
1 Introduction

Le développement et l'expansion de plus en plus importants du réseau mondial Internet sont à l'origine de nombreux nouveaux services et applications. Les conférences vidéo, la vidéo sur commande, le commerce électronique, l'enseignement à distance ne sont que quelques exemples de ces nouvelles applications. Cependant, les environnements particuliers dans lesquels ces applications sont déployées font que leur construction représente un vrai défi pour les programmeurs. Les caractéristiques inhérentes du réseau comme la hétérogénéité, l'asynchronisme, la distribution et la limitation des ressources imposent lors de la construction d'une application répartie la prise en compte non seulement des aspects purement fonctionnels mais aussi des problèmes (appelés par opposition non fonctionnels) comme la sécurité, les encombrements réseau, la tolérance aux pannes, le temps de réponse, etc.

L'ensemble des propriétés non fonctionnelles et leur gestion définissent la manière dont l'application réagit face à des problèmes ou des variations de l'environnement d'exécution. Ils déterminent sa **qualité de service** (QoS en abrégé). Le **niveau de QoS** est caractérisé par les "valeurs" particulières des propriétés non fonctionnelles déterminées par les types de gestion choisis. Maintenir le niveau de QoS stable malgré les variations d'environnement est un des grands défis des recherches d'aujourd'hui. A cette problématique se rajoute également l'aspect complémentaire de modification de la QoS selon l'évolution des besoins.

Les outils de construction d'applications réparties proposées aujourd'hui ne gèrent pas la QoS. En effet, des infrastructures comme CORBA de l'OMG [GGM97], JAVA-RMI de Sun [Sun97], DCE de l'OSF [OSF] gèrent les aspects système de manière transparente pour les programmeurs. C'est aussi une des raisons principales ayant contribué à l'utilisation de ces outils et à la "vulgarisation" des applications réparties. Cependant, vu que de tels systèmes ne prennent en compte que certains des aspects non fonctionnels et la gestion de ces aspects est indifférente à la sémantique des applications, leur utilisation perd de son efficacité quand des besoins spécifiques sont exhibés. C'est par exemple le cas des applications réparties construites au-dessus de CORBA et devant vérifier des contraintes temporelles.

Ayant reconnu comme très complexe, voire impossible, la construction d'un système fournissant toute la panoplie de services dont une application pourrait avoir besoin, les recherches autour des applications réparties ont pris une nouvelle direction. La question qui se pose ne concerne plus la réalisation d'un environnement rendant transparents les aspects système mais tend vers une implémentation ouverte qui permet plus de contrôle aux concepteurs d'applications. L'adaptabilité est le mot clé de cette nouvelle approche. Une adaptabilité qui concerne l'application et également les mécanismes système et qui pour l'instant se révèle être une réponse prometteuse aux besoins de gestion de la qualité de service.

2 Le stage

Ce stage de Magistère troisième année a été effectué dans le laboratoire SIRAC (Systèmes Informatiques Réparties pour Applications Coopératives) dont les activités de recherche concernent les problèmes de réalisation de services et d'outils pour la construction d'applications réparties.

S'orientant vers la nouvelle mouvance d'applications et de services adaptables, il a été très important pour les futures recherches de l'équipe d'avoir une vision globale des travaux

existants autour de l'adaptabilité et de la réalisation de la qualité de service. Le but de ce stage est justement d'essayer de répondre à ce besoin et de fournir l'état de l'art correspondant.

Etant donné la nouveauté de la problématique et la généralité des notions considérées, la présentation qui suit est certainement loin d'être complète en ce qui concerne la liste des projets existants. Néanmoins nous avons essayé de couvrir la totalité des courants et des approches adoptées dans la communauté scientifique en ce qui concerne la QoS et l'adaptabilité.

Nous commençons ce rapport par une partie consacrée à la notion de qualité de service, les différentes interprétations qui y sont associées et les principales motivations des recherches autour la QoS. Après avoir donné une définition et vu les différents aspects de la problématique, nous continuons avec la deuxième partie traitant l'adaptabilité. Nous fournissons une classification des travaux existants et essayons d'expliquer la liaison entre QoS et adaptabilité. La conclusion fournit un résumé de la vision globale que nous avons eue des travaux dans le domaine incluant les différents problèmes à résoudre et les perspectives. Les projets auxquels nous faisons référence tout au long de cet exposé sont listés et accompagnés d'une brève présentation dans l'annexe à la fin de ce rapport.

3 La qualité de service (QoS)

Dans cette partie nous allons essayer d'expliquer ce que cache le terme qualité de service si largement utilisé ces derniers temps. Tout d'abord nous allons définir la notion et ceci non seulement de manière générique mais en spécifiant également le sens le plus commun du terme. Nous considérerons les différents aspects et les principales motivations des recherches autour de la problématique de QoS. Tout au long de l'exposé nous essayons de fournir les arguments nous menant à la deuxième partie concernant l'adaptabilité.

3.1 Définition

La qualité de service (QoS) est le terme utilisé pour représenter l'ensemble des contraintes imposées par un usager (être humain ou composant logiciel) sur la performance d'une application lors de son exécution [Siq98]. Pour pouvoir fournir les services demandés (aspects fonctionnels) de manière satisfaisante, une application répartie a besoin de gérer des aspects complémentaires (non fonctionnels) comme le type de communication, la gestion d'état partagé, la sécurité etc. Pour cette raison certains auteurs définissent la QoS comme "toutes les propriétés autres que le comportement fonctionnel d'une application"[Sch98].

En prenant en compte le fait que différentes applications peuvent considérer différents aspects non fonctionnels et/ou gérer les mêmes aspects de manières diverses (différentes valeurs pour les propriétés non fonctionnelles), nous adoptons une définition légèrement différente de celle qui a été citée plus haut:

*La QoS est définie par l'ensemble des propriétés non fonctionnelles d'une application.
Son niveau est déterminé par les valeurs affectées à ces propriétés.*

La définition générique que nous venons de donner n'est considérée que depuis peu dans les milieux scientifiques. Au fait la première utilisation du terme QoS a été dans le cadre des performances réseaux et concerne la transmission de données. Les caractéristiques définissant alors la QoS sont la latence, la largeur de bande passante, le débit, etc. Le niveau de QoS peut être *best effort* ou assurer une bande passante donnée. Cette interprétation est encore très fréquente (des fois elle est même sous-entendue) surtout dans le cas d'applications multimedia dont les contraintes de performance s'expriment en termes de nombre de trames par seconde.

3.2 Problématique de la QoS

La QoS est intrinsèquement liée à la notion de **variation**. Le type de QoS varie en fonction de l'application considérée, le type de QoS peut évoluer avec les besoins des applications et des usagers, elle peut être gérée de différentes manières en fonction des différents déploiements d'une application ou en reflétant les fluctuations des conditions dans un seul environnement. En effet, les recherches autour de la QoS considèrent deux points principaux:

- *Les moyens d'assurer un niveau donné de QoS*: Une application a besoin d'un certain niveau de QoS pour fonctionner correctement d'après certains critères. Par conséquent il est très important de pouvoir maintenir le niveau de QoS malgré les fluctuations et/ou changements de l'environnement d'exécution. Il peut s'agir de variations temporelles (la disponibilité des ressources est un paramètre qui varie dans le temps) ou de variations spatiales (l'application doit pouvoir être déployée sur de différents types d'environnement). Pour arriver à maintenir le niveau de QoS il faut donc prendre en compte ces différentes variations et pouvoir adapter la gestion de la QoS en conséquence. Remarquons que nous venons de parler d'**adaptation**. Elle apparaît naturellement dans la discussion traitant la variation.

Nous pouvons classer cette problématique comme classique puisque le besoin de garantie d'un certain niveau de QoS a été ressenti dès le début des travaux autour des applications réparties. Cependant les solutions sont souvent ad hoc, visent des domaines spécialisés et sont par conséquent difficiles à utiliser.

Parallèlement à cette problématique classique se développe une nouvelle branche des recherches autour de la QoS concernant la modification de la QoS d'une application.

- *Les moyens de modifier la QoS*: Nous avons dit que la QoS est définie par l'ensemble des propriétés non fonctionnelles d'une application et par leur gestion. Il est très vraisemblable de vouloir faire évoluer une application, vouloir lui rajouter de nouvelles fonctionnalités ce qui de son côté entraînera le rajout de nouvelles propriétés non fonctionnelles ou la modification (de la gestion) des existantes. Dans ce cas nous parlons de possibilité de varier la QoS de l'application et cet aspect peut être considéré de manière orthogonale à la variation de l'environnement. Si nous disposons de moyens de varier la QoS, on peut conclure qu'il est possible d'adapter une ou plusieurs entités dans la pyramide d'exécution (application, système, couches intermédiaires). Encore une fois, la gestion de la variation suggère la possibilité d'adaptation.

Les deux points discutés sont reliés entre eux. En effet, si maintenir le niveau de QoS s'avère impossible, une dégradation de la QoS va s'imposer. D'autre part, une modification de la QoS n'a pas de sens si le niveau désiré ne peut pas être assuré.

3.3 Les différentes instances du problème

Dans la section définissant la notion de QoS nous avons donné l'exemple de la gestion des ressources réseau mais ce n'est qu'une facette de la problématique. Considérons quelques aspects de QoS identifiés lors de la construction d'applications réparties et voyons comment les différentes notions et problèmes discutées dans les deux sous-sections précédentes sont reflétés dans leurs cas.

- **La limitation de ressources**: un aspect de la QoS d'une application est la capacité de fonctionner malgré l'insuffisance de ressources. Ces ressources peuvent concerner les ressources réseau comme la largeur de la bande passante, la latence, etc. et dans ce cas ils affectent la manière dont les différentes parties de l'application interagissent. Il peut s'agir des ressources d'un système centralisé qui influencent une des entités de l'application répartie. Parmi ce type de ressources sont le temps CPU, la mémoire ou la puissance de la batterie

si il s'agit d'un ordinateur portable. Les ressources peuvent intégrer aussi la notion complexe de temps et déterminer la possibilité de vérifier des contraintes de temps dur ou de temps mou.

Nous voyons que la disponibilité de ressources affectent directement le fonctionnement de l'application. Si elle ne prend pas en compte ce fait, ne fournit aucune gestion des ressources (la gestion des ressources étant typiquement un aspect non fonctionnel) et ne prévoit pas de mode de fonctionnement lors de leur insuffisance, il est fort possible que le service ne soit pas fourni. Donc pour assurer un niveau de QoS donné, l'application doit pouvoir prendre en compte l'aspect variable de la disponibilité des ressources. En ce qui concerne le niveau de QoS il peut s'agir de best effort (aucune gestion complémentaire, la performance dépend de l'état de l'environnement) ou alors de garantie d'accès à la ressource limitée (possibilité de prédiction et de planification de l'exécution, bonne QoS). Le niveau de QoS peut être encore meilleur si une optimisation de l'utilisation des ressources est faite.

- **La sécurité:** certaines applications réparties manipulent des informations confidentielles qui doivent être protégées contre les différentes attaques qu'il peut y avoir lors de la transmission sur le réseau ou même lors de l'exécution d'un composant sur un système d'exploitation. Des exemples typiques sont la commerce électronique impliquant la communication du numéro de carte bleue et les opérations bancaires. Il est évident dans ce cas que si la sécurité n'est pas garantie, les applications ne fournissent pas un service de qualité et leur fonctionnement est compromis ou en d'autres mots la QoS n'est pas satisfaisante. Pour fournir la sécurité demandée lors d'une transmission, il faut prévoir des moyens de préservation de l'information par chiffrement ou par l'utilisation de voies de communication dédiées ainsi que des protocoles d'authentification garantissant l'identité des parties impliquées dans une opération de ce sorte. Nous voyons encore une fois que la gestion de la QoS est reliée à des aspects non fonctionnels. Pour assurer un niveau de confidentialité, il faut être conscient de l'état (trafic, pannes, etc) et du type (internet, intranet) des voies de communication. D'autre part il sera intéressant de pouvoir choisir entre différents types de sécurité (discrétionnaire, mandataire, etc.).

- **La tolérance aux pannes:** Il n'y a pas de système parfait qui ne tombe jamais en panne. Ce phénomène est fortement ressenti dans le cas des applications réparties où sont impliqués plusieurs systèmes reliés par un réseau. Le taux de probabilité d'une panne est beaucoup plus élevé que dans le cas d'une application centralisée. L'exécution d'une application répartie doit pouvoir faire face aux pannes qui peuvent rendre indisponibles certains ou/et empêcher la communication et la coordination entre d'autres composants de l'application. Pour gérer cet aspect de la QoS de l'application plusieurs mécanismes peuvent intervenir et nous en citons que quelques uns (remarquons que ce sont tous des aspects non fonctionnels): la réplication pour garantir la disponibilité d'un service (de son côté il pose des problèmes de cohérence), les techniques de cache, la construction d'état global, la détection d'une panne, etc. L'efficacité des mécanismes choisis définit le niveau de QoS. Le type de QoS est caractérisé par les types de pannes tolérées. Modifier la QoS dans ce cas (rendre un service encore plus disponible) ne serait rien d'autre que pouvoir tolérer d'autres types de pannes.

- **La persistance:** Nombreux sont les cas où des applications réparties manipulent des informations persistantes. Des exemples classiques sont les systèmes financiers ou les systèmes concernés avec des informations de santé publique. La persistance est donc un autre

exemple d'aspect non-fonctionnel qui doit être géré par les applications réparties. La gestion (et donc le niveau de QoS fourni) peut varier à cause des différents mécanismes disponibles dans les différents systèmes (protocole de sauvegarde, format des données sauvegardés, stockage) ou alors elle peut être paramétrisé selon les besoins des utilisateurs (informations pertinentes, contexte).

- **L'aspect transactionnel:** Les transactions sont de plus en plus impliqués dans le fonctionnement des applications réparties lors des communications entre composants, pour gérer les pannes, etc. La capacité d'une application de garantir la sémantique d'un type de transaction (mécanismes de stratégie optimiste, pessimiste, utilisation de 2PC, etc.) définit son niveau de QoS et peut représenter un point crucial pour la réalisation d'un service. Les différents types de transactions (propriétés ACID, sagas, transactions imbriquées, etc.) définissent de leur côté différentes QoS.

- **Le mode de fonctionnement (connexion/déconnexion):** Il s'agit ici du mode de fonctionnement des équipements portables (ordinateur, téléphone cellulaire, etc) et du fait que ces équipements peuvent ou non être connectés au réseau. Une application qui s'exécute en partie sur un mobile est confrontée à de nombreux problèmes. L'idéal pour l'utilisateur du mobile est de pouvoir travailler avec des applications réparties normalement sans subir des effets indésirables à cause de ses déplacement et avec le moindre désagrément lors d'un changement de mode. La liberté de l'usager est directement liée aux mécanismes de gestion d'aspects non fonctionnels comme les techniques de cache, la gestion de conflits d'accès sur de données partagées, etc. Le degré de liberté définit la QoS de l'application. Nous pouvons imaginer la complexité de gestion si l'usager a besoin de plus de liberté (modification de QoS) ou si cette liberté doit être simulée lors d'une déconnexion (changement d'environnement, assurer un niveau de QoS).

Nous voyons que la QoS recouvre de nombreux aspects et que dans tous les cas les problématiques liées à la variation (variation d'environnement et maintenir un niveau de QoS ou variation de la QoS) sont présentes. Cependant dans la plupart des travaux l'accent est mis sur l'assurance et la maintenance d'un niveau de QoS. Ceci s'explique par le développement historique des recherches autour de la QoS. En effet les différentes facettes de la QoS ont fait l'objet des recherches de communautés scientifiques spécialisées qui ont cherché et trouvé des solutions spécifiques pour la gestion du problème de leur intérêt: tolérance aux pannes, sécurité, mobilité, etc. Il n'est que récemment que la facilité de programmation a commencé à être recherchée en s'intéressant aux techniques facilitant la modification de la QoS et ceci est démontré clairement par le premier effort de standardisation qui est issu des nombreux travaux autour de la QoS et qui ne considère que le problème de la garantie d'un niveau de QoS.

3.4 Efforts de standardisation

Le premier standard autour de la problématique de la QoS [ISO], appelé *QoS Framework*, a pour but de fournir une base commune pour les différents standards spécifiant ou référant des exigences et des mécanismes de QoS dans un environnement technologique. C'est un travail qui reprend les spécifications données dans le modèle de référence pour les systèmes distribués ouverts [ITU] et qui décrit les moyens de caractérisation de la QoS, de spécification des demandes de QoS et de gestion de la QoS.

Le standard a été inspiré par la problématique des télécommunications et des réseaux tout en essayant de fournir des notions facilement adaptables pour d'autres domaines.

Définissant tout d'abord de nombreux termes comme la *caractéristique de QoS* qui est "un aspect quantifiable de la QoS", la *mesure de QoS* définie comme étant "une ou plusieurs valeurs observées d'une caractéristique de QoS", l'*information de QoS*, etc., le standard consacre une grande partie du document à la gestion de la QoS.

La *gestion de la QoS* est formée par l'ensemble d'activités mises en place dans le système pour surveiller, contrôler et administrer la QoS. Pour cela il faut définir des *mécanismes de QoS* permettant la création de conditions **garantissant un niveau de QoS** avant le début d'une activité, la *surveillance de la QoS*, le *contrôle* et la ***maintenance de QoS***. ISO identifie trois phases de la gestion de QoS: la *phase de prédiction* pendant laquelle l'application évalue l'environnement dans lequel elle va être exécutée, la *phase de définition* qui a pour but de mettre en oeuvre les conditions pour avoir une QoS désirable avant le début d'activité de l'application et qui peut inclure la négociation de QoS, des garanties, la mise en place de mécanismes particuliers de gestion de la QoS, etc. et la *phase opérationnelle* qui comme son nom l'indique est responsable de la gestion de QoS pendant l'exécution de l'application.

Ce résumé de la description de la gestion de QoS dans le document montre clairement que le problème considéré est la maintenance et l'assurance d'un niveau de QoS donné, La nature de la QoS est définie une fois pour toutes en choisissant les différentes caractéristiques de QoS à considérer.

Comme tout standard, le QoS Framework reste très générique et plusieurs questions se posent autour de la gestion de la QoS. Un point que ISO mentionne brièvement en discutant la phase de prédiction et qui en effet représente un grand problème est le choix de moyens efficaces et adaptés de spécification de la QoS. Le deuxième point concerne les actions à prendre en cas d'échec de la négociation entre application et système avant l'exécution ou en cas de dégradation pendant l'exécution. Le standard n'est pas explicite sur leur nature (et c'est compréhensible vu la diversité des aspects de QoS) mais elles ont une caractéristique commune: elles contribuent à la réalisation du fonctionnement de l'application malgré les variations de l'environnement ou en d'autres mots ce sont des activités d'**adaptation**.

De telles activités/mécanismes d'adaptations ne peuvent pas être impliqués dans le processus de gestion de la QoS si l'**aspect d'adaptabilité** n'a pas été considéré lors de la construction de l'application ou la réalisation du système. L'adaptabilité est déjà une approche acceptée lorsqu'il s'agit de la gestion de ressources (p. ex protocole RSVP [IETF]).

3.5 Synthèse

La QoS d'une application est déterminée par l'ensemble de ses propriétés fonctionnelles et leur gestion. Historiquement les recherches autour de la QoS ont commencé par des efforts dispersés sur des domaines spécialisés aboutissant à des solutions ad hoc très diverses. La problématique principalement considérée était celle d'assurance et de maintenance d'un niveau de QoS. Entre les deux solutions extrêmes de gestion de la QoS: l'une ne visant la gestion que dans une situation particulière (protocole) et l'autre essayant de modéliser globalement toutes les variations possibles et de fournir pour chaque cas une gestion adéquate correspondante, apparaît une solution de compromis. Cette solution essaie de recouvrir toutes les variations de l'environnement mais au lieu de fournir explicitement tous les mécanismes et/ou programmes de gestion correspondants elle prévoit les moyens d'adaptation.

La problématique autour de la QoS évolue et aujourd'hui la communauté scientifique s'intéresse également à la question de facilité de programmation et de modification de la QoS d'une application. L'expérience acquise lors des travaux précédents et la motivation de vouloir

éviter le reprogrammation complète de l'application ont orienté les recherches également vers la solution d'adaptation.

4 L'adaptabilité

Ayant discuté dans la partie précédente l'importance de l'adaptation pour la gestion de la QoS, nous consacrons cette partie à l'adaptabilité. Nous fournissons une classification des travaux qui existent autour de cet aspect. L'exposé suit une logique "descendante" en ce qui concerne la QoS. Nous commençons par les différents moyens d'expression de la QoS et la stratégie d'adaptation, continuons sur le processus d'adaptation et les différentes relations qui peuvent exister entre les couches dans un système (dont l'application) et terminons sur les mécanismes mis en place pour la réalisation de l'adaptation.

4.1 Qu'est-ce l'adaptabilité?

On entend par adaptabilité la capacité de réagir face aux variations des contraintes (de l'environnement) ou des besoins. Nous considérons l'adaptabilité comme la réponse de la problématique de QoS sachant que la QoS pose des questions de maintenance du niveau de service ou de modification de la QoS.

La généralité de la notion, tout comme le terme QoS, implique que selon les différentes interprétations que l'on choisit, selon les besoins du type d'application considéré, selon les systèmes et la culture des programmeurs, il est possible d'avoir de nombreuses approches différentes à la réalisation de l'adaptabilité. Ce sont ces différentes approches que l'on essaiera de présenter dans la suite.

4.2 Classifications des travaux existants

Dans la présentation que nous proposons nous nous intéressons tout d'abord aux stratégies d'adaptation et les moyens de leur expression. Nous discutons l'existence et les types d'expression et présentons les approches procédurales et déclaratives, parlons des langages utilisés pour cette expression, les problèmes rencontrés, etc. Le deuxième critère de classification que nous discutons porte sur le dynamisme de l'adaptation (adaptation statique et adaptation dynamique). La suite continue avec la question de l'entité réellement adaptée et de la participation des différentes couches système dans le processus d'adaptation. La partie termine avec les mécanismes de réalisation d'une adaptation.

Les questions auxquelles nous essayons de répondre sont respectivement: Décriv-on l'adaptation et si oui, de quelle façon? Quand est-ce qu'on l'effectue? Quelle entité est adaptée et qui effectue l'adaptation? Comment adapte-t-on?

4.2.1 Expression d'une stratégie d'adaptation

La stratégie d'adaptation définit les paramètres importants influençant le fonctionnement d'une application et spécifie les actions à entreprendre lorsque ces paramètres ne vérifient pas certaines propriétés (niveau de QoS fourni non satisfaisant). Nous considérons que la stratégie se situe à un niveau plus haut que les mécanismes d'adaptation particuliers puisqu'elle peut faire référence et organiser l'utilisation de plusieurs mécanismes de ce genre.

Nous identifions deux approches principales qui traitent le problème de la stratégie dans les travaux autour de l'adaptabilité. Nous les avons nommées approche de la stratégie implicite et approche de la stratégie explicite.

Stratégie implicite

La stratégie implicite est l'approche la plus fréquemment utilisée aujourd'hui. Il s'agit des cas où l'application fait usage d'un certain protocole sans avoir le droit de contrôle, ni de remplacement. Le protocole encapsule une solution à l'adaptation vis à vis d'un problème donné. La stratégie implicite consiste donc à utiliser ce protocole particulier pour gérer le problème considéré. Les exemples à citer sont nombreux: utilisation d'un service de communication donné (facilités proposés par CORBA, JAVA-RMI ou autre), d'un type de sécurité donné ou un type de gestion de la persistance (par exemple les services fournis par les EJB [Sun-ejb]), choix d'un protocole de réservation de ressources, d'un autre pour la gestion des pannes, etc.

L'adaptabilité proposée dans ces cas a un domaine de validité limité. Elle ne concerne que les variations de l'environnement considéré. Si l'application doit faire face à des changements non prévus dans cet environnement d'exécution ou être déployée dans un environnement "hostile" où ces protocoles ne sont pas efficaces et ne gèrent pas les problèmes de manière adaptée, elle peut perdre beaucoup au niveau de la performance et même se retrouver en état de blocage.

L'utilisation de cette approche se place surtout dans le cas de l'assurance et la maintenance d'un niveau de QoS donné. La modification de la QoS qui implique un changement des protocoles utilisés est dans la plupart des cas difficile à faire.

Stratégie explicite

Les applications réparties qui ont droit de contrôler la définition ou/et le choix des activités d'adaptation exhibent une stratégie explicite. Ce type de stratégie peut faciliter aussi bien la maintenance d'un niveau de QoS que la modification des propriétés fonctionnelles considérées. Cette stratégie peut être fournie soit de manière procédurale, soit de manière déclarative par l'application.

La **manière procédurale** consiste à programmer dans l'application la gestion des aspects liés à l'adaptation. Dans ce cas là, le programmeur de l'application manipule directement dans le code les mécanismes fournis par le système. La granularité de ces mécanisme peut varier et il est possible d'aller de gestion propre des ressources système, pouvant être assez difficile à faire et demandant beaucoup d'expérience, à l'utilisation de services fournis par une couche intermédiaire (dans le cas d'application répartie c'est le middleware) qui encapsulent des services de plus bas niveau. La nature des services peut être très diverse et en conséquence les moyens fournis au programmeur pour leur manipulation peuvent différer considérablement.

Dans le projet Prayer [BG97] l'application utilise des APIs fournies par le middleware pour annoncer ces besoins en termes de QoS. Elle a droit de définir des classes de QoS qui sont typiquement caractérisées par des bornes pour les valeurs acceptables des paramètres comme le débit, l'erreur (nombre de pertes), etc. La même idée, avec une interface plus simple (ne contenant que les méthodes *Request* pour les demandes de l'application et *Report* pour les notifications du système) et toujours centrée autour de la gestion de ressources, est implémentée dans les travaux du groupe sur les systèmes distribués de Dublin [Siq98]. En ce qui concerne les autres types de QoS nous pouvons citer Globe [STK+99] qui donne droit à programmer les stratégies de réplication et de communication pour la tolérance aux pannes ou Cherubim[Qui98] qui permet, en fournissant des APIs bien définies, la programmation de fonctions de sécurité spécialisées par l'application.

Alors que dans les projets cités l'application ne fait que fournir des informations aux services et les mécanismes de gestion restent les mêmes, d'autres projets comme Iguana[GC96]

ou les méta-espaces dans [CBC97] utilisent des APIs de type réflexif qui permettent de choisir l'implémentation d'un certain mécanisme (la réification).

La **manière déclarative**, comme son nom l'indique, consiste à fournir une spécification de la QoS (et de sa gestion) sans nécessairement considérer les mécanismes effectifs à mettre en place lors de la réalisation. L'approche vise donc la possibilité de pouvoir exprimer et documenter explicitement les aspects liés à la QoS pendant la phase de conception et d'éviter la pratique d'implémentation directe dans le code.

La question principale qui se pose concerne l'expression de la QoS et le langage à utiliser. Comme dans les domaines de tous les langages (ADLs, IDLs, langages de programmation, de modélisation, etc.), différentes possibilités se présentent en fonction des caractéristiques que nous voulons prendre en compte.

Le langage *QuAL* (Quality Assurance Language) [Flo94] est étroitement lié à la problématique multimedia et donc à la gestion des ressources réseau et les contraintes de temps. La proximité entre les notions manipulés dans le langage et les mécanismes système facilite le passage entre les deux niveaux. C'est le cas contraire avec le langage *QML* (Quality Modeling Language) [FK98]. Etant une extension du langage UML (Unifying Modeling Language) [UML], il reste très générique et ne considère que l'expression des besoins en termes de QoS sans prêter attention aux mécanismes réels d'implémentation. Les auteurs de *QML* ont également implémentés un environnement permettant l'échange de telles spécifications. Entre *QuAL* se trouvant tout près du niveau système et *QML* qui fournit un modèle "abstrait", le langage *QDL* (Quality Definition Language) essaie de trouver un compromis optimal. Spécialement prévu pour une utilisation avec CORBA et les applications réparties, utilisant trois sous-langages pour définir des contrats entre les différents objets, des structures d'objets distants et des ressources, *QDL* permet l'implication dans la gestion de la QoS aussi bien de mécanismes programmés par l'application que des mécanismes système. Les spécifications de la QoS et de sa gestion sont représentées comme des entités pendant l'exécution.

L'existence de langage déclaratif n'est pas forcée. Il est possible d'utiliser des directives ou des mots-clés comme les attributs de persistance dans les EJBs ou les extensions de l'IDL dans l'adaptation de Cherubim pour CORBA.

Expression de la QoS par rapport au code

L'emplacement de l'expression de la stratégie par rapport au code fonctionnel est un autre critère intéressant de classification. Il peut être considéré de manière orthogonale par rapport au type de stratégie explicite (procédurale, déclarative).

L'approche classique consiste à mélanger au code fonctionnel l'expression de la QoS. L'utilisation des APIs fournies par les mécanismes impliqués est typiquement dispersée dans les modules fonctionnels de l'application. Le grand défaut de cette solution est l'effet d'un code complexe qui est tout d'abord difficile à écrire vu que plusieurs aspects doivent être gérés en même temps par le programmeur en plus des fonctionnalités de l'application et ensuite il est difficile à maintenir et à comprendre justement à cause de la multitude d'aspects entremêlés. En conséquence un tel code est difficile à adapter et à réutiliser.

La deuxième approche n'est apparue que récemment et essaie de remédier à ce problème en fournissant les moyens de décrire les aspects reliés à la QoS séparément du code fonctionnel. L'objectif d'une telle solution est plus orienté "génie logiciel" dans le sens où on vise une programmation modulaire suivant le principe de "séparation des besoins". Dans le contexte de QoS, une telle structuration de l'application permettrait l'adaptation de la stratégie de QoS et la modification de la QoS assurée.

L'idée a été avancé par les travaux effectués dans XEROX autour de la programmation par aspects (*AOP* == *Aspect Oriented Programming*) [KLM+97]. Dans cette approche un aspect est une propriété ou une caractéristique du programme qui a été modélisé comme une entité explicite et ce n'est donc pas obligatoirement un aspect non fonctionnel. D'après les auteurs toutes les entités de ce genre doivent être explicitement représentées dans le code par des "composants" logiciels correspondants. Dans la plupart des cas d'aujourd'hui la structuration du logiciel suit une décomposition fonctionnelle et beaucoup de caractéristiques (dont les aspects non fonctionnels) sont implémentées de manière dispersée. Les auteurs appellent le code résultant "emmêlé" (*tangled code*).

Plusieurs projets de recherche travaillent sur les principes de AOP. Une réalisation est le langage *AspectJ* [LK98]. Extension de Java, il se concentre sur les possibilités "d'injecter" du comportement additionnel dans les programmes en définissant des entités (les aspects) pouvant modifier le comportement des objets en ajoutant à des endroits précis des définitions de variables, d'objets, de méthodes, etc. La prise en compte des aspects dans le code est statique et est faite à la compilation. *AspectIX* [HBG+98] est un projet qui réalise la même idée mais en considérant des applications CORBA. Les objets CORBA ont accès à des APIs fournies par le middleware pour activer/désactiver dynamiquement des aspects. Dans les deux cas cités les aspects peuvent être reliés à un comportement particulier, gérer un aspect non fonctionnel, etc. et sont définis par le(s) programmeur(s) avant l'exécution de l'application. Les deux approches peuvent être classées dans l'expression procédurale (voir plus haut) quand il s'agit de gestion de QoS. L'approche déclarative est adoptée par l'infrastructure des objets de qualité *QuO* avec le langage *QDL* [VZL+98]. *QDL* que nous avons considéré brièvement dans la partie précédente donne la possibilité de spécifier la stratégie de QoS (contrats, ressources, structures, etc.) séparément de l'implémentation des fonctionnalités de l'application.

Discussion

Dans cette sous-partie nous avons présenté les différentes approches en ce qui concerne l'expression de la (stratégie de) QoS. Les approches sont diverses et vont d'une gestion transparente à l'aide de protocoles jusqu'à une spécification explicite de la part de l'application. L'existence d'un tel nombre de solutions témoigne de la recherche active des solutions efficaces. En effet, dans tous les cas de sérieux problèmes se présentent.

L'utilisation de protocoles spécialisés ne permet aucun contrôle de la part de l'application et ne facilite pas la réutilisation et l'adaptation pour d'autres contextes.

Les approches essayant de donner plus de contrôle à l'application sont confrontées au choix d'un langage d'expression (dans notre cas de la QoS) et à la gestion du passage entre cette expression et la réalisation effective. Dans AOP une question qui se pose est celle du choix d'un langage (éventuellement spécialisé) pour la programmation d'un type d'aspect. Un langage manipule un nombre fini de notions et a une pouvoir d'expression limité. D'autre part, si plusieurs langages sont utilisés, l'intégration dans le programme final est problématique. AOP qui travaille surtout du point de vue de la programmation parle du problème du "weaver", le composant qui prend les différentes spécifications/programmations/etc. des aspects et se charge de produire (générer le code) du logiciel résultant. La difficulté est illustrée par les réalisations d'aujourd'hui qui utilisent pour l'expression d'aspects le langage de programmation lui-même. La même problématique est présente dans les approches utilisant des spécifications des besoins de QoS. Plus le langage de spécification est abstrait (exemple QML), plus il est facile à manipuler et à comprendre par une personne humaine, plus il rend difficile le passage vers l'implémentation.

4.2.2 Dynamisme de l'adaptation

Le critère de classification que nous considérons dans cette partie est le dynamisme de l'adaptation. Nous avons identifié trois solutions possibles:

- **Adaptation statique:** Ce cas correspond à une adaptation effectuée **avant** l'exécution de l'application en fonction des connaissances détenues de l'environnement de déploiement. Il peut s'agir d'une adaptation de la gestion pour assurer un niveau de QoS (choisir l'implémentation qui convient le mieux) ou d'une adaptation de la nature de QoS (choix des propriétés non fonctionnelles considérées). C'est le cas dans AspectJ où la prise en compte de certains aspects est faite à la compilation. Dans Jonathan [DHT+98], le fait de choisir d'implémenter les interfaces dédiées à une "personnalité" donnée fait que le type de communication est choisi (adapté) statiquement en fonction des besoins de l'application. Dans le standard des EJB, c'est également avant le déploiement des composants (appelés beans) que le constructeur décide de propriétés comme la persistance. Dans la même catégorie nous mettons les approches où l'adaptation est faite pendant une phase d'initialisation de l'application.

- **Adaptation spécifiée statiquement mais effectuée dynamiquement:** Cette solution consiste à prendre en compte pendant la construction de l'application les différentes variations de l'environnement et de définir les actions d'adaptation (la stratégie) en conséquence. La stratégie est ainsi spécifiée de manière statique (avant le lancement) mais les adaptations sont effectuées dynamiquement. C'est l'approche utilisée dans le cas des applications réparties "classiques" où les constructeurs décident d'utiliser un protocole donné pour la gestion d'un problème relié aux fluctuations de l'environnement. Le choix est statique alors que le fonctionnement de ce protocole prend en compte des informations dynamiques. Dans cette catégorie se trouve également l'infrastructure *QuO* avec la spécification de contrats et d'actions d'adaptation lors de la transition entre les différents niveaux de QoS et leur prise en compte pendant l'exécution. L'approche a été choisie également par les travaux autour des adaptations en fonction du contexte [SAW94] où des règles spécifiées statiquement guident les paramètres d'exécution, par Coda et Odyssey [NSN+97] qui s'intéressent aux mobiles et qui mettent en place des solutions (choisies statiquement) de gestion de la déconnexion, etc.

Dans cette même catégorie se trouvent les projets utilisant la principe de reflexivité permettant un choix dynamique entre les implémentations existantes comme dans DART[RL98] ou les JavaPods, projet en cours dans SIRAC. Une approche encore différente est celle de MultiSpace [GWB+99] où un environnement spécifique est défini, la possibilité de faire des adaptations dynamiquement est donnée à l'administrateur qui a le droit de charger du code à distance et d'installer des composants/services sur les noeuds du système.

- **Adaptation spécifiée et effectuée dynamiquement:** Cette dernière approche va encore plus loin et permet non seulement d'effectuer de l'adaptation pendant l'exécution mais aussi la définition et la mise en place dynamique de la stratégie d'adaptation. Pour l'instant nous ne sommes pas au courant d'une implémentation de cette idée, elle n'est qu'envisagée. AQuA et l'infrastructure des objets de qualité QuO prévoient la possibilité de chargement dynamique dans l'environnement d'exécution des entités représentant un contrat.

4.2.3 Entités concernées par une adaptation

Nous avons parlé de stratégie d'adaptation et de son dynamisme mais nous n'avons pas considéré quel peut être l'objet de l'adaptation et qui participe au processus d'adaptation.

Entité adaptée

Rappelons que nous nous intéressons aux adaptations permettant le déploiement d'une application répartie dans des environnements différents et/ou variables. Ceci n'implique surtout pas qu'il n'y a que l'application qui est adaptée. Les adaptations peuvent toucher toutes les entités impliquées dans l'exécution d'une application c'est à dire réseau (logiciel sur), système d'exploitation, middleware, application, etc.

En ce qui concerne l'*adaptation d'une application* il peut s'agir de sa structure ou de son fonctionnement. Les projets de AOP, AspectJ et AspectIX, peuvent adapter la structure et le fonctionnement des applications en prenant ou pas en compte certains aspects. Les travaux ayant choisi l'utilisation d'un méta-modèle représentant l'application peuvent simplement changer la structure en manipulant le graphe correspondant [CBC98]. L'approche peut être appliquée également pour changer le fonctionnement. D'autre part l'adaptation du fonctionnement peut être prévue par l'application en fournissant plusieurs séquences d'exécution et la possibilité de choix en fonction d'un paramètre d'environnement [BG97].

En ce qui concerne *les adaptations niveau middleware* les idées sont diverses. Il peut s'agir d'adaptation des mécanismes de manière réflexive comme dans les JavaPods (projet en cours dans SIRAC), de choix explicite de mécanismes dans un environnement comme dans Jonathan où la personnalité définit la gestion de la communication ou comme dans l'approche OCI [HNN+99] où le protocole de transport de CORBA peut être non seulement TCP/IP mais aussi UDP/IP ou IP Multicast.

Au niveau *du système d'exploitation* il peut s'agir de système extensible où l'application charge les services dont elle a besoin [SES+96] etc.

Participation dans le processus d'adaptation

Un autre aspect du problème est qui participe au processus d'adaptation. Les participants peuvent avoir plusieurs rôles. Il peut s'agir de *maîtres*: décider en fonction de quoi va être effectuée l'adaptation (paramètres significatifs de points de vue de la QoS considérée), décider du moment de l'adaptation, décider des mécanismes d'adaptation à utiliser, décider des mécanismes disponibles, etc. ou de *servants*: effectuer l'action décidée d'un maître.

Les différents cas de participation dans le processus d'adaptation sont donc très nombreux sachant qu'un de ces rôles peut être attribué à tous les niveaux (application, couches intermédiaires, système).

Les cas extrêmes de processus d'adaptation impliquent une seule entité (l'application uniquement, le système, etc.). Une exemple d'adaptation (décision et action) entièrement faite par l'application est l'approche d'AspectJ n'utilisant que des mécanismes langage d'adaptation ne dépendant d'aucune manière du système. Inversement, des adaptations système sans impliquer l'application (ce qui signifie que ces adaptations ne prennent pas en compte la sémantique particulière de l'application concernée) sont tous les protocoles prédéfinis de gestion de la tolérance aux pannes, de mise en cohérence de données partagées, etc.

Entre les deux extrémités (adaptation effectuée par l'application et adaptation effectuée par le système) se placent de nombreux cas de collaboration. Ceci implique l'existence de connaissances concernant l'application et détenues par le système ou/et vice versa. Dans le cas où l'application connaît, même partiellement, les caractéristiques de l'environnement d'exécution

tion on dit qu'elle est "sensible à l'environnement" (de l'anglais *environnement aware*). Cette "sensibilité" peut être envers les paramètres du réseau (*network aware*), du système d'exploitation (*system-aware*), du contexte qui peut être de location, social, etc (*context-aware*), etc. Inversement, pour un système qui prend en compte les particularité d'une application on dit qu'il est "sensible aux applications" (*application aware*).

Pour montrer la diversité des solutions retenues mais sans vouloir les énumérer toutes (ce qui serait certainement impossible), nous pouvons citer quelques exemples de collaboration.

Une première approche consiste à déléguer toutes les actions d'adaptation au système mais en fournissant des connaissances sur l'application. ("*application aware*"). En ce qui concerne les connaissances, il peut s'agir d'un modèle explicite de l'application. Dans la plupart des cas le modèle est représenté sous forme de graphe dont la nature et les informations peuvent varier. Il peut s'agir d'une approche clairement théorique [KCS98] incluant des calculs et des optimisations ou d'une approche plus "pratique" où le graphe ne représente que la structure globale de l'application et les exigences de proximité des différents objets [HL98]. Les adaptations prennent alors la forme de modification de la structure, de migration de certaines parties (typique dans les applications scientifiques où un équilibrage de la charge est nécessaire), etc.

Dans une deuxième approche, l'application participe plus au processus d'adaptation en indiquant les actions à entreprendre par le système. Le système est toujours "*application-aware*" mais c'est l'application qui devient "*system-aware*". C'est le cas dans Prayer où l'application définit des classes de QoS et indique pour chaque classe une action. Ces actions sont prédéfinies par le système et sont déclenchées lors de la violation de la classe de QoS correspondante. Donner la possibilité à l'application de choisir entre les mécanismes système a été aussi l'approche choisie dans Jonathan.

Il est possible d'aller encore plus loin en ce qui concerne les responsabilités à l'application en lui permettant la définition de mécanismes propres d'adaptation. C'est le cas de QuO avec la définition des contrats, de Globe avec la possibilité de composition des services système pour les techniques de réplication, d'AspectIX avec la programmation des aspects non fonctionnels, Cherubim avec la définition de techniques de sécurité, etc.

La diversité des besoins et des visions des constructeurs font que leurs choix peuvent différer considérablement. Ceci explique les différentes possibilités de réalisation de l'adaptation et les différents degrés de responsabilité de chaque niveau (système, application) dont nous n'avons donné que quelques exemples.

4.2.4 Mécanismes de réalisation

Cette section qui a pour but de présenter les différents mécanismes mis en place pour un processus d'adaptation est étroitement relié à la discussion de la section précédente. En effet, ces mécanismes vont être différents en fonction de l'entité à adapter et la manière de partager les responsabilités. Nous identifions quelques approches principales lors de la définition de ces mécanismes.

La première approche se concentre sur les moyens linguistiques. C'est typiquement le cas où toutes les actions d'adaptation sont gérées directement par l'application. Il peut s'agir des extensions langage comme dans AspectJ ou des approches réflexifs qui permettent la réification des mécanismes du langage comme dans Iguana [GC96]. Dans cette catégorie se trouvent également les efforts d'expression de la QoS comme dans QuAL [Flo94], QML [FK98], QDL [VZL+98], etc.

La deuxième approche est "l'inverse" de la première dans le sens où elle ne considère que des mécanismes système particuliers. C'est le cas des travaux dans Sumatra qui considèrent la migration d'objets et de processus pour des applications mobiles ayant pour but l'amélioration des performances (temps). Sumatra se penche aussi sur les problèmes de surveillance du réseau avec l'outil Comodo. La surveillance et le raisonnement sur l'état des ressources est un problème considéré dans Remos qui fait partie de l'action Quorum. Des recherches sont effectuées sur les techniques de caching, les techniques de transaction, techniques d'opération déconnectée, etc.

De plus en plus les travaux de recherche s'orientent vers la réalisation d'infrastructures fournissant plusieurs services et permettant aux applications d'intégrer leurs besoins spécifiques. De telles infrastructures prennent en compte d'une part un ensemble de services et d'autre part leur organisation/utilisation dans une application répartie. Ces deux aspects sont liés vu que le type de service influence les moyens d'intégration dans une application.

Nombreux sont les projets proposant de telles infrastructures et les approches sont différentes. Il existe des travaux qui sont plus orientés vers une énumération de services en identifiant les situations les plus courantes. C'est le cas de l'environnement ACE [Sch93] qui traite les patrons de communication. Une deuxième approche recherche la généralité et vise à fournir une base qui peut être utilisée pour le développement de services spécialisés. On peut citer Jonathan avec la définition d'un ORB minimal permettant l'implémentation et l'utilisations de plusieurs "personnalités" de communication comme CORBA ou JAVA-RMI. Une troisième approche qui se situerait vraisemblablement entre les deux premières consiste à définir un modèle fournissant un certain nombre de services de base prédéfinis et spécifiant des mécanismes d'organisation. Dans cette catégorie sont Globe [STK+99] avec son modèle à objets fragmentés et les possibilités d'implémenter différentes stratégies de tolérance aux pannes, la mémoire partagée de Khazana [CRS98] et ses primitives pour réaliser une gestion d'état partagé, DART qui s'intéresse au choix des séquences d'exécution et la modification des interactions pendant les invocations de méthodes, etc.

Discussion

Il nous semble que l'effort autour des mécanismes pour la réalisation de l'adaptabilité comme une réponse aux besoins de QoS prend principalement deux formes: expression de la QoS (niveau langage) et mécanismes de réalisation (niveau système). Les travaux se concentrant sur le problème de la liaison entre les deux sont peu nombreux.

Les infrastructures proposant des mécanismes obligent les constructeurs d'applications réparties de produire tout le code à la main. Dans un logiciel complexe demandant des services faisant intervenir plusieurs de ces mécanismes ceci peut être un défi difficilement surmontable.

Les travaux autour des langages d'expression de QoS ne considèrent généralement pas le passage vers l'implémentation. En effet, la génération automatique de code et ensuite la maintenance entre le code généré et sa spécification n'ont pas encore trouvé de solution satisfaisante dans le cas général. En ce qui concerne la QoS les réalisations actuelles traitent le plus souvent la gestion de ressources. Des aspects comme la sécurité, la tolérance aux pannes, etc. ne sont pas abordés. Les recherches sur le projet QuO sont des rares qui permettent l'expression et la liaison avec les mécanismes système de plusieurs aspects de QoS. Cependant le langage n'est pas vraiment déclaratif, les contrats définis ressemblent plus à des petits programmes dans lesquels on fait référence à des modules fonctionnels du logiciel complet.

Une solution intermédiaire et qui gagne de plus en plus de popularité est la réflexivité. La visibilité qu'elle offre du système peut être placée entre la transparence de l'approche déclarative et la visibilité totale dans l'approche où tout le code est écrit manuellement. Utilisée tout

d'abord dans des travaux autour des langages [GC96], elle commence à toucher des domaines comme les systèmes d'exploitation ou les middleware [Bla97]. La réflexivité répond au principe d'implémentation ouverte [MLM+97] où certains points décisifs du système sont accessibles et adaptables. Elle permet de maintenir automatiquement la liaison entre un niveau abstrait (méta-modèle) et le code. Avec la réflexivité il est possible non seulement de rajouter des services dans le code mais aussi de modifier l'implémentation des services existants.

La co-existence des trois approches montre bien que pour l'instant il n'y a pas de solution universelle qui réconcilie l'efficacité et la facilité.

4.3 Synthèse

Dans cette partie nous avons montré comment l'adaptabilité s'inscrit dans la problématique de gestion de la QoS. Nous avons discuté les différentes approches et les avons classifié selon l'existence et les moyens d'expression d'une stratégie, le dynamisme, la participation des différentes entités dans le processus d'adaptation et les mécanismes utilisés lors de ce même processus. Nous avons constaté que plusieurs problèmes et notamment ceux autour de l'expression des besoins de QoS (choix de langage), l'utilisation des mécanismes fournis (travail complexe de gestion pour le programmeur), la liaison entre les besoins "abstrait" de l'application et les mécanismes système, restent encore sans solution satisfaisante.

5 Conclusion

Les recherches spécialisées sur des problèmes différents comme la performance des réseaux, le traitement de la tolérance aux pannes, la synchronisation, etc. ont amené la communauté scientifique à la conclusion que le problème de la QoS dans son sens le plus large est une des priorités à résoudre dans le domaine de systèmes distribués et des applications réparties. D'autant plus que la technologie de distribution gagne de plus en plus de popularité avec l'énorme expansion de Internet et les facilités de collaboration qu'elle permet.

Etant principalement relié au problème de variations, la réalisation de la QoS trouve une solution dans la technologie d'adaptabilité. L'adaptabilité permet aux applications réparties de fournir leurs services malgré les changements des environnements d'exécution, réduit l'effort de déploiement sur d'environnements divers et facilite la modification des applications contribuant ainsi à la réutilisation de code.

La nouveauté de l'idée et la courte période de recherches font que la problématique est loin d'être résolue. Les moyens d'adaptation, leur liaison avec les besoins de QoS des applications et toutes les difficultés connexes font l'objet de nombreux travaux en cours et de débats scientifiques. Il ne faut pas croire que l'adaptabilité sera la solution miraculeuse de tous les problèmes de la répartition. Construire des applications en prenant en compte l'aspect d'adaptabilité en plus des problèmes propres à une situation donnée nécessite de l'effort de prédiction et d'organisation. Il est intéressant de pouvoir estimer cet effort et le comparer en termes d'efficacité de la solution (une solution plus générale tend d'être moins efficace) et en termes de temps de réalisation (grand effort à la première implémentation et révision éventuelle et partielle lors d'un changement) avec l'approche classique de développement de solution particulière pour toute situation différente. Cette estimation pourrait nous emmener à déduire le type d'application bénéficiant de l'approche de l'adaptabilité, décider de ces particularités et créer en conséquence des environnements réduisant considérablement l'effort initial de construction en fournissant un langage de description et en automatisant au moins partiellement la génération du code.

ANNEXE Glossaire des projets

Voici la liste des projets (triés par ordre alphabétique) considérés lors de ce travail. Nous avons essayé de fournir pour chaque projet un résumé présentant les principes des réalisations ainsi que l'adresse à laquelle il est possible de trouver des informations plus complètes.

I. Aspect Oriented Programming (AOP)

D'après AOP le problème principal lors de la construction d'une application avec les moyens d'aujourd'hui est l'impossibilité de modéliser "proprement" toutes les propriétés des cette application. L'approche standard consiste à modéliser un système logiciel en prenant en compte une décomposition fonctionnelle. Le modèle résultant contient donc des entités correspondant aux fonctionnalités considérées. Ces entités se retrouvent aussi au niveau du code sous forme de modules, procédures, etc. Cependant cette approche n'arrive pas à capturer d'autres aspects de l'application comme l'utilisation optimale de ressources, la synchronisation, la gestion d'erreurs de calcul, etc. Au niveau du code leur implémentation n'est pas centralisée mais est dispersée dans différents modules étant les implémentations des entités modélisées. Le phénomène est appelé dans AOP intersection (d'aspects) qui résulte en code "emmêlé" (termes anglais exacts *cross-cutting* et *tangled code*).

L'objectif d'AOP est donc de résoudre le problème d'intersection qui mène à un code illisible et difficilement réutilisable. Pour cela AOP propose l'utilisation de langages pour la description des différents aspects impliqués dans une application (ces langages peuvent être spécifiques de l'aspect ou avec une portée plus générale). Ces descriptions sont ensuite repris par un générateur de code (*weaver*) qui produit l'application finale. De point de vue de stratégie d'adaptation et de besoins de QoS, AOP vise la spécification séparée des besoins et de stratégies de gestion de QoS et la facilite de leur modification.

Pour plus d'informations le lecteur peut se référer à:

<http://www.parc.xerox.com/spl/projects/aop/>

II. The Adaptive Communication Environment (ACE)

ACE implémente un environnement fournissant des composants logiciels correspondant à des patrons de communication. Suivant une logique orientée-objet, ACE définit des classes abstraites et des moyens standards d'instanciation et d'interconnexion de ces classes. L'objectif est de faciliter le processus de création d'une application répartie en réutilisant et en spécialisant les composants prédéfinis.

Les aspects considérés par ACE sont le démultiplexage et la diffusion d'événements, la connexion et l'initialisation de service, la communication inter-processus, la gestion de mémoire partagée, la configuration dynamique des services distribués, la synchronisation et les services de plus haut niveau d'abstraction. La réalisation consiste à définir des *wrappers* qui encapsulent des mécanismes système dépendant de la plate-forme et un *framework* dans lequel ces wrappers sont intégrés et utilisés.

Parmi les classes de *wrappers* on trouve l'IPC SAP qui concerne le point d'accès à un service (socket, UNIX pipes, etc), l'initialisation de service pour laquelle sont définis un ensemble de composants de type Connector et Acceptor, les mécanismes de concurrence qui fournissent des abstractions des mécanismes de bas niveau (mutex, sémaphore, etc) comme les Objets Actifs, etc.

Dans le *framework* on trouve des réacteurs pour la diffusion des événements, des configureurs de services et des objets de gestion des flux.

Sur cette base, ACE définit toute une arborescence de composants qui peuvent être réutilisés et spécialisés et qui ont été utilisés pour définir des services de connexion distribuée et de monitoring.

Pour plus d'informations le lecteur peut se référer à:

<http://siesta.cs.wustl.edu/~schmidt/ACE.html>

III. AspectJ

AspectJ est une extension du langage Java défini par l'équipe de G.Kiczales à Xerox et mettant en oeuvre leur compréhension de la programmation par aspects (AOP).

Les aspects modélisent les fonctionnalités dont une application a besoin et qui ne sont pas captées comme des entités par la modélisation principale et la décomposition fonctionnelle qui en résulte. Ce sont des fonctionnalités comme la synchronisation entre plusieurs entités, le protocole de communication entre plusieurs entités, la gestion de la sécurité, etc. qui se retrouvent implémentés, dans les applications standards, par du code dispersé dans différents modules (malgré les exemples donnés, les aspects ne modélisent pas seulement des propriétés non-fonctionnelles). Pour éviter cette implémentation dispersée, AspectJ propose la possibilité de définir ces entités à part, dans les aspects.

Les aspects sont réalisés comme des bouts de programme pouvant influencer le comportement des objets composant une application. En référençant des points particuliers dans le code, il est possible de rajouter des entités complètes (nouvelles méthodes, nouvelles implémentations, nouvelles variables) ou des entités partielles (ajouter du comportement en début de méthode, avant une instruction particulière, etc). Les droits des aspects pour transgresser l'encapsulation des objets et effectuer des modifications sont pour l'instant universels.

Si deux aspects référencent un même point dans le programme, l'ordonnement des comportements spécifiés est fait dans l'implémentation actuelle de manière rudimentaire en utilisant des ordres partiels.

La prise en compte des aspects est faite à la compilation où un pre-processeur transforme d'abord le programme Java initial en un programme Java augmenté avec le code rajouté par les aspects est ensuite ce programme est compilé par un compilateur pouvant vérifier différentes contraintes de compatibilité. Pour changer l'ensemble d'aspects pris en compte une recompilation est nécessaire.

Pour plus de détails sur AspectJ, le lecteur peut se référer à: <http://aspectj.org/>

IV. AspectIX

AspectIX est un projet inspiré par l'idée de AOP et qui est relié à la problématique des ORBs et des applications réparties. Les réalisations dans ce cas ne restent pas seulement au niveau du langage de programmation comme pour AspectJ mais descendent d'un niveau et affectent la couche entre l'application et le système (le middleware).

Le modèle de base utilisé est celui des objets fragmentés. Dans ce modèle tout client détient une part (fragment) de l'objet distribué. La communication avec les autres fragments de l'objet est faite à l'aide de cette partie locale. Un tel fragment peut être intelligent et contenir des informations de configuration (caching, sécurité) ou alors être passif comme les stubs de CORBA.

Un fragment est composé de deux objets: un pour l'interface et un pour l'implémentation. Cette séparation permet le remplacement dynamique de l'implémentation qui est utilisée pendant l'exécution pour l'activation/désactivation des aspects. De leur cote les aspects sont définis comme des objets de configuration rattaches à un fragment donne. Leur implémentation est faite à la main en respectant une interface prédefinie.

Pour plus d'informations le lecteur peut se référer à:

<http://www4.informatik.uni-erlangen.de/Projects/AspectIX/>

V. Cherubim

Le projet Cherubim fournit une infrastructure basée agent pour permettre aux applications de définir des stratégies de protection spécifiques. L'infrastructure contient quatre composants:

- *Les agents de sécurité*: ce sont des agents mobiles encapsulant la gestion de droits, de l'authentification, de la délégation, etc. Ils peuvent être divisés en agents d'autorisation (appelés *capacités actives* et principalement considérés dans ce travail), agents de délégation et agents d'authentification.
- *L'infrastructure à agents* définit des domaines de sécurité (domaines de validité pour les agents) et les serveurs de gestion de ces domaines qui fournissent les moyens de création, modification et destruction d'agents.
- *Le responsable de l'exécution d'agents (agent runtime)* se charge de l'obtention des agents, de leur transfert, de leur vérification et finalement de leur exécution. C'est cet objet qui effectue la liaison entre les objets applicateurs et les stratégies de sécurité.
- *Les stratégies dynamiques* sont obtenues à l'aide d'une séparation entre la spécification de la stratégie de sécurité et la décision implémentée comme une fonction. L'introduction dynamique de nouvelles spécifications modifie le fonctionnement de cette fonction.

Les besoins spécifiques des applications sont intégrés en utilisant les APIs prédéfinis pour les agents de sécurité et la possibilité d'héritage et de re-implémentation de certains méthodes. De cette façon il est possible de spécifier des algorithmes de protection ou de gestion de droits.

Le projet a été intégré dans l'implémentation de CORBA JacORB. Les agents deviennent des squelettes et des talons spécialisés et le programmeur a droit d'utiliser des extensions de l'IDL pour intégrer des caractéristiques de sécurité.

Pour plus d'informations le lecteur peut se référer à:

<http://choices.cs.uiuc.edu/Security/cherubim/>

VI. Coda

Les projets Coda et Odyssey explorent les problèmes des applications s'exécutant sur des ordinateurs mobiles et devant accéder à des informations disponibles sur le réseau. Coda (Odyssey est considéré un peu plus loin dans la liste des projets) se concentre sur la réalisation d'une adaptation transparente pour l'application dans le cadre de l'utilisation d'un système de fichiers. Pour assurer l'accès aux fichiers et l'utilisation de l'information de manière cohérente, les travaux proposent des solutions concernant les opérations déconnectées, la réplication optimiste, la connectivité faible (cas de réseaux chers ou de largeur de bande très petite), les transactions d'isolation seulement (I de ACID) et la réplication de serveurs pour compléter les opérations déconnectées. Ce système est en utilisation effective depuis quelques années.

Pour plus d'informations le lecteur peut se référer à:

<http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html>

VII. COMERA

COMERA est une extension de l'infrastructure COM de Microsoft qui vise la flexibilité et la possibilité de personnalisation en fonction des besoins des applications. En utilisant le principe de structuration par composants et en allant encore plus loin et encapsulant des caractéristiques qui dans COM ne sont pas des composants, COMERA donne la possibilité aux applications de choisir les composants à utiliser sans avoir à reconstruire tout le système. Les exemples implémentés sont les *canaux multi-connexion configurables* où de manière transparente il est possible de communiquer avec plusieurs entités pour assurer la performance ou la tolérance aux pannes, le *choix de transport* qui permet l'exécution de DCOM sur n'importe quelle couche de transport, la *migration d'objets* qui assure la connexion entre un client et un objet qui a migré sur une machine différente.

Pour plus d'informations le lecteur peut se référer à:

<http://www.research.microsoft.com/~ymwang/papers/HTML/COMERA/F.htm>

VIII. DART (Distributed Adaptive Run-Time)

DART [RL98] fait partie des projets utilisant la réflexion pour réaliser l'adaptabilité. Les adaptations considérées sont le changement de l'implémentation d'un objet pendant l'exécution et l'extension dynamique d'un objet avec des propriétés générales. Elle sont réalisées dans DART en fournissant des moyens de réification des invocations de méthodes qui peuvent être respectivement des méthodes *reflectives* ou des méthodes *adaptables*.

Les méthodes reflectives permettent la personnalisation de l'environnement d'exécution de l'objet. Le niveau "méta" contient dans ce cas des objets gérant des services indépendamment de l'application. Quand un objet est appelé avec une méthode reflective, l'appel est redirigé vers ces méta-objets. DART propose plusieurs méta-objets de ce genre dont l'objet de réplication ou celui de persistance.

Les méthodes adaptables permettent aux programmeurs de gérer plusieurs implémentations et de choisir entre elles pendant l'exécution.

DART définit des mécanismes générales pour l'adaptabilité et notamment les méthodes reflectives, les méthodes adaptables, la possibilité de gérer des événements, la gestion de stratégies d'adaptation en réaction à ces événements, etc. Cependant, l'environnement ne se préoccupe pas des facilités et des moyens effectifs de construction de nouveaux méta-objets ou stratégies et nous restons avec l'impression que les différents méta-objets sont définis de manière ad hoc.

IX. dynamicTAO

DynamicTAO vise la reconfiguration dynamique de middleware. DynamicTAO est réalisé en prenant comme base TAO, un middleware réalisé pour le domaine d'avions et dans lequel toute adaptation est faite statiquement, et en l'adaptant pour supporter la réflexivité. La réflexion permet la réification dynamique et le changement des différentes *stratégies* sans relancer l'ORB. Les stratégies sont en effet des encapsulations d'aspects internes de l'ORB comme la concurrence, la gestion des invocations, etc.

La réification est faite à l'aide d'entités appelées des configurateurs. Ces configurateurs contiennent les dépendances entre certains composants du système et disposent des points d'attachement des implémentations des différentes stratégies. Etant présentées comme des bibliothèques pouvant être chargées dynamiquement, elle peuvent être reliées pendant l'exécution à l'ORB.

Une interface IDL est disponible pour effectuer des reconfigurations. Elle contient des méthodes d'inspection et des méthodes de configuration concernant le chargement d'un composant, l'effacement d'une implémentation, etc. DynamicTAO prévoit l'utilisation des aspects de sécurité de CORBA et rajoute la possibilité d'exécution de code Java pour empêcher une reconfiguration non autorisée ou non stable.

Pour plus d'informations le lecteur peut se référer à:

<http://choices.cs.uiuc.edu/2k/dynamicTAO/>

X. Entreprise JavaBeans (EJB)

Les EJB définissent une extension du modèle à composants des JavaBeans et visent la construction de services côté serveur. Aucune implémentation n'est imposée, seulement les interfaces des différentes entités et leurs relations sont définies.

Un composant EJB s'exécute dans un conteneur qui lui-même s'exécute sur un serveur EJB. Le conteneur est responsable de fournir les services dont un composant a besoin et c'est lui et non l'EJB applicatif qui prend en compte la spécificité de la plate-forme. Ceci permet le déploiement du même bean sur de différents serveurs.

Le modèle se concentre principalement sur les aspects de persistance et de sécurité. En effet, le serveur d'EJBs doit fournir un ensemble standard de services dont un service de gestion de transactions distribuées. Il doit également prévoir un conteneur d'EJBs qui de son côté gère le cycle de vie des composants, contrôle les transactions de manière implicite, gère la persistance et les services de distribution (l'interface entre serveur et conteneur n'étant pas spécifiée, les deux entités sont souvent confondues). Tous ces services sont utilisés par les composants, certains de manière complètement implicite. L'avantage de cette approche est que le travail des programmeurs est fortement diminué vu qu'ils s'occupent seulement des aspects fonctionnels. Ils peuvent paramétrer les services de persistance et de sécurité (sans changer le code fonctionnel) en utilisant des déclarations qui sont prises en compte au moment du déploiement.

Pour plus d'informations le lecteur peut se référer à: <http://java.sun.com/products/ejb/>

XI. Experiments with Reflective Middleware

Ce projet fait partie des travaux qui appliquent la réflexivité dans le domaine du middleware dans le but de fournir une facilité de configuration et d'adaptation.

La réflexion est utilisée pour exposer certains points de l'implémentation du système. En effet à chaque objet est associé un méta-espace composé de trois méta-modèles: d'encapsulation, de composition et d'environnement. *Le modèle de composition* fournit l'accès aux composants d'un objet sous forme de graphe. Dans ce graphe les objets sont connectés à l'aide de connexions locales (*local binding*) qui permettent de relier les interfaces des objets de manière indépendante du langage de programmation. Le graphe peut être inspecté et modifié. Le deuxième modèle, *d'encapsulation*, représente les interfaces en termes de leurs méthodes, leurs attributs et d'autres caractéristiques principales comme les informations de héritage. Le dernier modèle qui est celui *d'environnement* représente l'environnement d'exécution et est lié

à des aspects comme la gestion de messages (files, diffusion) et des processus (création, ordonnancement, etc.).

Pour plus d'informations le lecteur peut se référer à:

<http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/index.html>

XII. Globe

Globe vise à fournir un middleware supportant un grand nombre d'utilisateurs (*scalability*) et permettant l'implémentation de services spécifiques aux besoins des applications. Pour cela le système définit un modèle à objets où les objets sont physiquement distribués. L'implémentation des interfaces d'un tel objet sur une machine donnée forme un objet local. Un objet local est composé de quatre sous-objets: *objet sémantique* qui implémente les fonctionnalités de l'objet, *objet de réplication* qui contient l'implémentation d'une stratégie de réplication, *objet de communication* qui gère la communication entre les objets locaux et *objet de contrôle* qui régule les interactions entre les quatre sous-objets. Globe fait utilisation d'un IDL similaire à celui de CORBA qui permet de faciliter l'écriture des sous-objets. Pour l'instant l'objet de contrôle est à écrire à la main et un langage de définition d'objets (ODL) est en cours de spécification.

Pour plus d'informations le lecteur peut se référer à: <http://www.cs.vu.nl/~steen/globe/>

XIII. Iguana

Iguana est une extension de C++ qui permet la réification de caractéristiques du langage et la modification dynamique de leur implémentation.

Les méta-protocoles (MOPs = Meta-level Object Protocols) spécifient quels sont les méta-objets nécessaires pour implémenter le modèle à objets supporté par un langage. Iguana associe les méta-protocoles aux objets ce qui permet aux différents objets dans un programme d'utiliser des modèles différents. Le langage prévoit les moyens de définition de MOPs et les mécanismes d'association entre des MOPs et des objets concrets. Le projet identifie également un ensemble de catégories réifiables et notamment ce sont les classes, les méthodes, la création d'objets, la destruction d'objets, l'invocation de méthodes, l'héritage, etc.

Iguana n'est pas interprété et pour alléger le coût d'une compilation les MOPs et l'information associée sont générés seulement pour les objets qui ont explicitement choisi de les utiliser. De la même manière, les catégories de réification évitent l'évaluation de tous les mécanismes en spécifiant quels mécanismes sont à réifier.

Pour plus d'informations le lecteur peut se référer à: <http://dedanann.dsg.cs.tcd.ie/~coyote/>

XIV. Jonathan

Jonathan a pour but de définir un ORB flexible et adaptable qui supporte l'introduction de divers mécanismes de communication entre des objets dans un environnement distribué. Ce travail a été inspiré par des projets comme Subcontract. L'ORB se concentre sur la connexion où sous connexion on entend non seulement le processus d'interconnexion d'objets en suivant une certaine sémantique mais aussi le résultat de ce processus. Jonathan définit un ORB minimal qui fournit un environnement générique qui peut être utilisé avec différentes "fabriques de connexions" (*binding factories*) qui créent et gèrent des connexions différents (*binding objects*). Le réalisation concerne principalement la gestion de références distribués et des types. Cet ORB a été utilisé pour implémenter les "personnalités" CORBA et JAVA-RMI. Ces implé-

mentations sont faites statiquement et le choix pour les applications se fait pendant la construction.

Pour plus d'informations le lecteur peut se référer à:

<http://www.infres.enst.fr/~delpiano/CNET/Jonathan-1.4/doc/Jonathan.html>

XV. Khazana

Khazana offre l'abstraction d'état partagé et distribué. Ce projet implémente une mémoire globale où les applications ont le droit d'allouer de la place, d'écrire ou lire des données, etc. Khazana se charge de la gestion d'aspects comme la réplication, la gestion de la cohérence, la tolérance aux pannes, le contrôle d'accès et la gestion de la location. Les primitives fournies par ce service (*reserve/unreserve*, *allocate/free*, *lock/unlock*, *read/write*, *get/set*) sont de granularité fine et il n'y a pas d'entités de plus haut niveau d'abstraction comme les transactions, les objets ou les flux implémentées dans d'autres systèmes distribués. Les auteurs ont utilisé le système pour l'implémentation d'un système de fichiers distribué et planifient une implémentation efficace d'une couche de gestion d'objets distribués à l'exécution (comme CORBA ou DCOM).

Pour plus d'informations le lecteur peut se référer à: <http://www.cs.utah.edu/khazana/>

XVI. Ninja/MultiSpace

Le projet réalise un environnement dédié à la construction et à l'exécution de services hautement disponibles et en même temps flexibles et adaptables. L'idée principale est de fournir un environnement distribué particulier, une grappe (*cluster*), qui est facilement contrôlable par la possibilité de migrer et d'installer du code sur des différents noeuds de cet environnement et par la génération dynamique de niveaux d'indirection entre les clients et les services. Devant le monde extérieur un tel environnement a l'apparence d'un service monolithique, fiable et performant. Tous les problèmes de tolérance aux pannes, cohérence et disponibilité sont résolus à l'intérieur de cet environnement. Les applications construites au-dessus de cet environnement ont été le JudeBox donnant accès à une collection distribuée de disques et Kereitsu réalisant un service de messagerie instantanée entre des dispositifs hétérogènes.

Pour plus d'informations le lecteur peut se référer à: <http://ninja.cs.berkeley.edu/>

XVII. Odyssey

Odyssey, le successeur de Coda, continue les efforts dans le domaine d'applications mobiles accédant à des informations. Contrairement à Coda, Odyssey essaie de prendre en compte les besoins spécifiques des applications et de réaliser l'adaptation en conséquence ("*application-aware*"). Cet aspect est reflété par l'effort d'Odyssey de gérer les types de données manipulées par les différentes applications. Il prévoit un composant générique (le "vice-roi") qui est responsable de la gestion de modules spécifiques encapsulant le support système pour les traitements des différents type de données. Les auteurs ont exploré l'utilité de leur système avec plusieurs types d'applications dont un Video Player, Netscape Navigator et un Speech Recogniser.

Pour plus d'informations le lecteur peut se référer à:

<http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html>

XVIII. ParcTab

La problématique considérée dans ce projet traite le cas “d’applications mobiles” où il s’agit de **mobilité de l’utilisateur**. Le type d’adaptation est appelé “sensible au contexte” (*context-aware*). Sous contexte on entend un ensemble d’aspects comme la location, l’entourage en personnes, les ressources disponibles, etc. Pour toutes expérimentations un petit dispositif appelé ParcTab est utilisé. Il peut être tenu dans la main et utilise les rayons infra-rouges pour communiquer avec le réseau. Les applications développées concernent: *la sélection en fonction de la proximité* visant à faciliter le choix d’objets se trouvant plus près de l’utilisateur, il peut s’agir de choisir les dispositifs d’entrée/sortie; *la visualisation d’informations en fonction du contexte* avec comme exemple des informations concernant l’organisation globale d’un laboratoire si on se trouve dans son hall et l’agenda d’une personne si on se trouve dans son bureau; *les actions contextuelles* qui sont spécifiées à l’aide de règles SI-ALORS.

Pour plus d’informations le lecteur peut se référer à: <http://sandbox.parc.xerox.com/parctab/>

XIX. Prayer

Prayer s’inscrit parmi les travaux traitant la notion de QoS de point de vue réseau et gestion des ressources. Cet environnement inclue une infrastructure d’adaptation dynamique pour des applications mobiles qui fournit des mécanismes de négociation de service, de surveillance et de notification. Les applications spécifient leurs besoins en utilisant des APIs fournies par la couche de Prayer. Elles définissent des *classes de QoS* qui contiennent des informations comme les limites pour le débit, le délai, etc. Elle définit également des *blocs d’adaptation* qui sont des triplets contenant une classe de QoS, la procédure spécifiant la séquence d’exécution si cette classe est vérifiée et l’action (les actions sont prédéfinies) à prendre si elle est violée.

Pour plus d’informations le lecteur peut se référer à: <http://chaos.crhc.uiuc.edu/~bharghav/>

XX. Quality Definition Language (QDL)

QDL est un langage orienté vers la spécification de la QoS de l’infrastructure QuO. Le langage QDL est spécialement conçu pour décrire les besoins d’applications réparties. Il est prévu pour une utilisation avec CORBA. Il est composé de trois sous-langages: le langage de description de contrats CDL (Contract Description Language), le langage de description de structures SDL (Structure Description Language) et le langage de description de ressources RDL (Resource Description Language). Les deux derniers (SDL et RDL) sont encore en cours de développement.

CDL est utilisé pour décrire le contrat de QoS entre un client et un objet ce qui inclue la QoS que le client demande de l’objet, la QoS que l’objet pense fournir, les régions des possibles niveaux de QoS, les paramètres système (appelés conditions, elles peuvent être des encapsulations des paramètres système de plus bas niveau, prédéfinies ou définies et programmées par les constructeurs de l’application) à surveiller et finalement les actions à entreprendre lors d’un changement des besoins du client, des besoins de l’objet ou de la QoS.

Les facilités proposés par QDL sont clairement orientés vers une spécification la stratégie d’adaptation ce qui n’était pas le cas avec QML. Cependant, même si les contrats sont spécifiés séparément du code fonctionnel, il font références à ce code et peuvent inclure l’utilisation de variables définies par l’application particulière ou l’invocation de méthodes des objets implicites. Ceci pose des questions de réutilisation. D’autre part ces contrats supposent la

connaissance préalable de tous les états dans lesquels peut se trouver l'application répartie. Une telle énumération est difficile à faire.

Pour plus d'informations le lecteur peut se référer à:

<http://www.dist-systems.bbn.com/tech/QuO/>

XXI. Quality Modeling Language (QML)

Développé dans Hewlett-Packard, QML est implémenté comme une extension de UML centrée sur l'expression des besoins de QoS.

QML a trois abstractions principales: le *type de contrat*, le *contrat* et le *profile*. QML permet la définition de types de contrat qui représentent des aspects de QoS spécifiques comme la performance ou la fiabilité. Un type de contrat définit des dimensions qui peuvent être utilisées pour caractériser un aspect de QoS particulier. Une dimension a un domaine de valeurs qui peuvent être ordonnées. Les domaines peuvent être de trois types: les ensembles, les domaines énumérés et les domaines numériques. Un contrat est une instance d'un type de contrat et représente une spécification particulière de la QoS. Enfin, les profiles associent les contrats avec des interfaces, des opérations, les arguments et les résultats d'opérations.

A part les moyens de spécification, QML dispose d'un environnement se chargeant de la représentation de ces spécifications pendant l'exécution et permettant l'échange de telles spécifications entre les entités d'une application répartie. Cet échange sous-entend la possibilité de négociation entre les différentes entités et l'implication de mécanismes d'adaptation mais ce n'est pas un domaine d'intérêt pour les auteurs. QML reste à un niveau d'abstraction assez haut sans se préoccuper des mécanismes de mise en oeuvre de ces spécifications et des adaptations nécessaires.

Pour plus d'informations le lecteur peut se référer à:

<http://www.hpl.hp.com/techreports/98/HPL-98-10.html>

<http://www.usenix.org/publications/library/proceedings/coots99/frolund.html>

XXII. Quality Objects (QuO)

Le projet QuO a pour but de définir une infrastructure permettant la spécification et la gestion de la QoS dans des applications réparties composées d'objets distribués (surtout d'objets CORBA). Dans l'architecture QuO, les interfaces fonctionnels des objets distribués sont augmentés avec des interfaces dédiés à la gestion de la QoS. Dans la logique des QuO, le fonctionnement d'une application peut se trouver dans de différentes régions de QoS. L'identification de ces régions et des différentes transitions permet de choisir dynamiquement l'action d'adaptation. Les actions peuvent être des actions prédéfinies et se trouvant au niveau des mécanismes système (ex. négociation de ressources avec RSVP) ou des actions définies par l'application. Les QuO définissent un langage de spécification de la QoS et de la stratégie d'adaptation pour l'application et intègrent ces spécifications comme des entités exécutables dans la structure de CORBA.

Les QuO sont utilisés comme une partie intégrante dans plusieurs projets de recherche et notamment: *AQuA* (Adaptive Quality of Service for Availability) qui, comme son nom l'indique, se concentre sur les problèmes de disponibilité (prédiction, spécification, garantie), *DIRM* (Dynamic Integrated Resource Management) qui vise de donner aux applications le contrôle sur leurs communications au-dessus de RSVP et l'effort *Quorum* (Adaptive QoS in Dynamic Distributed Computing Environments) qui a pour objectif l'intégration de tous les

travaux autour de la QoS: compréhension et formalisation de la notion, gestion de la QoS dans de différentes situations et à différents niveaux dans le système, les garanties de QoS, etc.

Pour plus d'informations le lecteur peut se référer à:

<http://www.dist-systems.bbn.com/tech/QuO/>

<http://www.dist-systems.bbn.com/projects/AQuA/>

<http://www.dist-systems.bbn.com/projects/DIRM/>

<http://www.darpa.mil/ito/research/quorum/>

XXIII. Subcontract

Subcontract [HPM93] traite le problème de l'introduction de nouveaux mécanismes de communication entre les objets dans un système distribué. Il fait partie de l'environnement appelé Spring qui fournit un système d'exploitation réparti. Le système définit un modèle à objets dans lequel le client agit directement sur un objet. Cependant l'état de cet objet peut être stocké sur un site distant. Les sous-contrats sont des couches qui se rajoutent lors de la communication entre les objets clients et serveurs. Ces sous-contrats sont définis statiquement et sont stockés dans des bibliothèques. Les exemples de sous-contrats implémentés par les auteurs sont le *replicon* (qui supporte la réplication), le *simplex* (qui donne un mode de fonctionnement très simple entre le client et le serveur), le *cache* (gère des techniques de cache), etc.

XXIV. Sumatra

Une extension de l'environnement de programmation Java, Sumatra se concentre sur les mécanismes permettant la construction d'applications mobiles adaptables. Le mécanisme principal considéré est la migration de groupes d'objets entre deux machines d'exécution. La décision sur qui, quoi et quand migrer est laissée entièrement à l'application. Sumatra prévoit une interface pour la surveillance de ressources qui peut être utilisée par les applications lors de cette décision. Un outil (Komodo) de surveillance a été également construit pour l'estimation de la latence du réseau. Une des applications qui ont été implémentées pour cette expérience est l'Adaptalk qui vise l'optimisation du temps de réponse dans l'application de conversation par ordinateur (*talk*).

Pour plus d'informations le lecteur peut se référer à:

<http://www.antd.nist.gov/antd-staff/mranga/mranga.html>

Bibliographie

- [BG97] V. Bharghavan, V. Gupta
“A Framework for Application Adaptation in Mobile Computing Environments.”
Computer Software and Applications Conference '97, Bethesda, MD.
August 1997.
<http://www.timely.crhc.uiuc.edu/publications.html>
- [Bla97] G. Blair
“The Case of Reflective Middleware”
Proc. 3rd Cabernet Plenary Workshop, Rennes, France, April 1997.
<http://www.newcastle.research.ec.org/cabernet/workshops/3rd-plenary-papers/>
- [Cza98] K. Czarnecki PhD. Thesis Technische Universität Ilmenau, Germany 1998
“Aspect-Oriented Decomposition and Composition”
<http://www.prakinf.tu-ilmenau.de:80/~czarn/aop/>
- [CBC98] F. Costa, G. Blair, G. Coulson
“Experiments with Reflective Middleware”
ECOOP Workshop on Reflective Object-Oriented Programming and Systems (ROOPS'98), Brussels, Springer-Verlag, 1998
<http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/publ.html>
- [CRS98] J. Carter, A. Ranganathan, S. Susarla
“Khazana: An Infrastructure for Building Distributed Services”
Proceedings of the 18th Annual International Conference on Distributed Computing Systems, pp. 562--571, May 1998
<http://www.cs.utah.edu/khazana>
- [DHT+98] B. Dumant, F. Horn, F. Dang Tran, J.-B. Stefani
“Jonathan : an Open Distributed Processing Environment in Java”
Middleware'98 : IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Sept. 1998
- [DTH+98] B. Dumant, F. Dang Tran, F. Horn, J.-B. Stefani.
“Jonathan: an open distributed processing environment in java.”
Middleware'98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing,
The Lake District, U.K., September 1998
<http://www.infres.enst.fr/~delpiano/Jonathan.htm>
- [Flo94] P. Florissi
“QuAL: Quality Assurance Language”, Thesis proposal
Columbia University, March 1994
<http://www.cs.columbia.edu/~pgsf/qual.html>
- [FK98] S. Frolund, J. Koistinen
“Quality of Service Specification in Distributed Object Systems Design”
Proceedings of the 4th Conference on Object-Oriented Technologies and Systems (COOTS), April 1998
- [FK99] S. Frolund, J. Koistinen
“Quality of Service Aware Distributed Object Systems”

-
- Proceedings of the 5th Conference on Object-Oriented Technologies and Systems (COOTS), Mai 1999
- [GC96] B.Gowing , V.Gahill
“Meta-object protocols for C++ : The Iguana Approach”
Proceedings of Reflexion’96, April 1996, p.137-152
<http://dedanann.dsg.cs.tcd.ie/~coyote/>
- [GGM97] J-M.Geib, C. Gransart, Ph.Merle.
“CORBA: des concepts à la pratique”
EDITIONS MASSON, Paris, France, October 1997, ISBN: 2225830460
De l’information peut également être trouvée sur <http://www.omg.org>
- [GGW+99] I.Goldberg, S.Gribble, D.Wagner, E.Brewer
“The Ninja Judebox”
prévu pour publication dans 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, October 1999
<http://ninja.cs.berkeley.edu/pubs/pubs.html>
- [GWB+99] S.Gribble, M.Welsh, E.Brewer, D.Culler
“The MultiSpace: an Evolutionary Platform for Infrastructural Services”
Proceedings of the 1999 Usenix Annual Technical Conference,
Monterey, CA, June 1999
<http://ninja.cs.berkeley.edu/pubs/pubs.html>
- [HBG+98] F.Hauck, U.Becher, M.Geier, E.Meier, U.Rastofer, M.Steckermeier
"AspectIX : An Aspect-Oriented and CORBA-Compliant ORB Architecture"
Middleware’98 IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing,
Proceedings edited by Nigel Davies, Kerry Raymond and Jochen Seitz
<http://www4.informatik.uni-erlangen.de/Projects/AspectIX/>
- [HL98] D. Hagimont, D. Louvegnies
“Javanaise: Distributed Shared Objects for Internet Cooperative Applications”,
Proceedings IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware’98),
The Lake District, 15-18 September 1998
- [HNN+99] A.T.van Handersen, A.Noutash, L.J.M.Nieuwenhuis, M.Wegdam
“Extending CORBA with Specialised Protocols for QoS Provisionning”
International Symposium on Distributed Objects and Applications (DOA’99)
http://amidst.ctit.utwente.nl/amidst_publications.html
- [HPM93] G. Hamilton, M.Powell, J.Mitchell
“Subcontract : A Flexible Base for Distributed Programming”
Sun Microsystems Laboratories, Inc, Technical Raport SMLI TR-93-13
<http://www.sun.com/research/techrep/1993/abstract-13.html>
- [IETF] IETF RFC 2205
Resource ReSerVation Protocol (RSVP), Functional Specification
<http://www.ietf.org/html.charters/rsvp-charter.html>
- [ITU] ITU-T Rec. X 901 | ISO/IEC 10746
Reference Model - Open Distributed Proceession (RM-ODP)
<http://www.dstc.edu.au/Research/Projects/ODP/standards.html>
-

-
- [ISO] QoS Framework -ISO/IEC 13236
<http://wwwhome.ctit.utwente.nl/~tuquerre/Standards.html>
- [KCS98] R.Kravets, K.Calvert, K.Schwan
“Payoff Adaptation of Communication for Distributed Interactive Applications”
Journal on High Speed Networking: Special Issue on Multimedia Communications, 1998.
<http://www.cc.gatech.edu/systems/projects/adaptive.html#Papers>
- [KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Christina Lopes, Jean-Marc Loingtier, John Irwin
"Aspect Oriented Programming"
Proceedings of ECOOP, Finland, Springer-Verlag LNCS 1241, Juin 1997
<http://www.parc.xerox.com/spl/groups/eca/pubs/by-topic.html#AOP>
- [LK98] C. Lopes, G.Kiczales
“Recent Developments in AspectJ”
ECOOP’98 Workshop Reader, Springer-Verlag 1543
<http://aspectj.org>
- [MGN91] M.Makpangou, Y.Gouhant, J-P.Narzul
“Fragmented Objects for Distributed Abstractions”
Readings in Distributed Computing Systems, Casavant and M. Singhal, ed.,
IEEE Computer Society Press, 1994
<http://www-sor.inria.fr/publi/>
- [MLM+97] C.Maeda, A.Lee, G.Murphy, G.Kiczales
“Open Implementation Analysis and Design”
<http://www.parc.xerox.com/spl/projects/oi-at-parc/methodology.html>
- [NSN+97] B. Noble, M. Satyanarayanan, D. Narayanan et all.
“Agile Application-Aware Adaptation for Mobility”
16th ACM Symposium in Operating System Principles, 1997
<http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html>
- [OSF] Open Software Foundation, Distributed Computing Environment
<http://www.opengroup.org/dce/>
- [Qui97] T.Quin
“An Agent-Based Security Architecture for Supporting Application Aware Security”, Workshop Paper
Workshop on Research Directions for the Next Generation Internet,
Vienna, VA, May 1997
<http://choices.cs.uiuc.edu/Security/cherubim/papers/>
- [Qui98] T.Quin
“Cherubim Agent Based Dynamic Security Architecture”, Technical Report
University of Illinois at Urbana-Champaign, June, 1998
<http://choices.cs.uiuc.edu/Security/cherubim/papers/>
- [RAS+97] M.Ranganathan, A.Acharya, S.Sharma, J.Saltz
"Network-aware Mobile Programs"
USENIX, 1997 Annual Technical Conference, p.91-103
<http://usenix.org/publications/library/index.html>

-
- [RKC99] M.Roman, F.Kon, R.Campbell
“Supporting Dynamic Reconfiguration in the dynamicTAO Reflective ORB”
Technical Report UIUCDCS-R-98-2085, Department of Computer Science,
University of Illinois at Urbana-Champaign. February, 1999
<http://choices.cs.uiuc.edu/2k/dynamicTAO/>
- [RL98] P-G.Raverdy, R. Lea
“DART : A Distributed Adaptive Run-time”
Work in progress session, Middleware’98: IFIP International Conference
on Distributed Systems Platforms and Open Distributed Processing,
The Lake District, U.K., September 1998
<http://www-sor.inria.fr/mirrors/middleware98/programme.html>
- [Sat96] M. Satyanarayanan
“Mobile Information Access”
IEEE Personal Communications, February 1996
<http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html>
- [Sch93] D.Schmidt
“The ADAPTIVE Communication Environment: an Object-Oriented Network
Programming Toolkit for Developing Communication Software”
11th and 12th Sun Users Group Conference, December 1993 and June 1994.
<http://siesta.cs.wustl.edu/~schmidt/ACE-papers.html>
- [Siq98] F.Siqueira
“The Design of a Generic QoS Architecture for Open Systems”
Proceedings of the 19th IEEE Real Time Systems Symposium (RTSS’98),
Work in Progress Session, Madrid, Spain, December 1998.
<http://www.cs.tcd.ie/Frank.Siqueira/publicat.html>
- [Sun-ejb] Enterprise JavaBeans, Sun Microsystems
<http://java.sun.com/products/ejb/newspec.html>
- [Sun-rmi] Remote Method Invocation Specification, Sun Microsystems 1997
<http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html>
- [SAW94] B.Schilit, N.Adams, R.Want
“Context-Aware Computing Applications”
Proceedings of the Workshop on Mobile Computing Systems and Applications,
Santa Cruz, CA, December 1994. IEEE Computer Society
<ftp://ftp.parc.xerox.com/pub/schilit/wmc-94-schilit.ps>
- [SES+96] M.Seltzer, Y.Endo, C.Small, K.Smith
“Issues in Extensible Operating Systems”
WCSS’96 (First Annual Workshop on Compiler Support for System Software)
<http://www.eecs.harvard.edu/~chris/papers>
- [STK+99] M. van Steen, A.S. Tanenbaum, I. Kuz, and H.J. Sips
"A Scalable Middleware Solution for Advanced Wide-Area Web Services."
Distributed Systems Engineering, vol. 6(1), March 1999, pp.34-42
<http://www.cs.vu.nl/~steen/globe/>
- [UML] Unified Modeling Language Summary, Version 1.1, September 1997
<http://www.rational.com/uml>
-

-
- [VZL+98] R. Vanegas, J.A. Zinky, J.P. Loyall, D.A. Karr, R.E. Schantz, D.E. Bakken
"QuO's Runtime Support for Quality of Service in Distributed Objects"
Middleware'98: Proceedings of the IFIP International Conference on
Distributed Systems Platforms and Open Distributed Processing
The Lake District, U.K., September 1998
<http://www.dist-systems.bbn.com/papers/>
- [YCE+97] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler
"Using Smart Clients to Build Scalable Services"
USENIX Annual Technical Conference 1997
<http://usenix.org/publications/library/index.html>