements était destiné au placement. D'autres expérimentations ont également été conduites sur la duplication [**?**] mais sans faire cohabiter les deux services simultanément. Pour cette raison, nous ne présentons pas ici de module coordinateur.

FIGURE 7.1 – L´architecture du mécanisme d'allocation

La figure **??** propose un schéma de l'architecture du service complet. Aucun des services d'information ne collabore directement avec un service d´information de type différent. De ce fait, chacun peut être utilisé indépendamment des autres, et également en leur absence. Le point central du système d'allocation est le service de décision : c'est lui qui, en connaissance des services d'information disponibles, assure la circulation des informations.

Nous avons mis en oeuvre les quatre modules de notre solution sous la forme d'objets CORBA. Chacun des services dispose de plusieurs interfaces, adaptées chacune à un mode de collaboration. En particulier, tous les services ont une interface d'administration qui permet la modification dynamique de leurs paramètres. De plus, chaque service d'information est bâti autour d'une structure de données spécifique et d'un algorithme de manipulation de cette structure.

L'ensemble est regroupé en un seul processus serveur qui est appelé GO, pour gestionnaire d'objet.

Nous présentons dans la suite, pour chacun des services, les points principaux de son implémentation.

7.1.2 Le service de gestion de la charge

Le service de gestion de la charge (ou LOAD) a de multiples fonctions. Il doit tenir à jour une valeur représentant la charge locale. Il permet de diffuser cette information vers d'autres sites et de maintenir localement une représentation partielle de l'état, en terme de charge, des autres sites du réseau. Il est aussi responsable d'informer le service de décision des déséquilibres importants entre la charge locale et la charge des sites distants.

Le calcul de la charge locale est une tâche simple qui consiste, à intervalles de temps réguliers, à dénombrer les threads s'exécutant localement et à réaliser la somme des threads actives et d'une portion des threads en attente d'entrées/sorties.

FIGURE 7.2 – Structure de données de gestion de la charge

Chaque site dispose d'une structure de données formée d'une liste de taille variable de références contenant des informations sur d'autres sites (figure **??**). A chacune de ces références correspond : une charge, une date d'obtention et le nombre de fois que cette information a été demandée par le service de décision au cours de la dernière période de décision. Cette liste constitue la représentation partielle de la charge des sites du réseau.

Le maintien à jour, dans chacun des services de gestion de la charge, de cette structure repose sur l'utilisation de l'algorithme des enchères réduites [**?**]. Après chaque détermination de la charge locale, chaque gestionnaire émet cette information. Il l'émet vers les n sites les plus demandés par le service de décision lors de la dernière phase de décision, et également vers n autres sites choisis aléatoirement parmi tous les sites du réseau. En retour de chacune de ces émissions, le gestionnaire reçoit la charge locale du site récepteur. Il met à jour sa structure

avec ces informations. Les services de gestion de la charge des différents sites ne sont pas synchronisés. Les émissions ne sont donc pas toutes réalisées en même temps.

Lors de la fin d'une phase de décision, signalée par le service de décision, la liste locale des références de sites est analysée. Les références sont triées en fonction du nombre de fois qu'elles ont été demandées par le service de décision. Une référence de site disparaît de la liste locale si sa date de mise à jour est ancienne et si le gestionnaire de décision n'a pas demandé cette information lors de la dernière période.

Si, lors de la réception d'une information distante, le gestionnaire de charge constate un déséquilibre important, il propose ce site au service de décision pour une migration d'urgence en envoyant une alarme au service de décision.

FIGURE 7.3 – Le service d´information gérant la charge

Le service de gestion de la charge dispose de trois interfaces (figure **??**) :

1. une interface d'administration permettant de modifier dynamiquement ses paramètres.
2. une interface accessible par tous les autres services de gestion de la charge, leur permettant d'échanger des valeurs de charge.
3. une interface permettant au service de décision local de lui demander le critère décrivant l'action de déplacement d'un objet local vers un site distant. Cette interface est également utilisée pour signaler la fin d'une phase de décision.

7.1.3 Le service de gestion des relations

Le service de gestion des relations (ou COMOBS) doit maintenir, pour tous les objets locaux, une liste de leurs relations avec les différents sites du réseau. Il doit également, sur requête du service de décision, fournir une ou plusieurs actions (déplacement d'un objet vers un site) potentiellement bénéfiques.

La structure de données (figure **??**), du service de gestion des relations, est une liste dynamique des objets locaux avec, pour chacun d'eux, une liste des sites avec lesquels ils entretiennent une relation. Pour chacun des sites est stocké le volume échangé sur la période en cours (débit), l'intensité et la part relationnelle au début de la période. Pour chaque objet, les communications échangées avec des objets locaux sont stockées sous la même forme que les

autres, mais ne sont pas intégrées à la liste. Cela est dû au fait qu'une migration vers le site local, d'un objet y résidant déjà, n'est pas une action à proposer au service de décision.

FIGURE 7.4 – Structure de données du service de gestion des relations

Un objet est responsable de la signalisation de son arrivée et de son départ d'un site. Cela permet au gestionnaire d'optimiser la taille de ses listes. Chaque objet est également responsable de maintenir à jour les débits de ses relations. Pour ce faire, il invoque le service de gestion des relations lors de chaque communication, et lui fournit le site cible ou origine de la communication, et le volume échangé. Si un objet tente de mettre à jour un débit alors qu'il n'est pas connu localement, il est enregistré de façon implicite. Si l'objet oublie de signaler son départ d'un site, son entrée dans la liste est maintenue jusqu'à la disparition de toutes ses relations. Une relation disparaît lorsque son volume tombe en-dessous d'un certain seuil et qu'elle n'est pas réactualisée pendant une période.

A intervalles de temps réguliers, le gestionnaire analyse l'ensemble des relations qu'il maintient. Pour chacune d'elles, il calcule la nouvelle intensité et la nouvelle part relationnelle, puis il remet le débit à 0. Pour chaque objet, il trie ses relations en accord avec le produit de l'intensité par le rapport des parts relationnelles. Puis, il trie la liste des objets en fonction du même index, appliqué seulement à la première relation de leur liste.

Lorsque le service de décision en fait la demande, le service de gestion des relations lui fournit des actions à évaluer. Pour chaque action, le système de gestion des relations fournit la référence de l'objet, le critère formé par la différence de l'intensité des relations locales et de l'intensité des relations avec un site distant. Nous avons défini l'ordre suivant : le service

fournit successivement la première relation de chaque objet, jusqu'à épuisement des objets. Il fournit ensuite successivement la deuxième relation de chaque objet et ainsi de suite. Un objet ayant un âge inférieur à l'âge de stabilisation, cette action est ignorée. Lors de la fin de la phase de décision, signalée par le processus de décision, le curseur pointant sur la dernière action transmise est repositionné sur la première action du premier objet de la liste.

Notons que la phase de tri et la phase de décision se font en exclusion mutuelle, c'est-à-dire qu'un tri ne peut pas débuter tant que le service de décision n'a pas signalé la fin de la phase de décision, et que le service de décision ne peut obtenir une action que si la phase de tri est terminée.

FIGURE 7.5 – Le service d´information gérant les communications

Le service de gestion des relations dispose donc de trois interfaces (figure **??**) :

1. une interface permettant à chaque objet local de mettre à jour le débit sur une de ses relations. Cette interface sert également aux objets pour s'enregistrer ou se désenregistrer du gestionnaire local.
2. une interface d'administration permettant de modifier la valeur de ses paramètres.
3. une interface permettant au service de décision d'obtenir une action à évaluer et le critère afférent. Cette même interface sert à signaler la fin d'une phase de décision.

7.1.4 Le service de gestion des besoins applicatifs

Le service de gestion des besoins applicatif (ou ORM) [**?**] doit maintenir, pour tous les objets ou composants locaux, une liste de leurs contraintes, préférences et colocations. Il gère également la liste des propriétés du site local. Il doit, sur requête du service de décision, fournir une ou plusieurs actions (déplacement d'un objet vers un site) potentiellement bénéfiques ou un critère d'évaluation du positionnement d'un objet sur un site du point de vue de ses besoins. Ce critère est établi à partir des besoins de l'objet qui sont mis en correspondance avec les propriétés exprimées par un site comme nous l'avons décrit au chapitre **??**.

Avec ce serveur, nous respectons la *Trading Object Service Specification* de CORBA [**?**, **?**]. En effet, l'IDL standard des propriétés est utilisé ainsi que les opérateurs de l'*OMG Constraint Language*. Le serveur ORM permet de prendre en compte la problématique de l'hétérogénéité au niveau du site dans un même domaine. Les besoins applicatifs pris en compte sont divisés en deux catégories : les contraintes et les préférences. Dans cette optique, les contraintes sont associées au minimum de ressources nécessaires et adéquates pour qu'un objet s'exécute dans le système. Inversement, les préférences sont des informations facultatives pour permettre d'affiner la connaissance des besoins de l'objet pour une meilleure décision de placement au niveau du service de décision.

La notion de préférence est divisée en deux parties :

– les préférences liées à l'environnement matériel et système que nous appelons *préférences* comme le type de processeur, la capacité mémoire, le système d'exploitation, etc.
– les préférences logicielles que nous appelons *coallocations*. La coallocation est composée d'objets applicatifs et/ou d'applications avec lesquels l'objet sélectionné désirerait cohabiter.

Une liste d'objets locaux est construite dans chaque serveur ORM. La gestion des ressources offertes par la machine hôte est également prise en compte. Le serveur ORM peut ainsi évaluer les objets par rapport aux ressources locales du site. Cette évaluation sera prise en compte ensuite soit par le service de décision pour déterminer si un placement est valide, soit par le serveur ORM pour valider le placement d'un objet ayant modifié la liste de ses besoins.

Le serveur ORM utilise des fonctions pour la mise à jour de ses informations et la communication avec d'autres serveurs ORM : des fonctions de calculs et des fonctions d'administration. Ce serveur est un module informationnel, il peut, pour fournir un critère au module de décision, évaluer des objets par des modes de calculs que nous exposons ci-après.

Le calcul des contraintes

Le respect des contraintes du site est le minimum à prendre en compte pour le placement ou la migration des objets. Le calcul de contraintes est donc une part importante du serveur qui doit assurer que les objets disposent toujours du minimum nécessaire.

La liste des propriétés du site est mémorisée sous la forme d'un champ contenant le nom de la propriété et d'un champ contenant une note associée à la propriété. Les propriétés du site sont comparées aux besoins de l'objet. On vérifie, une à une, que chaque contrainte de l'objet est présente dans la séquence des contraintes du site. Si la contrainte est trouvée dans la séquence du site, on compare alors sa valeur avec celle de l'objet tout en prenant en compte l'opérateur logique de contraintes de l'objet. Si l'opération logique entre la valeur de propriété du site et celle de l'objet est vraie alors cette contrainte est satisfaite et on continue la comparaison avec les autres contraintes de l'objet et du site. On obtient les règles de calculs suivantes :

– si la séquence de contraintes de l'objet satisfait la séquence de propriétés de la machine, l'évaluation vaut 0. Cela signifie que toutes les contraintes liées à l'objet sont équivalentes aux propriétés du site ou bien qu'elles sont incluses dans cet ensemble. C'est la condition *sine qua non* pour continuer les calculs sur les préférences et la coallocation.
– si il n'y a pas satisfaction de contraintes, l'évaluation vaut -1, c'est à dire que le site ne répond pas aux exigences de l'objet, les contraintes deviennent l'équivalent d'un veto. Par conséquent, il est inutile de poursuivre les calculs sur les préférences et la coallocation. On peut proposer l'objet concerné pour une migration car son environnement actuel ne lui convient pas.

Il faut noter que si une contrainte d'objet ne se retrouve pas dans les propriétés du site (nom de la contrainte introuvable), l'arrêt est immédiat car les contraintes de l'objet ne seront pas satisfaites. Le calcul des préférences et de coallocation de l'objet est fait lorsque la somme des évaluations des contraintes vaut 0.

Le calcul des préférences

Le calcul des préférences utilise la même base que le calcul des contraintes. Il faut simplement utiliser la séquence de préférences de l'objet et la comparer à la séquence de propriétés du site. Cependant, cette évaluation peut être considérée comme étant un paramètre d'affinage du critère de contraintes. L'évaluation des préférences se situe dans l'intervalle [0, 10] avec 0 la plus faible note possible et 10 la plus forte. Si l'objet n'a pas de préférences, on lui attribue la note max de 10. On utilise également des opérateurs logiques pour mener à bien ces calculs. Nous détaillons ici les différents opérateurs et le calcul associé :

Soient *Prop* la valeur de la contrainte du site et *Pref* la valeur de la préférence de l'objet.

Opérateur	`Calcul associé`
$>$	((Prop - (Pref + 1)) + 10) / 2
$<$	(((Pref - 1) - Prop) + 10) / 2
$\geq$	((Prop - Pref) + 10) / 2
$\leq$	((Pref - Prop) + 10) / 2
$!=$	Vrai = 10 ou Faux = 0
$==$	Vrai = 10 ou Faux = 0

TABLE 7.1 – Opérateurs et calculs associés des préférences

Ces calculs permettent l'évaluation des préférences d'un objet par rapport aux propriétés exprimées par le site. Nous avons fait ce choix de calcul des préférences car une analyse sémantique des préférences nécessiterait une analyse lexicale assez poussée. Après avoir évalué les préférences, il faut calculer le critère de coallocation.

Calculs de coallocation

Le calcul de coallocation utilise un principe différent des deux critères précédents. L'évaluation de la coallocation se situe, également, dans l'intervalle [0, 10] avec 0 la note la plus faible et 10 la plus forte. Si l'objet n'a pas de désirs de coallocation, on lui attribue la note moyenne de 5. Le besoin de coallocation est exprimé grâce à un nombre compris entre 1 et 5 d'opérateurs "+" ou "−". Les calculs sont les suivants :

Soit n le nombre d'opérateurs avec $1 \leq n \leq 5$.

Soit C la présence sur le site d'un objet ou d'une application de coallocation.

Nombre d'opérateur	`C`	`¬C`
$n+$	$5 + n$	$5 - n$
$n-$	$5 - n$	$5 + n$

TABLE 7.2 – Opérateurs et calculs associés des coallocations

Il faut noter que cette évaluation reste dans l'intervalle [0, 10]. Cette évaluation est le dernier critère à être pris en compte par le serveur ORM. Il ne reste plus qu'à agréger ces trois critères pour construire une seule variable représentative des trois calculs.

Agrégation des critères

L'agrégation des critères de préférence et de coallocation est possible si la satisfaction des contraintes est réalisée. On calcule alors une moyenne de ces deux critères auxquels sont associés des poids identiques. Cependant, rien n'empêche de définir des poids différents pour favoriser un critère particulier. On peut résumer l'agrégation de manière suivante :

Soit f une fonction de calcul, E le résultat global de l'évaluation.

Soit n le nombre de préférences et m le nombre de coallocations.

- Si (f(contraintes) == -1) $\Rightarrow E = -1$
- Si (f(contraintes) == 0) $\Rightarrow E = \frac{\sum_{i=0}^{i=n} f_i(Preference_i) + \sum_{j=0}^{j=m} f_i(Coallocation_i)}{n+m}$ et $E \in [0, 10]$

Nous avons modélisé les calculs des différents critères des objets présents dans le serveur ORM. Le critère E, issu de l'agrégation des critères de contraintes, de préférences et de coallocations, sera ensuite envoyé au serveur de décision afin de construire le critère global, en utilisant les informations du serveur de charge et du serveur de relations. A partir de l'évaluation globale par le service de décision, une décision de placement ou de migration sera mise en place. La partie suivante présente le mécanisme de décision basé sur l'analyse multi critères.

7.1.5 Le service de décision

Le service de décision ne contient pas de données hormis les paramètres de son algorithme, c'est-à-dire les valeurs de veto, des coefficients, de la base de temps et du quota qui lui est accordé. Son fonctionnement est simple : les valeurs des critères obtenues auprès des services d'information sont comparées au veto les concernant, puis, si la comparaison est satisfaisante, elles sont intégrées dans le critère de synthèse qui est finalement comparé au seuil d'indifférence. Si ce critère de synthèse est supérieur au seuil, le déplacement est tenté. Le service de décision prend contact avec son homologue sur le site de destination. Si ce dernier n'est pas déjà en cours de transfert d'un objet, et si sa charge est toujours satisfaisante, l'objet est migré.

Lors de sa migration, un objet signale son départ au gestionnaire de relations local, et son arrivée au gestionnaire de relations distant. Il maintient alors ce dernier informé de ses nouvelles communications.

FIGURE 7.6 – Le service de décision

Le service de décision dispose comme les trois services d'information de trois interfaces (figure **??**) :

1. une interface d'administration permettant de modifier ses paramètres.
2. une interface permettant au service de gestion de la charge de lui signaler des traitements exceptionnels.
3. une interface permettant de collaborer avec un autre service de décision pour garantir la validité d'une migration.

Deux stratégies sont employées pour le placement d'un objet :

– La première est proactive. Elle est appliquée par le module de décision (POM) qui consulte les moniteurs d'informations tels que le COMOBS, le LOAD et l'ORM pour déplacer un objet. La figure **??**, décrit cette stratégie qui utilise le graphe relationnel du COMOBS pour identifier l'objet local, et la cible distante de la migration. Si la source est surchargée, cette stratégie est appelée également source-initiative.
– La deuxième est réactive. Elle est appliquée par les alarmes qui sont envoyées par les moniteurs d'informations tels que le LOAD et l'ORM au module de décision (POM). Lorsqu'un site sous-chargé est trouvé, le serveur LOAD identifie la source qui a la plus haute soumission (le site plus surchargé) dans sa liste et il envoie une alarme à la source. Le POM du site source cherche alors, en utilisant le graphe relationnel, l'objet qui a la relation la plus faible pour déplacer. Il utilise effectivement une stratégie serveur-initiative. Les serveurs ORM et COMOBS peuvent envoyer également une alarme au POM pour déplacer un objet qui a changé ses besoins de ressources (contraintes) dynamiquement et ne peut pas être satisfait par les ressources locales ou dont le comportement relationnel a changé.

L'ensemble de ces services est regroupé dans un seul processus-serveur, appelé Gestionnaire d'Objets ou GO. Par la suite, nous l'appelerons également GO-site par opposition au serveur GO-domaine qui sera en charge de la gestion des objets au niveau des domaines. Nous développons la mise en place de ce serveur dans la partie suivante.

7.2 La gestion interdomaine

La problématique de la gestion de composants entre des domaines différents a volontairement été laissée à l'écart de nos discussions jusque là ; ceci pour deux raisons : la première est qu'elle est globalement proche et la seconde est qu'elle a ses propres particularités qui ne lui permettent pas d'être entièrement assimilé à la gestion interne à un domaine. La gestion interne à un domaine, que nous appelerons intradomaine, voit l'ensemble du réseau comme un seule ressource utilisable à son gré. L'introduction de la notion de domaine dans le modèle précédent amène, aux frontières des domaines, une cassure dans nos hypotèses de communication. Ceci se traduit par un intérêt différent dans les deux cas :

– dans les domaines, le but est d'optimiser l'utilisation des ressources locales au domaine sans distinction des sites. Nous privilégions donc l'optimisation globale du système par rapport à l'optimisation limitée à une application.
– a l'opposé, chaque domaine est vu comme un système indépendant qui essaye d'optimiser ses paramètres sans prise en compte globale. Entre les domaines, le but est donc d'optimiser pour les clients locaux à un domaine la qualité et le coût d'accès aux services, en respectant les contraintes des communication à longue distance.

Dans cette partie nous donnons donc un cadre à la notion de domaine avant d'aborder la problématique qui y est liée. Nous développons ensuite la réalisation du service d'administration

FIGURE 7.7 – Le placement dans les domaines : stratégie proactive (POM)

FIGURE 7.8 – Le placement dans les domaines : stratégie réactive (LOAD)

FIGURE 7.9 – Le placement dans les domaines : stratégie réactive (ORM)

interdomaine.

7.2.1 La notion de domaine

Notre travail s'appuie sur la définition des domaines donnée par RM-ODP. Un *domaine* $< X >$ est un ensemble d'objets, chacun de ces objets étant relié à un objet contrôleur par une relation caractéristique $< X >$. En fait, on retrouve la notion de groupe avec en plus un objet contrôleur. Celui-ci peut décider de l'identité des objets qui appartiennent au domaine associé. En général, l'objet contrôleur n'est pas un membre du domaine associé. L'objet contrôleur peut administrer différentes politiques dans un domaine associé. Par exemple, les objets dans un domaine de sécurité sont soumis à la politique de sécurité décidée par un contrôleur de sécurité. Un domaine peut avoir des sous-domaines qui sont des sous-ensembles du domaine associé.

D'un point de vue pratique, nous avons fait le choix de considérer un domaine comme un réseau local. Les propriétés qui y sont attachées permettent de supposer une certaine homogénéité, plus particulièrement au niveau de l'environnement d'exécution des composants, de la politique d'administration et de la sécurité. Cette considération n'est donc pas stricte mais permet d'en donner une représentation pratique.

La notion de domaine est également présente dans la norme CORBA [**?**], quoique parfois floue. D'après la norme, un domaine est une partie indépendante dont les composants (machines) possèdent des caractéristiques communes et pour lesquelles des règles communes sont observées. La notion de domaine ne repose donc pas uniquement sur les caractéristiques matérielles des réseaux telles que nous les avons définies précédemment. En particulier, il est possible de définir les domaines selon deux critères, l'un technologique et l'autre administratif.

7.2.2 Problématique du placement interdomaine

Il est admis qu'un réseau local est géré intérieurement d'une manière globale, centralisée, par un administrateur chargé de s'assurer que les applications qui s'y exécutent disposent de ressources suffisantes. Par contre, les différents réseaux locaux ne sont pas gérés extérieurement d'une manière globale, chacun est indépendant. Ceci implique qu'il n'y a pas uniformité dans la disposition des ressources sur l'ensemble des réseaux. La recherche des ressources nécessaires à une application est donc généralement réalisée "à la main" pour l'implantation d'une application répartie.

Le développement d'applications à base de composants dans un seul domaine pose le choix du découpage des données et des traitements entre les clients et les serveurs mais tient très peu compte des contraintes de placement du fait de la disponibilité et de la sûreté du réseau local. L'exploitation pose le problème du placement des différents composants entre les ordinateurs connectés au réseau. Ce déploiement se fait en évaluant la charge (processeur, disque) engendrée par chacun, ce qui permet de définir le placement des composants. A l'opposé, le développement d'applications à base de composants dans un contexte multidomaine est dépendante de son déploiement. Étant donné les différences de niveau de service (débit, sécurité, sûreté, etc) entre un réseau local et un réseau à longue distance, il semble naturel de prendre ces contraintes en compte lors du développement d'une application, celles-ci ayant des répercussions directes sur la disponibilité et les performances du logiciel. Sur ces bases, le déploiement et l'exécution d'une application ne peuvent se faire qu'en respectant les volontés

et les contraintes fixées au moment du développement.

Pour les applications monodomaine qui utilisent toutes le même réseau (local), la seule optimisation du débit de communication ne peut se faire qu'en regroupant sur un même site les composants de l'application qui communiquent beaucoup. Pour les applications multi-domaines, l'optimisation des débits de communication sera dépendante du type de réseau utilisé. L'optimisation nécessite alors l'introduction d'une topologie avec des débits associés, des fonctions de coûts, etc. Ainsi, le critère de communication est assurément le critère le plus important.

Certaines applications ont besoin d'une garantie quant à la confidentialité de leurs échanges ou de se protéger contre les intrusions. L'introduction de la notion de domaine dans nos applications à base de composants rend ce problème plus crucial. De plus, les différentes classes et politiques de sécurité implantées dépendent du support offert par l'ORB. Il semble donc qu'à ce titre tous les sites et domaines ne soient pas équivalents. Les caractéristiques de sécurité des domaines vont donc influer sur le placement initial d'une application. Il n'y a cependant pas de dynamicité dans la définition de la sécurité pour CORBA, ce qui limite l'intérêt du placement dynamique à la dynamicité de l'application. Les applications doivent être en mesure de définir les contraintes de sécurité dont elles ont besoin. Ces contraintes peuvent être résolues au placement initial en fonction de caractéristiques des domaines mais également prisent en compte par un service de placement dynamique pour assurer leur respect dans le comportement dynamique d'une application.

7.2.3 Architecture de la gestion de domaine

Étant donné la structuration hiérarchique en sites et domaines des réseaux locaux, nous avons choisi de développer un service indépendant, centralisateur des informations au niveau d'un domaine et représentant vis-à-vis des autres domaines.

FIGURE 7.10 – Architecture d'un service de placement

La figure **??** donne un aperçu de la répartition des serveurs. La structuration est hiérarchique, avec une gestion uniforme sur les sites du réseau local, basée sur les serveurs POM et les moniteurs d'information, et des gestionnaires, communs à l'ensemble de ces sites, qui gèrent les échanges avec les autres domaines. L'architecture de la gestion interdomaine repose sur trois composants :

- le gestionnaire de domaine (Domain Authority) qui est chargé de la gestion des interactions entre domaines sous forme de contrats. Il offre les services pour la mise à jour et le stockage des contrats.
- le gestionnaire de contraintes de domaine (Constraint Enforcer) qui gère la mise à jour, le stockage et la vérification des contraintes liées au domaine.
- le dépôt de contraintes applicatives (Constraint Repository) qui centralise au niveau du domaine les contraintes liées aux objets.

Par rapport au modèle d'administration décrit au chapitre **??**, le gestionnaire de domaine est un module de décision et les deux autres des modules d'information. Les trois composants sont regroupés dans un processus serveur que nous appelons le GO-domaine.

Dans la suite de cette partie, nous donnons une description détaillée de ces modules et des traitements qu'ils réalisent.

7.2.4 Le gestionnaire de domaine

Le gestionnaire de domaine est le représentant du domaine local pour le placement interdomaine et la diffusion des services. Il consitue donc le point d'entrée du domaine et coopère avec les autres gestionnaires de domaine pour mettre en place une politique de placement cohérente. Puisqu'il intervient pour rendre transparents plusieurs niveaux (placement, gestion des contraintes et diffusion des services), ce serveur est appelé *Domain Authority (DA)*. Il peut d'ailleurs servir à implanter d'autres types de coopération entre les domaines, comme par exemple servir d'intermédiaire aux modules de sécurité de différents domaines. Nous présentons ici ses différentes fonctionnalités.

La gestion des contrats

A l'initialisation d'un domaine, le gestionnaire de domaine échange son *Interoperable Object Reference* (IOR) avec les autres gestionnaires de domaine autorisés, choisis par l'administrateur de domaine. Après l'échange des IOR, le DA peut communiquer avec d'autres DA via le service de nommage local.

Les DA établissent entre eux des *contrats* qui sont gérés par les administrateurs de domaine. Les contrats sont des fichiers contenant les services qui sont accessibles depuis d'autres domaines, les conditions d'accès et les domaines pour lesquels l'accès est autorisé. Ils participent donc à la mise en place de la politique de sécurité globale. Les administrateurs établissent deux contrats par domaine : le *Import Contract* et le *Export Contract*. Le Import Contract fixe les services qui peuvent être importés et les domaines depuis lesquels ils peuvent être importés. Le Export Contract fixe les services qui peuvent être exportés et les domaines vers lesquels ils peuvent être exportés. Le Domain Authority communique avec les autres Domain Authority pour importer leurs services et il exporte les siens vers ces domaines. Quand un Domain Authority est initialisé, il cherche à établir des contrats avec les autres Domain Authority.

Ainsi les objets ne peuvent utiliser des services distants que si ces services sont définis dans le Import Contract ou si ils ont explicitement été exportés par le service lui-même. Ceci permet de garantir la sécurité d'accès aux services entre les domaines.

La syntaxe d'un contrat est donnée, dans la notation Backus-Naur étendue par :

```
<spécification>  ::= <définition>*
<définition>     ::= <contrat><contenu>+
<contrat>        ::= 'IMPORT' | 'EXPORT'
<content>        ::= 'DOMAIN' <nom_de_domaine><service>+
<nom_de_domaine> ::= <identificateur>
<service>        ::= ','<identificateur>
```

La mise à jour des contrats peut s'effectuer dynamiquement. Par exemple, lorsqu'un nouveau domaine est initialisé. Le Domain Authority (DA) offre une interface pour la mise à jour des contrats.

Le placement des objets

La décision de déplacement vers un domaine est basée sur les informations issues du COMOBS. En fonction de celles-ci, le POM choisit un objet à déplacer et demande au Domain Authority de le migrer. Le Domain Authority local en demande la permission au Domain Authority cible. La vérification de l'adéquation des propriétés du domaine aux contraintes de l'objet est réalisée par le Constraint Enforcer, qui est alors invoqué avec la liste des contraintes associées à l'objet. Si l'autorisation de migration n'est pas accordée, par exemple parce que les contraintes de l'objet ne peuvent être satisfaites, alors l'objet candidat suivant est évalué. Si l'objet est autorisé à se déplacer, alors le Domain Authority copie le fichier exécutable de l'objet choisi dans le répertoire d'importation du domaine cible, si ce fichier n'existe pas encore. Pour faire la migration, le service de cycle de vie et les *factory* sont utilisés [**?**]. Il est possible de permettre la migration dans un environnement hétérogène, si le fichier exécutable de l'objet est accessible pour le type de processeur du domaine cible. Ensuite, un *factory* crée un nouveau processus puis transfert l'objet et son état dans un domaine cible.

Le site sur lequel est placé l'objet dans le domaine cible dépend de la politique de migration choisie par ce domaine. Si la politique de migration n'autorise le placement que sur un seul site, pour des raisons de sécurité que nous avons déjà évoquées, alors l'objet s'exécute sur ce site. Si la politique de migration autorise le placement sur l'ensemble des sites du domaine, c'est le Domain Authority cible qui choisit le site sur lequel il place l'objet. La décision est laissée à sa responsabilité. Néanmoins, pour ne pas interférer avec le travail des GO-sites locaux, la décision intègre une consultation de ceux-ci.

Si le domaine distant ne peut pas satisfaire les besoins de l'objet, le gestionnaire de contraintes de domaine peut proposer un autre domaine (dans sa liste de contrats) pouvant satisfaire les besoins de l'objet au DA source. Le DA source informe le POM en lui faisant une nouvelle proposition. Si le POM trouve que cette proposition est satisfaisante par analyse multicritère, il demande au DA de migrer l'objet. Si la décision n'est pas satisfaisante, la migration est impossible dans ce cas.

Les propriétés associées à un domaine sont des attributs tels que le niveau de sécurité, le type de système d'exploitation, le débit d'une ligne et le type d'architecture matérielle d'un domaine. L'introduction d'un second niveau qui prend en compte les propriétés permet de mettre en place une gestion spécifique au domaine. Ces attributs sont gérés par le Constraint Enforcer que nous présentons dans la partie suivante.

7.2.5 Le Constraint Enforcer

Les propriétés d'un domaine sont gérées par le Constraint Enforcer. Il agrège les critères issus des propriétés d'un domaine à partir des propriétés de l'ensemble des machines gérées par le serveur *ORM*. Puisqu'il gère les propriétés d'un domaine, le Constraint Enforcer est en mesure d'évaluer la cohérence entre ces propriétés et les contraintes d'un objet (d'où son nom). Ce serveur offre une interface pour positionner et lire dynamiquement les valeurs des propriétés du domaine. Il permet également au Domain Authority de rechercher les contraintes associées à un objet et de les envoyer à un autre Domain Authority pour évaluer une possibilité de migration.

La vérification exprimée est simple mais dépend du type des propriétés. Certaines sont qualitatives, par exemple le type de système d'exploitation, dans ce cas la vérification des contraintes nécessite que l'attribut recherché soit défini sur le domaine. D'autres propriétés sont quantitatives, par exemple le niveau de sécurité nécessaire, dans ce cas la vérification des contraintes nécessite qu'un niveau d'attribut minimum soit défini pour le domaine. Le Domain Authority peut rechercher les contraintes associées à un objet et les envoyer à un autre Domain Authority pour évaluer une possibilité de migration. La déclaration des contraintes est faite par l'objet directement au Constraint Repository.

Les princiaples ressources que peut fournir un domaine sont les suivantes :

la sécurité qui est garantie dans le domaine, avec le niveau qui est associé, par exemple en terme des classes de sécurité définies dans [**?**] ;

l'architecture : les domaines sont supposés hétérogènes, il est donc possible de caractériser l'architecture d'un domaine de manière à garantir les possibilités d'exécution ;

les connexions extérieures avec le type de protocole utilisé et le débit associé à chaque ligne ;

les périphériques : liste des différents périphériques, on peut supposer qu'un type de périphérique est disponible pour l'ensemble du réseau local, la transparence d'accès étant assurée par le système d'exploitation ;

les serveurs : liste des serveurs présents.

Le vérificateur de contraintes non seulement doit contrôler des contraintes, mais peut également jouer un rôle de courtier pour trouver le meilleur site en utilisant l'analyse multicritère de propriétés qualitatives ou quantitatives au niveau du site.

7.2.6 Le dépôt de contraintes applicatives (Constraint Repository)

En l'absence de serveur ORM ou en coopération avec ceux-ci, les contraintes associées à un objet sont enregistrées dans le serveur de contraintes (Constraint Repository). Les principales contraintes prises en compte dans notre implantation concernent le niveau de sécurité, le débit supporté vers les autres domaines, le système d'exploitation gérant les sites du domaine, le type d'architecture de ces sites, ou d'autres en fonction de besoins particuliers. La déclaration des contraintes est faite par l'objet directement au Constraint Repository.

Le dépôt permet au Domain Authority de rechercher les contraintes associées à un objet et de les envoyer à un autre Domain Authority pour évaluer une possibilité de migration. La déclaration des contraintes est faite par l'objet directement au Constraint Repository.

Toutes les contraintes associées à un objet sont enregistrées dans le serveur de contraintes (`Constraint Repository`). Les principales contraintes prises en compte dans notre implantation concernent le niveau de sécurité (de S1 à S3), le débit supporté vers les autres domaines (de B1 à B3), le système d'exploitation gérant les sites du domaine et le type d'architecture de ces sites. Le serveur de contraintes mémorise les contraintes des services dans le format suivant :

```
<service>        ::= <nom_de_service> ',' <contrainte>
<nom_de_service> ::= <identificateur>
<contrainte>     ::= <securite> ',' <debit> ',' <OS> ',' <plate-forme>
<securite>       ::= 'S1' | 'S2' | 'S3' | ''
<debit>          ::= 'B1' | 'B2' | 'B3' | ''
<OS>             ::= 'solaris' | 'NT' | 'linux' | ''
<plate-forme>     ::= 'x86' | 'sparc' | 'powerpc' | ''
```

Le serveur de contraintes offre une interface de modification dynamique des contraintes qui permet de spécifier, de modifier ou d'accéder aux contraintes associées à un objet.

Le Constraint Repository peut facilement être remplacé par un ensemble de serveurs coopérants de type ORM pour implanter une politique de gestion des contraintes internes à un domaine. Dans ce cas, la déclaration des contraintes associées à l'objet se fera au niveau du serveur local à un site. Comme nous l'avons vu, cette architecture permet de prendre en compte des contraintes plus variées et en nombre plus important. Néanmoins, cette implantation nous permet, d'une part, d'évaluer l'intérêt de l'utilisation des contraintes dans le placement interdomaine et, d'autre part, de regrouper à ce niveau les informations spécifiques à l'optimisation du placement et de l'exploitation des composants.

7.2.7 Les traitements

Pour mettre en place le placement automatique entre les domaines, le GO-Domaine implante les méthodes suivantes :

mise à jour des objets : après leur calcul périodique, les serveurs COMOBS transmettent au GO-Domaine les objets qui possèdent une relation vers l'extérieur ayant mis en évidence une modification de l'équilibre relationnel. L'appel de cette méthode a pour effet de mettre à jour la liste des objets. Son appel est périodique pour chaque GO-Site mais apériodique du point de vue du GO-Domaine, les GO-Sites n'étant pas synchronisés. Les paramètres de cette méthode sont l'identificateur d'objet, la liste des relations et la liste des contraintes. L'invocation de cette méthode déclenche une analyse des relations.

mise à jour des ressources : la mise à jour de la liste des ressources disponibles dans le domaine permet la gestion de cette liste. Les paramètres de cette méthode comprennent un indicateur ajout/suppression, un identificateur de la ressource et son niveau de disponibilité (pour les contraintes qualitatives). L'invocation de cette méthode est réalisée par un administrateur au moyen d'outils de gestion des ressources. L'appel est apériodique.

validation des contraintes : cette méthode est invoquée avant une migration et en cas de modification de contraintes. Avant de migrer un objet, le GO-Domaine vérifie auprès du GO-Domaine du domaine cible si les contraintes liées à l'objet sont compatibles avec

les ressources du domaine. Lors d'une modification des contraintes liées à un objet, le serveur de contraintes vérifie si le domaine d'exécution respecte les nouvelles contraintes. La réponse à cette invocation se limite à un accord ou un désaccord. Le paramètre de cette invocation est la liste des contraintes liées à l'objet. L'invocation est apériodique.

analyse des relations : périodiquement le GO-Domaine évalue les objets dont il dispose. Le résultat de cette évaluation est un objet candidat à la migration. Le GO-Domaine envoie alors la liste des contraintes liées à cet objet au GO-Domaine cible afin de vérifier l'adéquation du domaine. Si le domaine accepte, le GO-Domaine réalise la migration, l'objet est déplacé vers le site de l'objet avec lequel il a le plus d'échanges.

7.2.8 La gestion de la sécurité

Le Domain Authority est chargé d'implanter les vérifications nécessaires aux migrations. Il gère également le problème de la sécurité dans l'accessibilité aux services. Il ne gère pas la sécurité au niveau des relations, ce travail étant réalisé, comme cela est précisé dans la spécification CORBA, par le module de sécurité. Il est donc nécessaire, pour réaliser une politique globale et cohérente, de définir les interactions entre ces deux serveurs. Le Domain Authority s'occupe de la déclaration d'accessibilité des services externes au domaine auprès du module de sécurité. Réciproquement, le module de sécurité doit pouvoir vérifier ou demander les droits d'accès à un service au Domain Authority. Cette structure permet de rendre transparente l'utilisation du module de sécurité aux objets. En effet, un objet se contente de déclarer son accessibilité au Domain Authority. Le respect de cette accessibilité est ensuite garantie par le module de sécurité sans intervention supplémentaire de la part de l'objet.

La sécurité a été présentée jusque là comme une ressource du système. Cependant, elle présente un intérêt particulier dans le cadre du placement entre les domaines. C'est un problème qui ne peut être contourné dans un contexte multidomaine de prestation de services. Une attention particulière doit donc y être portée. Elle est vue différemment suivant le niveau auquel elle s'applique :

- au niveau des domaines, nous définissons la sécurité comme une valeur attribuée au domaine, cette valeur traduisant le niveau de sécurité garanti globalement dans le domaine.
- La prise en compte des contraintes de sécurité en même temps que la volonté de mettre en œuvre un placement facilitant les communications, nous a conduit à donner la possibilité au domaine de dédier un site à l'accueil des serveurs qui sont importés dans le domaine. Cette technique suppose de réaliser une surveillance accrue de ce site et de le soustraire à la gestion de charge du domaine. Néanmoins, le site se trouvant dans le domaine, les communications sont optimisées. Un domaine dispose alors des choix suivants :
  - ne jamais accepter l'importation de serveurs,
  - proposer un site d'accueil sur lequel sont placés les objets importés,
  - autoriser le placement des serveurs importés sur n'importe quel site du domaine.
- Au niveau des applications, la sécurité est traitée sous deux formes : la sécurité du domaine, donc du site, dans lequel elle se trouve et la sécurité des liens qu'elle utilise. La définition de ces données se fait dans le GO-Site par une définition dynamique des contraintes.

7.3 Conclusion

Nous avons réalisé l'implantation d'une partie du modèle proposé au chapitre précédent. D'abord dans le cadre d'un réseau local en limitant les contraintes liées au réseau, puis dans le cadre de réseaux étendus. Pour chacune de ces plates-formes, nous avons défini la problématique qui y est liée et une architecture de service adaptée. Ainsi, dans le cas de domaines, il a été plus simple de centraliser la gestion des contraintes et des relations au niveau applicatif malgré les inconvénients liés à la centralisation de l'information.

Le travail présenté dans ce chapitre a donc une conotation beaucoup plus technique. Cependant, l'expérience de la mise en oeuvre des services et des algorithmes de placement dynamique est une étape indispensable dans la connaissance des problèmes qui y sont liés. Comme je l'ai dit en introduction la simulation de ce type de service s'avère difficile et peu précise. Ce service étant destiné à la gestion de comportements dynamiques c'est dans sa mise en oeuvre dynamique que nous avons pu mieux comprendre les propriétés qui y étaient attachées.

La mise en oeuvre du service au dessus d'une plate-forme CORBA nous a permis d'en apprécier le support et d'en identifier les limites.

- le modèle objet et d'invocation distante offre un cadre adapté au développement d'objets-serveurs coopérant pour fournir un service à l'application. Ce qui est le cas des différents serveurs que nous avons développés. Cependant, le manque de précision dans la norme quant au support d'interfaces multiples par un même objet fait que cette fonctionnalité n'est pas systématiquement mise en oeuvre par l'ORB. Il en résulte un support difficile de la notion de vue fonctionnelle et de groupe d'interaction tel que nous l'avons défini pour les différents éléments de notre architecture. Sur ce point, la version 3 de la norme, en cours de finalisation, apporte une réponse avec la notion de facette.
- les services spécifiés dans l'architecture OMA n'offrent pas, à priori, de cadre adapté à l'implantation de modules d'information. Les spécifications qui pourraient servir de support à un tel service sont celles des traders ou des services d'évennements. Mais leur lourdeur et l'évidence de leur conception centralisé les rendent coûteux à consulter et mal adaptés. La nécessité d'un tel service, en support à l'administration, paraît pourtant évidente.
- les interactions entre les modules d'information et les modules de décision reposent naturellement sur la communication gérée par le bus CORBA. Le déclanchement d'une alarme par l'un des modules de décision doit pouvoir être réalisé en utilisant le service d'évennement. Nous n'avons cependant pas utilisé cette possibilité. Il serait probablement intéressant de l'explorer car elle permet de généraliser facilement la diffusion des alarmes à différents sites. Elle pourrait donc servir de support à d'autres algorithmes de répartition de la charge.
- l'architecture OMA inclut la définition d'un service de cycle de vie (`Lifecycle Service`) permettant le positionnement au sein de l'architecture de méthodes de déplacement des objets gérés. Par contre, il existe une contradiction entre la définition de ce service et le support inexistant dans la norme à la redirection des communications vers l'objet déplacé. Ici encore, la troisième version de la norme apporte des éléments de réponse à travers le modèle de composants. Les communications d'un composant étant prises en charge par le container, les procédures de suivi des communication peuvent y être implantées.
- la notion de domaine introduite par les spécifications de la sécurité est en phase avec la définition que nous en avons faite. Elle traduit une réalité qu'il est difficile de contourner dans les problèmes de placement dynamique.

- la description des objets dans CORBA n'offre pas de support particulier à l'association de propriétés avec les objets, propriétés qui peuvent être utiles pour l'administration de ceux-ci en indiquant des besoins ou des caractéristiques particulières. L'association de fichier de description est possible dans les modèles de composants. Ces fichiers constituent un support intéressant pour la mise en place de nos besoins applicatifs.
- le support à l'interaction entre des services placés sur différents noeuds et la découverte dynamique de l'architecture physique sont insuffisant dans la norme. Le produit COOL-ORB offrait, par exemple, des interfaces au niveau du noeud et du domaine permettant le parcours des ensembles sous forme de listes. Ce type de fonctionnalité est un plus dans la gestion du placement ou pour l'administration des objets puisque ces services vont agir en terme de site et de domaine. Les normes de composants n'apportent pas de réponses satisfaisantes à ces lacunes puisqu'aucune représentation physique n'est inclue.

Chapitre 8

Analyse et performances

Dans ce chapitre, nous allons montrer comment nous avons réalisé les tests nécessaires à la validation de notre travail. Nous avons choisi de mettre en œuvre des tests réels en utilisant un simulateur d'applications au dessus d'une implantation du gestionnaire de composants car l'ensemble des paramètres à prendre en compte était trop important pour être traduit dans un simulateur de système. Une maquette a donc été réalisée au dessus d'un ORB. Cependant, la généralisation du travail pour d'autres environnements à objets est directe puisque le seul service utilisé est celui de nommage, disponible sur l'ensemble des environnements. Un visualisateur a également été développé pour analyser les résultats obtenus.

Le choix de nos tests est empirique, cependant nous utilisons une méthode basée sur les plans d'expérience pour en limiter le nombre et pour montrer les interactions entre les paramètres et les différents types d'applications. Les tests interdomaines sont réalisés pour montrer le gain qualitatif, tandis que les tests intradomaines sont effectués pour montrer l'aspect quantitatif et les comportements du gestionnaire d'objets avec différents jeux de paramètres.

8.1 Synthèse

Cette partie, très technique puisqu'elle décrit et analyse les résultats obtenus par notre maquette, présente cependant des intérêts quant à la démarche utilisée pour obtenir ces résultats. Notre contribution se situe à deux niveaux : les résultats et ce qu'ils expriment, d'une part, et la méthode utilisée pour le choix des paramètres du service, d'autre part.

En ce qui concerne les résultats, le premier que nous avons obtenu concerne les possibilités d'utilisation d'informations sur les communications dans l'optimisation du placement. Le deuxième concerne l'utilisation d'une procédure de décision multicritère pour compenser les effets d'un critère par ceux d'un autre. Il est ainsi possible de passer sous des seuils ou de lever des contraintes pour obtenir un meilleur placement. Le troisième, plus évident, concerne la variation des résultats du placement en fonction des paramètres utilisés dans l'observation. Ces trois résultats ont été donnés dans la thèse de Pascal Chatonnay. Le quatrième, et dernier, résultat concerne la dépendance entre les applications et les jeux de paramètres optimum pour leur gestion. Ceci suppose une adaptation nécessaire des procédures de décision aux appplications d'où l'idée de séparer le module de décision de définir un module d'administration plus général en charge des politques de placement.

En ce qui concerne la méthode utilisée pour le choix des paramètres, l'idée avait initialement été dévelopée par [**?**]. Elle permet un choix rigoureux des plans d'expérience en fonction des niveaux affectés aux paramètres ce qui permet de fiaibiliser les résultats. Dans un cadre expérimental tel que le nôtre (j'aime à rappeler que les applications dynamiques s'exécutant dans un environnement étendu et public tel que le WEB constitue un environnement expérimental au même titre que les environnements qui sont étudiés en physique) ces méthodes sont trop peu souvent utilisées et gagneraient à être reconnues. La mise en place et l'utilisation de ces méthodes ont été expérimentées par Ludovic Bertsch au cours de son DEA.

8.2 La définition du cadre

L'objectif de la réalisation d'une maquette est de montrer l'intérêt d'un service administrant le placement dynamique de composants dans un environnement mono ou multidomaine. Cependant, un certain nombre de contraintes sont imposées par le contexte matériel visé. Parmi les contraintes possibles, certaines ne permettent pas de mettre en évidence l'intérêt de la démarche, nous avons donc réalisé des choix. Dans cette partie, nous donnons et justifions les limitations imposées par la réalisation de la maquette.

8.2.1 Les domaines

Pour ce qui est de l'interconnexion des domaines, nous ne considérons que deux types de réseaux : les réseaux locaux et les réseaux à longue distance. Cette réduction est également simplificatrice, car elle nous permet de ne définir que trois cas distincts de communication :

1. les communications locales à un site, offrant toutes les garanties désirées en débit, sécurité, etc ;
2. les communications à l'intérieur d'un domaine, offrant des garanties de sécurité. Les communications sont multipoints et le débit est plus réduit ;
3. les communications entre les domaines n'offrant aucune garantie. Les communications sont point à point.

8.2.2 Les besoins applicatifs

Au chapitre **??**, nous faisons ressortir un nombre important de besoins applicatifs qui sont susceptibles d'intervenir dans le placement des applications. Nous ne cherchons pas à valider ici une approche complète mais plutôt à montrer que le placement automatique avec contraintes peut avoir un intérêt. Nous nous sommes donc limités à un petit nombre de contraintes. Par contre, pour permettre d'en montrer l'intérêt, nous avons choisi les contraintes sur les aspects qui nous intéressent. C'est-à-dire, l'accès aux ressources, la sécurité et la qualité de service.

8.2.3 La gestion de charge

La gestion de charge est un paramètre important du placement automatique dans un réseau local. Lorsqu'un objet doit être migré, le service de placement vérifie auparavant si les

conditions de charge sont bien conformes. Le problème est différent dans un environnement multidomaine. Nous sommes donc partis du principe que l'indépendance entre les domaines est peu propice aux échanges de charge, l'optimisation des communications ayant un rôle plus important dans ce contexte. Pour ces raisons, nous avons réalisé deux implantations. La première gère les réseaux locaux et prend en compte un nombre important de critères. Elle s'appuie sur une procédure de décision multicritère sommative. La seconde gère les échanges utilisant des réseaux d'interconnexion et utilise une procédure de décision hiérarchique. Les échanges de charge sont donc limités aux besoins des optimisations de débit.

8.2.4 La sécurité

La sécurité des systèmes interconnectés recouvre différents aspects comme nous l'avons montré dans le chapitre **??**. La prise en compte de ces différents aspects de manière précise entraîne une augmentation importante de la complexité des processus de décision. Certaines normes définissent cependant des classes de sécurité pour les systèmes informatiques. Puisque nous considérons qu'un domaine est homogène, nous supposons que le niveau de sécurité est le même dans tout le domaine. Les classes de sécurité définissent des niveaux que nous pouvons identifier par des numéros. La prise en compte des aspects de sécurité est donc limitée à la définition d'un niveau de sécurité nécessaire de la part de l'application et du niveau de sécurité offert par le domaine ceci est réalisé à travers des valeurs pouvant traduire les classes de sécurité telles qu'elles sont définies dans les normes.

8.2.5 La migration des objets

La migration des objets repose sur le service proposé par [**?**]. Ce service permet uniquement la migration d'objets serveurs. Cette contrainte limite évidemment les situations de test mais n'est pas incompatible avec nos buts. En effet, dans le modèle applicatif visé par ces tests, les clients s'exécutent sur des stations extérieures au réseau local. Ils représentent l'interface d'accès de l'utilisateur aux composants. Dans ce contexte, il n'est pas souhaitable de les migrer puisqu'ils ne pourraient plus rendre leurs services.

8.3 L'environnement de test

Pour valider la gestion automatique du placement entre les domaines, il est nécessaire d'utiliser une plate-forme matérielle composée d'au moins deux domaines. Cependant, pour valider l'introduction des contraintes dans le processus de décision, il est nécessaire de disposer d'au moins trois domaines. Chaque domaine étant composé d'au moins deux machines, les tests interdomaines sont réalisés sur au moins 6 machines.

Nous avons mis en œuvre un visualisateur pour permettre une vision graphique des résultats obtenus lors de nos tests. Les scénarios peuvent ainsi être rejoués et analysés après l'exécution des objets. Ce visualisateur permet également de réaliser des courbes synthétiques des performances d'un jeu de paramètres.

Nous avons également implanté une interface d'administration sur les serveurs d'administration d'objets afin d'augmenter notre capacité à évaluer un nombre important de paramètres

et de scénarios de test. Les procédures de décision des serveurs peuvent ainsi être modifiées dynamiquement pour s'adapter aux besoins des applications.

8.3.1 La plate-forme

Pour les tests, nous les séparons en deux parties : les tests interdomaines et les tests intradomaines. Pour les tests interdomaines, nous avons un ensemble de huit machines qui sont toutes reliées par un réseau Ethernet. Elles sont structurées en trois domaines. Le premier domaine est composé de quatre PC Intel qui utilisent le système d'exploitation x86-Solaris. Le deuxième et le troisième sont constitués de deux machines Sun Sparc qui utilisent le système d'exploitation Sparc-Solaris. Nous n'avons pas réalisé de test entre des domaines physiquement éloignés car nous ne disposions pas d'une telle plate-forme. De plus, nous voulons seulement montrer l'aspect qualitatif dans les tests interdomaines. Or, notre système et les applications tests peuvent s'exécuter dans le monde Internet.

Pour les tests intradomaines, nous utilisons le premier domaine qui est pratiquement homogène (PC Intel, x86-Solaris) avec quelques caractéristiques différentes telles que la vitesse de processeur.

8.3.2 Le simulateur d'applications

Le but du simulateur d'applications est d'imiter différents types d'applications pour mesurer la réponse du service de gestion sur les architectures applicatives. Développer une application réelle dans un environnement distribué nécessite beaucoup de moyens et de temps. De plus, une application ne donne pas nécessairement une vision générale de la variété des applications possibles.

Architecture du simulateur

FIGURE 8.1 – Architecture du simulateur

Le simulateur d'applications permet, à partir de fichiers de configuration, de lancer une application composée d'objets clients, serveurs ou client/serveurs. Les paramètres donnés aux objets sont alors, par exemple, leur durée d'exécution ou les serveurs invoqués. Dans la figure **??**, le simulateur d'applications est bâti sur quatre composants :

Starter : c'est le serveur de lancement des applications qui permet la création des objets. Les objets l'interrogent ensuite pour connaître leurs paramètres. Ces paramètres et la configuration des applications sont définis dans les fichiers de configuration.

Event : c'est le serveur des événements et des traces. Il mémorise le début et la fin de l'exécution des objets, les migrations des serveurs et des clients/serveurs.

ParamGo : il permet le paramétrage dynamique du POM, du LOAD, du COMOBS et de l'ORM. Les données utilisées par ce composant se trouvent dans des fichiers de paramètres.

Master : c'est le programme principal qui lance l'exécution des serveurs ParamGo, Events et Starter. Le Master peut lancer plusieurs tests successifs définis dans un fichier plantest. Chacun des tests est caractérisé par : un fichier de configuration de l'application lancée, un fichier de résultats généré par Event et un fichier de paramétrage des serveurs utilisés par paramGo.

Description des fichiers de simulation

Voici un exemple de fichier de paramétrage d'une application composée d'un serveur et un client :

```
s serv0 galaad perceval 5 1 CPU == INTEL 1 SECU > 3 0

c cli0 galaad yvain 1 2000 100 10 2 serv0 0    1000 0.1 10
                                   serv0 1000 1000 1   100000
```

Nous décrivons brièvement ici la syntaxe des fichiers de configuration en l'illustrant par notre exemple. La première lettre d'une ligne donne le type d'objet (`s` pour serveur, `c` pour client et `i` pour client-serveur), puis son nom. Ici nous avons un serveur (`serv0`) et un client (`cli0`). Nous définissons ensuite le domaine `domainName` (ici `galaad` pour les deux objets) et les sites `siteName` (respectivement `perceval` et `yvain`) d'exécution des objets. Les clients attendent `delay` secondes (ici `1`) avant de commencer, ils consomment du temps CPU avec les mesures `cpuConsum` (ici `100`) et ils utilisent les IO (entrée-sortie) avec les mesures `io` (ici `10`) pendant une durée de `duration` (ici `2000`). Les clients invoquent un serveur `serverName` (ici `serv0`) depuis `start` (ici `0` pour la première invocation et `1000` pour la deuxième) pendant une durée de `duré` (ici `1000` pour les deux) avec la probabilité de `garde` (respectivement `0.1` et `1`) en utilisant une tirage aléatoire. Chaque invocation est composée de `volume` octets (respectivement `10` et `100000`).

Un serveur fait le calcul avec les mesures `nbLoop` (ici `5`) lorsqu'il reçoit une invocation. Nous pouvons également exprimer les contraintes pour le serveur et le client/serveur. `nbConstrainte` est le nombre de contraintes (ici `0`). Les clients ne déclarent pas de contraintes car ils ne sont pas déplacés.

Avec ce simulateur, nous pouvons imiter facilement les différents types d'applications afin d'analyser les comportements du mécanisme d'allocation de ressources avec un jeu de paramètres. L'utilisation d'un temps pendant lequel les objets réalisent leurs invocations permet de mieux les synchroniser entre eux et donc d'engendrer des comportements permettant de tester les capacités de notre maquette.

8.3.3 Le visualisateur

Un visualisateur a été développé en Java afin d'analyser les résultats plus facilement et de visualiser les migrations réalisées. Il est composé de deux applications : un programme permettant de localiser et de suivre des objets dans un environnement monodomaine ou multidomaine, et un logiciel élaborant une analyse statistique à l'aide d'histogrammes et de courbes.

Le visualisateur utilise les fichiers résultats générés par l'objet `Event` pour voir le placement des composants des applications ainsi que les dépendances entre objets. Les outils statistiques développés permettent la visualisation sous forme d'histogrammes et de courbes des résultats obtenus lors des simulations. Ainsi les graphiques de résultats dans ce chapitre (les figures **??**, **??**, **??**, **??**) sont générés par ces outils.

Avec ce visualisateur, nous pouvons vérifier que la décision de migration basée sur le modéle relationnel et l'analyse multicritère est efficace. Les fichiers résultats générés par le simulateur d'applications ne sont pas simples à interpréter et le fait de les visualiser permet une meilleure analyse des résultats qu'ils contiennent. Nous passons ainsi de résultats bruts à des résultats qui sont plus simples à étudier.

8.4 Tests interdomaines

La mesure quantitative d'un service qualitatif présente peu d'intérêt, puisque les résultats dépendent de facteurs non maîtrisables comme le trafic des réseaux à longue distance par exemple. Nous avons donc illustré l'utilisation du service de placement interdomaine par une expérience qui a été effectivement réalisée sur un ensemble de 8 machines. La moitié de ces machines, le domaine 1, est composée de PC et l'autre, les domaines 2 et 3, de machines SUN.

FIGURE 8.2 – Évolution d'une application dans un environnement multidomaine (avant)

Dans la figure **??**, nous présentons l'exécution d'une application sur trois domaines différents. Ici, la notion de domaine est associée à celle d'un réseau local, environnement que nous considérons comme homogène du point de vue de la sécurité. A chaque domaine est donc associé un niveau de sécurité avec :

- S3 équivaut à un niveau de sécurité minimum, par exemple avec une protection entre les applications.
- S2 équivaut à un bon niveau, par exemple permettant d'authentifier un utilisateur avant de lui donner la clef d'accès à un serveur.

– S1 équivaut à un niveau de sécurité très stricte supportant une procédure d'authentification à chaque accès aux serveurs, le cryptage des requêtes, etc.

Dans la figure **??**, le serveur s'exécute dans le domaine 2 alors que la plupart de ses clients s'exécutent dans les domaines 1 et 3. Cette situation consomme beaucoup de bande passante sur les réseaux d'interconnexion des domaines. Ces réseaux étant généralement moins performants que les réseaux locaux, cette situation entraîne un ralentissement de l'application et une moins bonne qualité de service de la part du serveur pour les clients.

FIGURE 8.3 – Évolution d'une application dans un environnement multidomaine (après)

Une fois le problème détecté, le module de décision analyse la situation pour mieux placer le serveur (voir la figure **??**). Celui-ci, ayant définit comme propriété qu'il avait besoin d'au moins un niveau de sécurité S2, n'est pas placé dans le domaine 3 où il a le plus de clients mais dans le domaine 1. Ce qui met en évidence le gain apporté par l'introduction des besoins applicatifs au niveau des domaines.

Ainsi, les principaux résultats de test sur les domaines concernent cette utilisation des besoins applicatifs. Un second résultat, qui concerne moins nos recherches, est d'avoir étendu le service de migration d'objet sur lequel nous nous reposons pour permettre la migration entre sites hétérogènes. Nous n'avons pas, ici, mis en évidence de gain dans les communications par manque de moyen. En effet, les domaines étant simulés sur un réseau local (nous n'avons donc utilisé que la définition logique des domaines), les gains de communication sont les mêmes que ceux que nous présentons plus loin. Pour finir, nous avons proposé une structure permettant l'intégration d'une politique de sécurité au sein de la gestion d'objets. Chronologiquement, cette réalisation est antérieure à celle du serveur ORM. Elle nous a donc aussi permis de mieux évaluer les potentialités des besoins applicatifs et nous a poussé à les introduire au niveau des sites.

8.5 Tests intradomaines

Les tests intradomaines cherchent principalement à mettre en évidence un gain de performance. Pour cela, nous avons défini un protocole de test permet d'évaluer autant que possible les différents paramètres du système. La sélection des paramètres et des expériences, pour être rigoureuse, doit se faire suivant des procédures établies. Nous présentons ces procédures puis la sélection des expériences et nous finissons par les résultats des tests.

8.5.1 Le choix des expériences

Le choix des expériences et l'ordre dans lequel elles doivent être faites est particulièrement important pour garantir la validité des résultats expérimentaux. C'est sur cette validité que repose les conclusions d'une étude. Nous ne proposons pas une analyse comparée des différentes méthodes de choix d'expériences. Ce travail est fait dans de nombreux ouvrages [**?**, **?**, **?**, **?**]. Ce problème a également déjà été étudié dans le cadre de l'allocation de ressources [**?**, **?**]. Dans la suite, nous nous limitons au rappel des étapes principales de l'élaboration d'un ensemble minimal d'expériences à réaliser et nous présentons la méthode que nous avons retenue.

Avant de poursuivre, il est nécessaire de préciser quelques points de vocabulaire. Ce que jusqu'à maintenant nous avons nommé paramètre, l'expérimentateur le nomme facteur. Les différentes valeurs, données à un paramètre durant une suite d'expériences, sont appelées les niveaux d'un facteur. Finalement, l'ensemble des expériences combinant les différents niveaux des différents facteurs est appelé un plan d'expériences. Nous utilisons ces termes par la suite.

L'approche expérimentale se décompose en plusieurs étapes. La première, le criblage, consiste à identifier les facteurs sur lesquels il est possible d'agir, et, parmi ces facteurs, déterminer lesquels sont importants. La deuxième étape, l'échantillonnage, doit, en fonction des facteurs sélectionnés, définir pour chacun d'eux un ensemble de niveaux (valeurs) pour lesquels des expériences doivent être faites. La troisième étape a pour objectif de définir un plan d'expériences en fonction des facteurs sélectionnés et de leurs niveaux. Une fois les expériences réalisées, la quatrième étape consiste à analyser les résultats, et si cela est nécessaire, elle déclenche l'itération du processus. Le processus peut être recommencé, soit depuis l'étape d'échantillonnage pour affiner un résultat, soit depuis le criblage pour introduire un nouveau facteur.

Considérons un nombre f de facteurs et leurs nombres de niveaux respectifs n_i. Pour réaliser le plan factoriel complet, c'est-à-dire réaliser l'ensemble des expériences définies par une combinaison quelconque des niveaux de chacun des facteurs, il faut faire $\prod_{i=1}^{f} n_i$ expériences. Pour diminuer la quantité d'expériences à réaliser, il faut utiliser des plans d'expériences fractionnaires. Les plans fractionnaires, bâtis sur la base des plans complets, consistent à ne retenir que les expériences maximisant les variations de niveau pour chaque facteur. Autrement dit, à chaque expérience d'un plan fractionnaire plusieurs facteurs changent de niveau par rapport à l'expérience précédente. Parmi les multiples méthodes de construction de plan fractionnaire, nous avons choisi d'utiliser la proposition faite dans [**?**]. Cette méthode, appelée méthode Taguchi, du nom de son inventeur, est basée sur l'utilisation de tables prédéfinies pour un nombre de facteurs et de niveaux. Le tableau **??** présente une table permettant de réaliser un plan de 12 expériences pour étudier l'impact de 11 facteurs ayant chacun deux niveaux. En utilisant plusieurs colonnes pour un même facteur, il est possible, de multiplier le nombre de niveaux en diminuant le nombre de facteurs. Les tables définies par Taguchi ont l'avantage de réaliser une bonne répartition des expériences dans l'espace d'expérimentation.

Dans la suite de ce chapitre nous détaillons l'utilisation de la méthode Taguchi pour l'étude expérimentale du gestionnaire d'objets que nous avons défini.

La sélection des facteurs

Nous utilisons différentes valeurs pour les facteurs et exécutons différents scénarios. Nous essayons de montrer l'impact des facteurs sur les performances. Pour simplifier la réalisation

facteur	1	2	3	4	5	6	7	8	9	10	11
expérience											
1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	2
3	1	1	2	2	2	1	1	1	2	2	2
4	1	2	1	2	2	1	2	2	1	1	2
5	1	2	2	1	2	2	1	2	1	2	1
6	1	2	2	2	1	2	2	1	2	1	1
7	2	1	2	2	1	1	2	2	1	2	1
8	2	1	2	1	2	2	2	1	1	1	2
9	2	1	1	2	2	2	1	2	2	1	1
10	2	2	2	1	1	1	1	2	2	1	2
11	2	2	1	2	1	2	1	1	1	2	2
12	2	2	1	1	2	1	2	1	2	2	1

TABLE 8.1 – Un plan de 12 expériences

du test, nous avons choisi d'utiliser la répartition présentée dans le tableau **??**. Avec six facteurs, nous avons la possibilité d'utiliser quatre niveaux pour chacun d'eux, sauf pour celui qui doit être limité à deux niveaux. Nous présentons ci-dessous les niveaux que nous utilisons.

Pour chacun des facteurs le choix des niveaux est réalisé de façon à répartir équitablement les expériences sur une gamme intéressante de valeurs, du point de vue de la plate-forme et du type général d'application.

1. la base de temps, périodicité du serveur POM, détermine en grande partie la sensibilité du système. Elle détermine également la quantité de ressources consommées par ce même système. Nous utilisons quatre niveaux exprimés en secondes : 20, 40, 80, 100.
2. la durée d'amortissement, comme la base de temps, détermine la sensibilité du système. Nous l'exprimons sous la forme d'un multiple de la base de temps. Nous utilisons quatre niveaux : 1, 2, 5, 20.
3. le rapport des coefficients d'importance décrit l'importance relative de la charge face aux communications. C'est un nombre réel pour lequel nous définissons quatre niveaux : 1, 1.5, 2, 3. Ceci est équivalent aux couples (k_{charge},$k_{relation}$) suivants : (0, 0.7), (0.23, 0.47), (0.35, 0.35), (0.47, 0.23). Le premier niveau consiste à privilégier les communications vis-à-vis de la charge. Théoriquement, cette approche ne doit pas donner de bon résultat. Le coefficient d'importance pour les besoins d'objet est toujours 0.3 pour l'instant.
4. le veto sur la charge indique quelle amélioration minimum sur la charge doit garantir un déplacement. Nous utilisons quatre niveaux, exprimés en nombre de threads : -30, -15, 0, 5. Les niveaux en valeur négative signifient qu'un déplacement peut mener à une situation moins bonne en terme de charge.
5. le veto sur les communications indique quelle amélioration minimum sur les communications doit garantir un déplacement. Dans notre cas nous posons plus un veto pour empêcher les grosses pertes que pour garantir un gain. Nous utilisons quatre niveaux, exprimés en octets par seconde : -1024, -128, -16, +16. Le niveau +16 implique qu'un déplacement doit obligatoirement améliorer les communications pour être accepté, cela

est antinomique avec la gestion de charge. Les valeurs données ci-dessus sont exprimées en volume de données transmis sur la base du critère non normalisé.

6. le seuil d'indifférence définit la valeur minimum du critère de synthèse pour admettre un déplacement. Cette valeur traduit le coût de l'opération de déplacement. Nous utilisons deux niveaux exprimés dans l'unité du critère de synthèse : $0.01Borne$, $0.1Borne$.

facteur / expérience	B_{temps}	D_{amort}	R_k	V_{charge}	$V_{relation}$	S_{indif}
1	20	1	1.5	-30	-30	0.01
2	20	1	2	5	5	0.1
3	20	20	3	-30	5	0.1
4	40	2	3	5	-30	0.1
5	40	5	1	-15	-15	0.01
6	40	20	2	0	0	0.01
7	80	20	1.5	5	-15	0.01
8	80	5	1	0	-30	0.1
9	80	2	1	-15	0	0.01
10	100	5	1.5	-15	0	0.1
11	100	2	2	-30	-15	0.1
12	100	1	3	0	5	0.01

TABLE 8.2 – Premier plan d'expériences

Nous obtenons, en introduisant ces valeurs dans la table de Taguchi, le plan d'expérience présenté dans le tableau **??**. Nous disposons de 12 expériences à réaliser pour différents types d'applications ou de groupes d'applications. L'analyse de la réponse du système à ces expériences permet de bâtir une prévision de la réponse théorique du système en fonction des valeurs d'un groupe de facteurs. Si les réponses théoriques prévues et des mesures effectives, pour les mêmes valeurs de facteur, donnent des résultats équivalents, le modèle de prévision est correct. Dans le cas contraire, il est nécessaire de bâtir un nouveau plan de test. Ce dernier doit, soit contenir de nouveaux facteurs, soit explorer plus précisément une gamme de valeurs pour un ou plusieurs facteurs.

Nous présentons, ci-dessous, dans la partie consacrée aux performances, des résultats expérimentaux obtenus grâce au plan défini par le tableau **??**.

8.5.2 Les performances

Cette partie présente les expériences que nous avons réalisées et les premières conclusions que nous en déduisons. Le nombre d'expériences à réaliser, même en utilisant la méthode Taguchi, est particulièrement grand. De plus, les résultats dépendent du contexte des applications que nous ne pouvons pas généraliser. Nous proposons, par la suite, les résultats des tests que nous avons réalisés pour obtenir une première validation de nos travaux. Nous avons mené ces tests sur des types d'applications spécifiques, de façon à mettre en évidence les qualités du système que nous proposons. Nous disposons également de résultats quantitatifs faisant apparaître dans quel cadre notre solution est efficace et dans quel cadre elle ne l'est pas.

Nous étudions ci-dessous le comportement de cinq applications. La première montre, sur un cas précis, comment le placement, en tenant compte des communications, diffère du placement dirigé par la charge seule. La deuxième application montre que le système optimise les communications pour deux serveurs mal placés. La troisième montre comment le système déplace des objets regroupés dans un même site, en tenant compte du critère de charge et du critère de communication. La quatrième montre l'influence sur les performances du critère de préférences d'objets. Finalement, nous présentons les résultats du mécanisme lorsqu'il gère un ensemble important d'applications lancées concurremment.

8.5.3 L'optimisation des communications

La première expérience que nous avons réalisée concerne le gain potentiel lors de l'optimisation des communications. Nous avons réalisé ces tests pour voir la différence de performances entre la communication locale et distante dans le système sans gestionnaire et avec notre gestionnaire. Pour cela, nous avons exécuté un client et un serveur en local puis en distant pendant une durée de 2000 secondes et mesuré le nombre d'invocations réalisées durant cette durée.

Configuration	GO		Sans GO	
	Local	Distant	Local	Distant
CPU 1, V 10	22350	21891	25574	25333
CPU 1, V 1000	21974	20250	25267	23186
CPU 1, V 100000	8148	2745	9465	2972
CPU 10, V 10	3041	3025	3161	3160
CPU 10, V 1000	3036	2990	3155	3122
CPU 10, V 100000	2448	1538	2618	1625
CPU 20, V 10	1554	1552	1609	1604
CPU 20, V 1000	1552	1541	1608	1602
CPU 20, V 100000	1379	1035	1449	1082

TABLE 8.3 – Résultats de la différence entre GO et sans GO.

Le tableau **??** montre les résultats obtenus pour les invocations locales et distantes avec le système GO et avec un système sans GO. Lorsque la consommation de CPU par l'objet client augmente, la différence de performances entre la machine locale et la machine distante diminue puisque le temps passé en communication est moindre. A l'opposé, si le volume de communication (V) entre le client et le serveur augmente, la différence devient plus grande. Si la consommation de CPU est de 1 et le volume de communication est de 100000, le surcoût de notre système est d'environ 14%. Ceci est dû notamment à l'observation du COMOBS, car chaque invocation au serveur est enregistrée par le COMOBS. Si le rapport entre CPU et communication augmente alors le surcoût diminue (environnement 3% pour le CPU 20, et le V 10). Nous pouvons donc conclure que le surcoût de notre système est principalement lié à la saisie d'informations puisque, dans les deux cas, la procédure décision a le même coût. Une solution pour diminuer le surcoût est donc de faire une mise à jour périodique des informations. Cependant, les informations collectées deviennent moins précises.

Nous pouvons déduire des expériences précédentes le gain du placement pour le scénario

CPU 1 et V 100 000. Le nombre d'invocations distantes sans GO est de 2972 et le nombre d'invocations locales avec GO est de 8148. Théoriquement, si le gestionnaire de placement est capable de positionner le serveur près de son client et donc lui permettre de réaliser un plus grand nombre d'invocations, nous pouvons avoir une amélioration maximale de 174%, hors coût de migration. Ceci montre que l'intérêt du placement du point de vue des communications est énorme, même avec le surcoût de notre système.

8.5.4 Expérience sur l'optimisation des communications

Ce premier test a pour objet de montrer une différence caractéristique entre un placement dirigé uniquement par la charge et un placement dirigé par la charge et par les communications. Le déplacement d'une entité entre deux sites de charge équivalente permet d'obtenir une optimisation du temps d'exécution.

FIGURE 8.4 – Mobilité du serveur en fonction des communications

La figure **??** schématise une application composée de deux clients et d'un serveur. Les clients sont placés sur des sites distincts, et invoquent concurremment le serveur. A l'origine le serveur est placé sur le site du premier client (C0). L'expérience est découpée en périodes (2000 secondes). Durant une période, l'un des clients invoque faiblement le serveur (un volume de 10 octets et une probabilité de $0,1$) alors que l'autre client (C1) l'invoque fortement (un volume de 100000 octets et une probabilité de 1). Puis, durant la période suivante, la tendance est inversée, c'est le premier client qui échange de plus gros volumes avec le serveur. L'indicateur du serveur pour la consommation du temps CPU est fixé à 1. Dans cet exemple, les clients sont non-migrables, seul le serveur est pris en charge par le mécanisme de gestion des ressources. Le but est alors qu'il se déplace entre les deux pour optimiser les communications.

Nous cherchons alors à vérifier que le déplacement du serveur peut amener une optimisation du temps d'exécution. Notons que cette optimisation ne peut être obtenue que par limitation des communications distantes car, dans le cas d'une allocation de ressources dirigée uniquement par la charge, la différence de charge (1 pou 2) est insuffisante et le serveur n'est jamais déplacé. Par contre, si l'allocation de ressources intègre les communications et si les volumes de données échangées le justifient, le serveur passe successivement d'un site à l'autre. Il est bien évident que cette solution n'est efficace que si la migration a un coût faible devant le coût des communications échangées. Ces tests sont réalisés uniquement sur deux machines.

Dans le graphique **??**, nous présentons les résultats de l'exécution de l'application décrite ci-dessus en fonction des différentes configurations du mécanisme d'allocation. Nous présentons également le résultat de l'exécution sans gestion de ressources.

Pour les expériences 2, 4, 7, 8 et 12, aucune migration n'est réalisée et les résultats sont moins

FIGURE 8.5 – Résultats de l'application 1S2C

bons avec la gestion de ressources. Ceci est dû au fait que l'exécution de l'application testée ne peut être améliorée que du point de vue des communications et au surcoût de notre système qui est en moyenne de 6 %. Dans ces cas, le temps pris par la recherche de l'optimisation est du temps perdu, il est responsable de l'allongement des durées d'exécution. Les expériences 2, 4 et 7 posent des veto importants sur la charge et ainsi ne permettent pas la migration du serveur. Nous constatons, dnas ce cas, que le temps utilisé par le système n'influence pas le surcoût de notre système.

Pour les expériences 1, 5, 9 et 11, les paramètres utilisés permettent une amélioration des performances. Dans ces expériences, les valeurs de veto pour la charge sont toutes négatives. Ceci n'empêche pas donc un placement qui entraîne une dégradation de charge. L'expérience 1 donne une augmentation moyenne des performances de l'ordre de 16%. Les paramètres utilisés dans l'expérience 1 sont plus sensibles à leur environnement avec une base de temps courte, une durée d'amortissement courte, des veto négatifs et un seuil d'indifférence bas. Dans ce cas, l'expérience 1 donne la meilleure performance car le système détecte le changement et réalise une migration rapide pour avoir un gain plus important. Par contre, l'expérience 6 donne le moins bon résultat. Ceci est dû au fait que la durée d'amortissement utilisée est la plus longue et que le placement est réalisé trop tard, car la migration du serveur a lieu en fin de période. Cette migration devient alors un mauvais placement. Un mauvais placement peut causer une baisse des performances, dans ce cas, la perte est de 31%.

Globalement, par ce test, nous montrons que sur une application très simple, en fonction du jeu de paramètres utilisé, nous obtenons des résultats allant de -31% à +16% du temps d'exécution sans optimisation de l'allocation des ressources. Pour le meilleur paramétrage, la diminution de 16% du temps d'exécution est obtenue uniquement en optimisant les communications. Pour ce test, nous déduisons qu'il est propice d'utiliser des valeurs négatives pour les veto et d'avoir une durée d'amortissement sensiblement courte. Finalement, la base de temps est négligeable pour le surcoût de notre système mais la durée d'amortissement est elle importante.

Le gain est aussi dépendant du grain d'invocation des applications. Si le volume d'invocations augmente par rapport à la consommation du temps CPU pour cette invocation (100.000/1), le gain s'accroît. Nous allons montrer cette relation dans le paragraphe suivant.

8.5.5 Expérience sur les facteurs multicritères

Ce troisième test a pour objet de montrer un placement optimisant les communications par rapport à la charge. Ce système permet de passer par des phases sous-optimales pour atteindre une situation optimale. Ici, la procédure de décision multicritère prend toute son importance car c'est elle qui rend le déplacement possible.

L'application présentée sur le schéma **??** est constituée de deux clients et de deux serveurs. Un client et un serveur sont placés sur chaque site. Chacun des clients invoque les deux serveurs. Le premier client invoque le premier serveur (local) avec un volume de données très faible (10 octets) mais invoque le deuxième serveur (distant) avec un volume de données très important (100 000 octets). De la même manière, le deuxième client invoque le deuxième serveur (local) avec peu de données mais fortement le premier serveur (distant). Dans ce cas, le système n'est pas optimisé du point de vue des communications et nécessite le déplacement des deux serveurs mais, pour y arriver, il est nécessaire de compenser le critère de charge par un critère de communication important.

FIGURE 8.6 – Mobilité du serveur en fonction des communications

Dans le graphique **??**, nous obtenons un bon résultat par rapport au test sans GO sauf pour les expériences 2, 4 et 7 qui posent des veto importants sur la charge et n'autorisent pas la migration du serveur. Le surcoût dans ce cas est d'environ 2%. L'expérience 1, qui utilise les paramètres les plus sensibles à son environnement donne le meilleur résultat soit une amélioration de 51%. Dans ce scénario de test, il n'existe pas de mauvais placements ce qui fait que toutes les expériences ayant réalisé une migration permettent une amélioration des performances.

L'expérience 6, qui donnait les plus mauvais résultats précédemment, donne ici des résultats acceptable. Nous pouvons donc imaginer que le jeu de paramètres optimum est dépendant de l'application. Cette hypothèse se confirme par la suite.

8.5.6 Expérience sur la charge et les communications

Ce quatrième test nous permet de montrer que notre service de placement tient compte en même temps de la charge et des communications qui sont deux critères contradictoires. Ce test est constitué de deux serveurs, un client/serveur, et huit clients qui sont lancés sur le même site (voir la figure **??**) et dont les communications sont réparties entre les deux serveurs. Si tous les objets se trouvent sur le même site, nous pouvons optimiser les communications et non la charge. Inversement, si les serveurs sont distribués, nous pouvons optimiser la charge et non les communications.

FIGURE 8.7 – Résultats de l'application 2S2C

FIGURE 8.8 – Mobilité du serveur en fonction des charges

Avec un bon choix de paramètres, notre système est capable de trouver un équilibre entre les deux critères. Les résultats obtenus dans la figure **??** montrent que seules les expériences 6 et 8 donnent de moins bons résultats car elles n'ont pas réalisé de migrations. Pour les autres expériences, les gains varient entre 250 et 350%, ils sont donc très importants.

8.5.7 Expérience sur la validité des paramètres

Par la suite, nous avons réalisé une expérience similaire, en modifiant uniquement les paramètres de l'application de test. L'idée est de vérifier que, sans changer les paramètres du service de gestion des objets, les résultats sont valides pour plus d'un type d'application.

Dans cette expérience, les résultats obtenus (figure **??**) sont négatifs pour la plupart des plans d'expérience : seules les exécutions 1 et 11 donnent des résultats positifs, avec un gain de l'ordre de 25%. Nous pouvons conclure que les paramètres qui sont très sensibles à leur environnement ne donnent pas toujours de manière absolue la meilleure performance car ils peuvent facilement induire une instabilité du serveur. Leur choix est donc un élément primordial de la qualité du service de placement et il est dépendant de l'application.

FIGURE 8.9 – Résultats de l'application 2S1I8C

FIGURE 8.10 – Résultats de l'application 2S1I8C

8.5.8 Expérience sur la coallocation

Ce cinquième test nous permet de montrer une amélioration des performances en tenant compte d'un critère supplémentaire tel que la coallocation des applications. Nous avons lancé 2 clients sur chaque site de notre plate-forme de 4 machines. Les deux serveurs sont invoqués également par tous les clients. Chaque invocation du serveur consomme 5 mesures de CPU. Nous spécifions la préférence du premier serveur et du deuxième serveur avec une coallocation négative 5−, c'est-à-dire que les deux serveurs préfèrent être séparés. Ces deux serveurs sont lancés sur le même site, avec une préférence de coallocation négative, les deux serveurs sont séparés avec le placement.

Cette expérience n'a été conduite qu'avec un seul jeu de paramètre. Nous avons une moyenne d'invocations de $65,75$ pour les deux serveurs séparés sur des sites différents. La moyenne d'invocations pour les deux serveurs sur le même site est de $33,25$. Nous obtenons donc une

FIGURE 8.11 – Mobilité du serveur en fonction des charges et les besoins d'applcations

amélioration de 98% par l'influence supplémentaire des préférences de coallocation du serveurs.

8.5.9 Modification du comportement de l'application

Le dernier test présente un scénario plus complexe et plus général qui inclut un changement important et durable du comportement de l'application. Le but est de valider la capacité de notre service à prendre en compte un changement de ce type. Les objets sont distribués, interagissent entre eux et s'exécutent concurremment. Cette application est constituée de 34 objets s'exécutant simultanément sur un ensemble de 4 machines : 11 serveurs, 3 clients/serveurs et 20 clients. Dans ce scénario, les objets (serveurs, clients, clients/serveurs) sont distribués au hasard, ce qui fait qu'ils ne sont pas dans une position optimisée selon les critères tels que la charge et la relation. Certains clients invoquent les serveurs de manière uniforme pendant 4000 secondes, la durée de la simulation. Cependant, au bout de 1500 secondes, une part importante des objets change son comportement pour la consultation des serveurs. Ils changent, par exemple, de serveurs invoqués ou le volume des communnications.

Dans la figure **??** et le tableau **??** nous présentons les résultats des 12 tests pour l'application décrite ci-dessus.

Les valeurs reportées dans ce tableau sont les moyennes d'invocations des clients, le nombre de migrations du serveur, le rapport avec la durée d'exécution sans équilibrage de charge et le pourcentage d'amélioration.

Dans le cadre de ce test, les résultats mettent en évidence trois types de comportements distincts en fonction des jeux de paramètres. Le premier comportement, visible pour les jeux de paramètres 1, 5 et 11 est caractérisé par une augmentation importante des performances allant jusqu'à 137% du temps d'exécution. Dans ces expériences, les paramètres de veto sont tous bas et, par conséquent, la migration est privilégiée. Le deuxième comportement se caractérise par un impact faible (-25%) de l'allocation de ressources pour les tests 2, 4, 7 et 12 où aucun placement n'est réalisé. Le troisième comportement montre typiquement que la base de temps utilisée n'influence pas le surcoût de notre système.

La conclusion que nous pouvons tirer de cette expérience est que notre système est capable d'améliorer radicalement les performances dans les systèmes complexes et déséquilibrés. La plupart des jeux utilisés apportent un gain suffisant pour justifier l'utilisation du service.

FIGURE 8.12 – Résultats de l'application 11S2I20C

Cependant, étant donné le surcoût important (25%) engendré par un jeu de paramètres mal adapté, il est crucial de savoir choisir ce jeu en fonction de l'application exécutée.

8.5.10 Analyse des paramètres

Nos expériences de test sont empiriques. Les résultats dépendent non seulement du matériel tel que les processeurs et les réseaux, mais aussi des différents types d'applications. Malgré la difficulté pour généraliser les résultats, nous essayons d'analyser les paramètres, particulièrement pour le dernier test qui est plus complexe et général. Avec la méthode Taguchi, nous pouvons faire le calcul des interactions entre les facteurs. Nous montrons ici l'effet moyen d'un facteur de veto sur les critères de charge et de relation.

Les effets moyens de veto de charge sur les quatre niveaux de valeur sont :

$$E_{(Vcharge=-30)} = (\frac{1282 + 870 + 862}{3}) - 706,17 = 298,50$$

$$E_{(Vcharge=-15)} = 208,50$$

$$E_{(Vcharge=0)} = -209,17$$

$$E_{(Vcharge=5)} = -297,84$$

Nous constatons que la valeur des veto la plus petite a un effet plus positif sur le résultat. Les effets moyens des veto de relation dans les quatre niveaux de valeur sont :

$$E_{(Vrela=-30)} = 16,16$$

$$E_{(Vrela=-15)} = 119,16$$

$$E_{(Vrela=0)} = 8,5$$

$$E_{(Vrela=5)} = -143,84$$

Expérience	Invocation	Migration	Accélération	Amélioration (%)
Sans gestion	542	0	1	0
1	1282	80	2,37	137
2	408	0	0,75	-25
3	870	1	1,61	61
4	408	0	0,75	-25
5	1205	25	2,22	122
6	805	9	1,49	49
7	409	0	0,75	-25
8	477	4	0,88	-12
9	799	15	1,47	47
10	740	12	1,37	37
11	862	41	1,59	59
12	409	0	0,75	-25

TABLE 8.4 – Résultats de l'application 11S3I20C

Les valeurs des veto de relation sont moins évidents, mais nous pouvons conclure que la valeur positive (5) a un effet négatif sur le résultat.

L'interaction entre veto sur la charge et sur la relation aux niveaux respectifs -30 et -30 est la suivante :

$$I_{(Vcharge=-30,Vrela=-30)} = \frac{(Y_1)}{1} - M - E_{(Vcharge=-30)} - E_{(Vrela=-30)}$$
$$I_{(Vcharge=-30,Vrela=-30)} = 1282 - 706,17 - 298,50 - 16,16 = 261,17$$

Il faut réaliser beaucoup de plans d'expériences pour avoir une réponse théorique qui donne une approximation de notre système. Nous pourrions effectuer des mesures complémentaires pour, d'une part valider le modèle, et d'autre part affiner les approximations en effectuant plus de mesures à proximité des points optimums. Etant donné le grand nombre de combinaison des facteurs qui nécessite le très grand nombre des plans d'expériences pour analyser des paramètres, nous pouvons avoir une solution qui est plus pratique dans ce contexte, en intégrant l'auto-adaptation que nous allons expliquer dans le chapitre suivant.

8.6 Conclusion

Nous avons réussi à montrer l'intérêt de faire un placement interdomaine et intradomaine qui tient compte de plusieurs critères simultanément. En effet, les gains apportés par le service de gestion de l'exécution des objets sont toujours positifs si le jeu de paramètres est adapté. Ces gains sont dépendants de la configuration de l'application, ce qui est évident puisque son placement initial est plus ou moins optimisé pour son exécution. Les gains obtenus en gérant dynamiquement ce placement peuvent alors atteindre plus de 300%.

D'autre part, nous avons constaté que les résultats sont dépendants des paramètres choisis et du contexte des applications et, étant donné le surcoût important (25% en moyenne) engendré

par un jeu de paramètres mal adapté, il est crucial de savoir choisir ce jeu en fonction de l'application exécutée. Plusieurs solutions sont envisageables :

- La première consiste à réaliser un nombre important d'expériences pour essayer de trouver un corrélation entre le type de l'application et le jeu de paramètres optimum. Cette solution est laborieuse car elle peut engendrer un nombre considérable d'expériences avant de donner des résultats probants. D'autre part, elle pose plusieurs problèmes : comment combiner les paramètres lorsque plusieurs applications, éventuellement contradictoires, s'exécutent et comment prendre en compte le dynamisme des appplications à base de composants.
- La seconde solution consiste à prendre un jeu de paramètres moyen qui soit adapté à la plupart des applications. C'est, sur les expériences que nous avons réalisé, le cas du jeu 1 qui ne fournit pas toujours les meilleurs résultats mais donne toujours des résultats positifs. Les principaux défauts de cette solution sont son inadaptation probable à certaines applications ayant des comportement atypiques et le fait qu'il est possible de mieux faire en cherchant une solution adaptée au contexte.
- La dernière solution consiste à calculer dynamiquement le jeu de paramètres. Nous avons déjà effectué quelques recherches dans ce sens. Elles sont présentées dans [**?**] pour une proposition d'adaptation génétique de ces paramètres et dans [**?**] pour une proposition plus intuitive.

Avec la méthode Taguchi, nous avons essayé de montrer les corrélations entre plusieurs facteurs. Il est évident que cette méthode, dans un contexte expérimental tel que le notre, présente de nombreux intérêts. Cependant, les plans d'expériences, pour tester la combinaison d'un grand nombre de paramètres, sont très nombreux. Nous avons donc limité les niveaux de nos facteurs à des valeurs intéressantes. Or, il est difficile actuellement de généraliser ces paramètres ceci nécessiterait une longue série de tests.

Chapitre 9

Conclusion et perspectives

9.1 Conclusion

Durant les dix dernières années, les systèmes d'information ont subi une mutation complète de leur conception en passant d'une architecture centralisée autour d'un ordinateur principal - qui était le seul fournisseur de service - à une architecture distribuée intégrant le réseau comme son squelette, son support principal, et donnant accès à des services à l'échelle planétaire. Cette révolution de la structuration s'est évidemment accompagnée d'une révolution dans la réalisation et l'implantation des applications offertes. En effet le choix initial, centralisé, était justifié par la simplicité de mise en oeuvre. Le passage à un environnement distribué a complexifié la programmation mais, paradoxalement, lui a été bénéfique en l'obligeant à plus de modularité. Si, dans les premiers temps, les réalisations applicatives ont été un peu chaotiques, elles ont au moins permis d'apprendre à mieux maîtriser ces architectures nouvelles. Aujourd'hui, l'émergence de nouveaux outils plus adpatés, permet d'envisager le développement d'applications réparties avec plus de sérénité. Parmi les technologies candidates à la maîtrise des environnements distribués, la programmation et le support de composants sont universellement reconnus comme étant une solution adaptée. Pourquoi ? Parce que les composants permettent d'offrir des services à travers une interface claire ce qui les rend facilement réutilisables, parce que la modularité permet de mieux maîtriser le code, de le faire évoluer et de le déployer sur un grand nombre de plates-formes, parce que le modèle intègre le moyen d'intervenir en temps d'exécution permettant ainsi plus de souplesse dans l'utilisation, etc. Ainsi, à l'instar de la révolution architecturale des réseaux et de l'Internet, la révolution de la conception et du développement d'applications est en train d'avoir lieu.

Une des avancées majeures due au modèle de programmation par composant est la dynamicité de l'accès aux services. Cette dynamicité est la clef d'un nouveau mode de conception où la structure de l'application n'a qu'une validité temporaire. L'utilisateur ou un composant peut choisir l'entité qui lui rend le service et la changer lorsqu'il le désire. Cependant cette dynamicité pose des problèmes pour la gestion des composants. Si les liens entre les composants changent, la structuration intiale des composants, leur placement sur le réseau, peut devenir inadaptée. De même, en changeant dynamiquement le paramètrage d'un composant, l'utilisateur en modifie les besoins ce qui peut entraîner une inadéquation entre l'environnement d'exécution et le composant.

Dans les applications traditionnelles, qu'elles soient centralisées ou basées sur un schéma client/serveur, l'administrateur du système cherche à évaluer le taux d'utilisation de chacune des ressources (réseau ou ordinateur) de manière à offrir à l'utilisateur un confort maximum. Ceci peut être réalisé manuellement puisque le grain des applications est tel qu'il peut avoir la connaissance de chacune. Cependant, en passant à un modèle d'application basé sur les composants, la granularité des entités diminue ce qui rend le travail de l'administrateur beaucoup plus complexe puisque le nombre des entités et des liens augmente. La dynamicité du concept apporte également à la complexification du travail. Pour résoudre ce problème, nous avons proposé dans ce document un service d'administration automatique de l'exécution des composants. Ce service doit se substituer à l'administrateur pour toutes les tâches de positionnement des composants et leur optimisation.

Tel qu'il a été conçu, ce service s'intègre dans un environnement d'exécution de composant de type CORBA ou autre. Nous avons insisté sur sa conception modulaire et le découpage en fonctionnalités indépendantes. Ainsi, le service est conçu sur le même principe que les applications qu'il gère ce qui lui confère une grande facilité d'intégration tout en laissant aux applications le choix du niveau de service utilisé. La définition de ce service est principalement issue des expériences que nous avons acquises lors de la réalisation d'un service de gestion d'objets pour CORBA. Un de ses intérêts, par rapport à d'autres services ou d'expériences équivalents, est la prise en compte de plusieurs types d'informations et l'utilisation d'une procédure de décision basée sur l'analyse multicritère. Bien sûr d'autres types de procédures de décision peuvent être utilisées, y compris des procédures de type ELECTRE ou basées sur la théorie de l'algorithmique génétique. La mise en pratique de ce modèle, même si elle n'est que partielle, nous a permis de montrer la faisabilité et l'intérêt de ce modèle.

La contribution générale de notre travail se situe donc au niveau du support à la gestion automatique de l'exécution des composants. Nos recherches s'étendent de la modélisation de services à base de composants à la mise en pratique dans un environnement CORBA. Des domaines annexes ont été abordés à travers les procédures de décisions dans un environnement dynamique, l'étude des environnements répartis à objets et composants, la simulation d'applications réparties et la mise en oeuvre de plans d'expériences permettant de réduire le nombre de tests lors de l'évaluation d'une maquette.

La mise en oeuvre d'une plate-forme de tests pour les services en environnement distribué est un travail de longue haleine. Ainsi, entre le début de la réalisation de la première maquette et aujourd'hui, il s'est écoulé cinq années qui lui ont été presque entièrement consascrées. Dans le cadre de ce travail, j'ai encadré, sous la responsabilité du Professeur Hervé Guyennet, deux doctorants dont la thèse est finie et un dont la thèse est en cours. Trois étudiants en DEA ont également pris part au projet pour réaliser leur stage de recherche et deux binômes de DESS nous ont aidés dans le développement d'outils.

9.2 Perspectives

Dans l'état actuel, nous avons validité la faisabilité et l'intérêt de la mise en oeuvre d'une telle maquette. Ce travail a donc fourni les résultats excomptés. Néanmoins la tâche n'est pas terminée et elle me donne toujours l'impression de n'être qu'au commencement. Dans cette partie, nous présentons les perspectives que nous envisageons.

Certains aspects de l'administration de l'exécution de composants n'ont cependant pas été traités mais le mériteraient : utilisation de différentes procédures de décision dans un même domaine mais sur des noeuds différents, faire cohabiter plusieurs modules de décision par exemple sur la migration et sur la duplication, la définition et l'utilisation d'un moteur de règles simple permettant d'étendre la sémantique associée aux besoins applicatifs, etc. Ces aspects deviendront critiques dans les environnements de composants dès que nous voudrons les gérer de manière automatique. La maquette actuelle, comme nous l'avons dit, est implantée au dessus de COOL-ORB qui n'est plus supporté depuis le rachat de la société Chorus par Sun. Le portage sur un autre ORB ne serait pas très coûteux et nous avons déjà évalué le travail nécessaire pour OrbAcus. Cependant, ce travail n'apporterait pas grand chose de plus que la maquette existante. Pour toutes ces raisons, nous avons décidé de réaliser son portage sous un environnement Java. Ceci nous permettra d'évaluer son adaptabilité et de gérer un plus grand nombre de composants.

Nos travaux nous ont permis de bien connaître les environnements de composants. Or, on ne travaille pas dans ce domaine sans imaginer les possibilités qu'il offre pour des domaines d'application très variés. Ainsi, nous avons fait état de travaux dans le domaine du parallélisme. Nous nous intéressons plus particulièrement aux problèmes de médiation (trading). En effet, la dynamicité d'invocation entre les composants évoquée précédemment suppose, pour pouvoir être mise en oeuvre de manière efficace, d'avoir recours à des services spécifiques appelés médiateurs. Le rôle d'un médiateur est alors de sélectionner, en fonction de besoins exprimés par le client, le composant le plus adapté au traitement de sa requête. Le médiateur peut également prendre, pour le client, le rôle de courtier de requête. Un étudiant de DEA réalise actuellement son stage sur ce sujet. De plus, nous avons, dans ce domaine, entrepris une collaboration avec la société StellarX qui commercialise des solutions de commerce électronique.

Chapitre 10

Liste des publications personnelles

10.1 Chapitre dans un livre

1. Placement dynamique et répartition de charge : application aux systèmes répartis et parallèles
   co-auteur avec Hervé Guyennet
   Edité par G.Bernard, J. Chassin de Kergommeaux, B. Folliot et C. Roucairol
   Collection didactique INRIA, décembre 1996

10.2 Communication dans une revue à audience internationale

2. "Evaluation of a multicriteria method to optimize resource access in distributed object systems"
   Huah Yong Chan et Laurent Philippe,
   a paraître dans Parallel and Distributed Computing Practices, special issues on Distributed Object Systems,
   Nova Sciences Editor
3. "A comparison study of dynamic load balancing algorithms"
   Hervé Guyennet, Bénédicte Herrmann, François Spies et Laurent Philippe
   International Journal of Mini and Microcomputers, Vol 19, Nb 3, 1997, ISMM Editor
4. "Dynamic Object Positionning"
   François Bourdon, Pascal Chatonnay, Bénédicte Herrmann Laurent Philippe
   Special Issues in Object-Oriented Programming. ECOOP'96. Max Mulhauser
   Ed. Dpunkt Verlag, p 374. 1996. Livre collectif réalisé à partir des articles de la conférence ECOOP'96.

10.3 Articles dans une revue à audience nationale

5. "Mise en oeuvre d'un algorithme parallèle sous CORBA et PVM"
   Jean-Luc Anthoine, Pascal Chatonnay, David Layimani, Jean-Marc Nicod et Laurent Philippe

a paraître dans Techniques et Sciences Informatiques, numéro spécial Parallélisme, distribution et objets, Editions Hermes.

6. "Placement dynamique dans les systèmes répartis à objets"
Pascal Chatonnay, Bénédicte Herrmann,Laurent Philippe, François Bourdon, Pascal Bar, Christian Jacquemot
Calculateurs Parallèles, Vol. 8, Nb. 1, 1996, Editions Hermès.

7. "Migration de processus dans Chorus/MiX"
Laurent Philippe et Guy-René Perrin
La Revue Electronique sur les Réseaux et l'Informatique Répartie (EJNDP), Nb. 1, avril 1995.

10.4 Cours dans une école à audience nationale

8. "Placement dynamique et répartition de charge, application aux systèmes répartis et parallèles", cours donné en collaboration avec H. Guyennet à l'école sur le placement et la répartition de charge, Presqu'ile de Gien, juillet 96

9. "Répartition de charge dans un applicatif Objet/Corba", cours donné à l´école du CNRS ICARE'97 sur la conception et la mise en oeuvre d'applications parallèles irrégulières de grande taille, Aussois, décembre 1997.

10.5 Communications dans un colloque à audience internationale

10. "Specification of a Scilab Meta-Computing extension",
Frédéric Lombard, Sylvain Contassot, Jean-Marc Nicod et Laurent Philippe
Conférence Internationale ICPP, Editée par IEEE, Toronto, Canada, Septembre 2000

11. "Auto-adaptive administration of resource allocation".
Huah Yong Chan, Pascal Chatonnay, Bénédicte Herrmann et Laurent Philippe
Conférence Internationale, ERSADS'99, Madeira Portugal, avril 99

12. "Parallel Numerical computation using CORBA"
Pascal Chatonnay, David Laiymani, Jean-Marc Nicod et Laurent Philippe
Parallel and Distributed Processing Techniques and Applications (PDPTA) , Las Vegas, USA, july 1998.

13. "Dynamic Object Positionning"
François Bourdon, Pascal Chatonnay, Bénédicte Herrmann et Laurent Philippe
Conférence Internationale, ECOOP'96, Editée par LNCS, Linz, Austria, juillet 96.

14. "Performances of the Logical Ring Load Balancing Algorithm"
Hervé Guyennet, Bénédicte Herrmann, François Spies et Laurent Philippe
Conference Internationale, IPDCS 1994, Ed. IASTED, Washington, USA, Octobre 1994.

15. "Simulation of Dynamic Load Balancing Algorithms for Multicomputers"
Hervé Guyennet, Bénédicte Herrmann, Laurent Philippe et François Spies
Conférence internationale , Singapour, septembre 1994.

16. "A performance study of dynamic load balancing algorithms for multicomputers"
Hervé Guyennet, Bénédicte Herrmann, Laurent Philippe et François Spies
Conférence internationale MPCS'94, IEEE Ed., Ischia, Italie, Mai 1994.

17. "Distributed scheduling for multicomputer"
Laurent Philippe et Guy-René Perrin
Conférence internationale HPCN'94, Springer Verlag Ed., Munich, Allemagne, Avril 1994.

18. "Building a distributed UNIX for multicomputers"
Bénédicte Herrmann et Laurent Philippe
Conférence internationale PACTA'92, Barcelone, Espagne, Septembre 1992.

19. "UNIX on a multicomputer : the benefits of the Chorus architecture"
Bénédicte Herrmann et Laurent Philippe
Conférence internationale Transputers'92, Arc et Senans, France, Mai 1992.

20. "Multicomputers UNIX based on Chorus"
Bénédicte Herrmann et laurent Philippe
Conférence internationale EDMCC2, LNCS, Springer Verlag Ed., 1990, Munich, Allemagne, Avril 1990.

10.6 Communications dans un colloque à audience nationale

21. "Spécification de services sous CORBA pour une extension meta-computing de `Scilab`"
Sylvain Contassot-Vivier, Frédéric Lombard Jean-Marc Nicod et Laurent Philippe
Douzième Rencontres Francophones du parallélisme (RenPar'2000),
Besançon, France

22. "Administration automatique d'applications"
Huah Yong Chan, Pascal Chatonnay, Bénédicte Herrmann et Laurent Philippe
Conférence française CFSE'1, Chapitre français de l'ACM SIGOPS, Rennes juin 99

23. "Un gestionnaire de placement interdomaine sur CORBA",
Huah-Yong Chan et Laurent Philippe,
Dixièmes Rencontres Francophones du parallélisme (RenPar'10), Strasbourg, France, juin 98.

24. "Une méthode multicritère pour l'allocation dynamique des ressources aux objets"
Pascal Chatonnay, Laurent Philippe et François Bourdon
Deuxiemes Journees de recherche sur le Placement Dynamique et la Répartition de Charge (JRPRC2). Lille, France, 1998

25. "Etude comparative d'algorithmes d'équilibrage de charge"
Pascal Chatonnay, Bénédicte Herrmann et Laurent Philippe
SIPAR Workshop 1995,N. Droux editor, Biel, Suisse, Octobre 1995.

26. "Equilibrage de charge dans les systèmes à objets, une expérience avec COOL v2",
Pascal Chatonnay, Bénédicte Herrmann, Laurent Philippe et François Bourdon
Journées de recherche sur le placement dynamique et la répartition de charge. Université Pierre et Marie Curie - Paris. les 11 et 12 mai 1995.

27. "Un serveur de tubes UNIX pour Chorus"
Bénédicte Herrmann et Laurent Philippe
publié dans les actes de la conférence Convention UNIX, Paris, 1989.

10.7 Rapports de projets et de contrats

28. "Single System Image"
rapport technique du projet ESPRIT PUMA, 1990.
29. "Dynamic Reconfiguration on Chorus"
rapport technique du projet ESPRIT PUMA, 1992.
30. "Autour de l'équilibrage de charge dans les systèmes distribués", rapport 1 du marché SEPT 95 1W 005,
31. "Spécifications d'un gestionnaire d'objets pour COOLv2", rapport 2 du marché SEPT 95 1W 005,
32. "Un gestionnaire de placement dynamique pour le système à objets COOL v2", rapport 3 du marché SEPT 95 1W 005,
33. "Le placement interdomaine sur COOL-ORB, problématique", rapport 1 du marché SEPT 96 1W 027.
34. "Le placement interdomaine sur COOL-ORB, spécifications", rapport 2 du marché SEPT 96 1W 027.

Annexe A

La méthode Taguchi

Calcul des interactions entre facteurs

Dans la suite, nous donnons un ensemble de définitions, et les calculs correspondants, afin de traiter les données récupérées à l'issue de la phase d'expérimentation, dans le but de pouvoir les analyser.

Soit M la moyenne générale des réponses.

Soit Y_i la réponse à l'issue de l'expérience i.

Définitions

Effet moyen d'un facteur A au niveau N :

$E_{(A=N)}$ =(Moyenne des réponses lorsque $A = N$)$-M$

exemple : $E_{(CA=5)} = \frac{(Y_3+Y_4+Y_{12})}{3} - M$

Interaction entre deux facteurs A et B à des niveaux respectifs i et j :

$I_{(A=i,B=j)}$= (Moyenne des réponses lorsque $A = i$ et $B = j$) $-M - E_{(A=i)} - E_{(B=j)}$

exemple : $I_{(Load=0,Com=-30)} = \frac{(Y_8)}{1} - M - E_{(Load=0)} - E_{(Com=-30)}$

Résultats

Soient $[A]$ le vecteur associé au facteur A tel que, lorsque A est à son n-ième niveau, la n-ième composante du vecteur $[A]$ soit à 1, tandis que les autres sont nulles.

exemple : Lorsque $Load = 5$ (niveau 3), $[Load] = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$.

Définition de la réponse théorique

La réponse théorique d'un système est la somme de la moyenne générale, des effets de chaque facteur, et des interactions des facteurs interagissant.

exemple : Dans le cas d'un plan à 2 facteurs A et B à 2 niveaux, en interaction, la réponse théorique est :

$$\widetilde{Y} = M + \left[\begin{array}{cc} E_{A=1} & E_{A=2} \end{array}\right][A]$$
$$+ \left[\begin{array}{cc} E_{B=1} & E_{B=2} \end{array}\right][B]$$
$$+^t [A] \left[\begin{array}{cc} I_{A=1,B=1} & I_{A=1,B=2} \\ I_{A=2,B=1} & I_{A=2,B=2} \end{array}\right][B]$$

Une fois tous les effets calculés et toutes les interactions calculées, on regroupe les résultats dans une formule donnant la réponse théorique :

$$\widetilde{Y} = M + \left[\begin{array}{cccc} E_{(Load=-30)} & E_{(Load=-15)} & E_{(Load=0)} & E_{(Load=5)} \end{array}\right][Load]$$
$$+ \left[\begin{array}{cccc} E_{(Com=-30)} & E_{(Com=-15)} & E_{(Com=0)} & E_{(Com=5)} \end{array}\right][Com]$$
$$+ \left[\begin{array}{cc} E_{(CA=0.01)} & E_{(CA=0.1)} \end{array}\right][CA]$$
$$+ \left[\begin{array}{cccc} E_{(Amort=1)} & E_{(Amort=2)} & E_{(Amort=5)} & E_{(Amort=20)} \end{array}\right][Alpha]$$
$$+ \left[\begin{array}{ccc} E_{(Rk=1)} & \cdots & E_{(Rk=3)} \end{array}\right][R_k]$$
$$+^t [Load] \left[\begin{array}{ccc} I_{(Load=-30,Com=-30)} & \cdots & I_{(Load=-30,Com=5)} \\ \vdots & \ddots & \vdots \\ I_{(Load=5,Com=-30)} & \cdots & I_{(Load=5,Com=5)} \end{array}\right][Com]$$
$$+^t [Load][\ldots][CA]$$
$$+^t [Load][\ldots][Amort]$$
$$+^t [Load][\ldots][DistantLoad]$$
$$+^t [Com][\ldots][CA]$$
$$+^t [Com][\ldots][Amort]$$
$$+^t [Com][\ldots][DistantLoad]$$
$$+^t [CA][\ldots][Amort]$$
$$+^t [CA][\ldots][R_k]$$
$$+^t [Amort][\ldots][R_k]$$

Annexe B

Les fichiers de simulation d'applications tests

Les formats des fichiers de simulation

Nous avons utilisé quatre fichiers différents générés par le simulateur, «plantest» , «config», «result» et «param». Ces quatre fichiers sont nécessaires dans nos applications.

Les plans de test

Ces fichiers contiennent des informations donnant la correspondance entre les fichiers résultats, les fichiers paramètres et les fichiers de configurations.

Le format d'une ligne du fichier plantest est le suivant :

```
Chemin_du_fichier_config Chemin_du_fichier_result Chemin_du_fichier_param
```

Le fichier de plan de tests est la clé de voûte de tous les autres fichiers. Il nous permet de connaître l'emplacement des fichiers config, param et result.

Les fichiers de configurations

Ces fichiers configurent les objets en répartissant les clients, les serveurs, les clients/serveurs et les serveurs à travers les sites au lancement d'une simulation.

Le format des fichiers de configuration est le suivant :

```
c : client
```

```
i : client/serveur
```

```
s : serveur
```

```
c objectName domainName siteName delay duration cpuConsum io nbServerInvoc [serverName
start duré garde volume]*
```

```
i objectName domainName siteName nbLoop cpuConsum io nbServerInvoc [serverName
garde volume]* nbConstrainte [Constrainte]*
```

```
s objectName domainName siteName nbLoop nbConstrainte [Constrainte]*
```

Un exemple de fichier de configuration pour le test 11S3I20C

```
s serv1 galaad perceval 5 0
s serv2 galaad perceval 3 0
s serv3 galaad perceval 2 0
s serv4 galaad gauvain 1 0
s serv5 galaad gauvain 4 0
s serv6 galaad gauvain 2 0
s serv7 galaad gauvain 1 0
s serv8 galaad gauvain 3 0
s serv9 galaad galaad 5 0
s serv10 galaad yvain 0 0
s serv11 galaad yvain 10 0
i cs1 galaad gauvain 2 5 0 1 serv1 1.0 10000 0
i cs2 galaad perceval 1 10 0 1 serv11 1.0 10000 0
i cs3 galaad yvain 3 1 0 1 cs2 1.0 10000 0
c cli1 galaad perceval  1 4000 10 0 1   serv2 0 4000 1 100000
c cli2 galaad perceval  1 4000 10 0 1   serv2 0   4000 0.8 1000
c cli3 galaad perceval  1 3000 1 0 1   cs2 0   3000 0.5 100
c cli4 galaad perceval 1 4000 10 0 2   serv10 0   1500 1 100000
                                       serv8 1500 2500 1 1000
c cli5 galaad perceval 1 4000 10 0 2   serv2 1500   2500 1 10000
                                       serv10 0 1500 0.5 1000
c cli6 galaad perceval 1000 3000 10 0 2   serv3 0   500 1 1000
                                          cs3 500 2500 0.9 1000
c cli7 galaad galaad 1 4000 10 0 2   cs1 0   1500 1 1000
                                     serv4 1500 2500 0.7 1000
c cli8 galaad galaad 3000 1000 10 0 1 cs2 0 1000 1 100
c cli9 galaad galaad  1 4000 10 0 2  serv5 0   4000 0.4 100
                                     serv11 1500 2500 1 10000
c cli10 galaad galaad  1 3000 10 0 2 serv6 0   1500 0.6 100
                                     serv2 1500 1500 1 100000
c cli11 galaad galaad  1 1500 10 0 2 cs1 0   1500 0.8 100
                                     serv9 0 1500 1 100000
c cli12 galaad gauvain 1 4000 10 0 2 serv8 0   1500 1 100000
                                     serv1 1500 3500 0.9 1000
c cli13 galaad gauvain 1 3000 10 0 1 serv7 0   3000 1 1000
c cli14 galaad gauvain 1500 2500 10 0 1 cs1 0   2500 1 100000
c cli15 galaad gauvain 1 4000 10 0 2 serv11 0 1500 0.8 1000
                                     serv5 1500 2500 1 10000
c cli16 galaad gauvain  1500 1500 10 0 2 serv8 0 1500 0.3 100
                                      serv11 0 1500 1 100000
c cli17 galaad yvain  1 4000 10 0 2 cs3 0   1500 1 10000
                                   serv9 1500 2500 1 10000
c cli18 galaad yvain  1 4000 10 0 1 serv9 0   4000 0.7 100
c cli19 galaad yvain 1 3000 10 0 2  cs3 0   1500 1 100000
                                    serv10 1500 1500 0.4 1000
c cli20 galaad yvain 3000 1000 10 0 1 serv3 0   1000 1 100000
```

Les fichiers result

Ces fichiers nous informent sur le temps et le nombre de migrations des serveurs et des clients/serveurs ainsi que sur le nombre d'invocations des clients.

Le format des fichiers de résultats est le suivant :

```
[start|stop] objectType objectName siteName stopTime localTime nbInvoc
```

ou

```
[preMigrate|postMigrate] serverName state objectType objectName domaineName siteName
execTime avgTpi nbInvoc
```

Le premier format montre quand les objets commencent ou arrêtent, ainsi que le nombre d'invocations **nbInvoc** faites par le client. Ceci nous permet d'analyser les performances de clients.

Le deuxième montre quand et où les objets migrent, ainsi que la moyenne du temps de traitement d'invocations (**avgTpi**) du serveur (client/serveur) avant (**preMigrate**) et après (**postMigrate**) migration. Ceci nous permet de savoir combien de migrations sont réalisées pour le serveur et de comparer les performances de serveur avant et après le placement.

Avec les fichiers de résultats, notre visualisateur, développé par les étudiants DESS en Java, peut décrire les résultats en graphique.

Les fichiers param

Ils réalisent les paramétrages du POM et du COMOBS. Les fichiers paramètres fixent les seuils de charge et de communications. Le format des fichiers de paramétrages du gestionnaire d'objets est le suivant :

```
pom site seuilLoad seuilCom seuilCA rapportFacteurs reorgPeriod
```

```
load site nbSite seuilMiniEfface exchangeLoadPeriod
```

```
comobs site alpha beta defaultMasse ageMiniMig seuilRelPart compRPPeriod
```