Adaptabilité des Services Techniques dans le Modèle à Composants

Colombe Hérault

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

LAMIH

UVHC
UNIVERSITE DE VALENCIENNES ET DU HAINAUT - CAMBRESIS

THÈSE

présentée et soutenue publiquement le 15 juin 2005

pour l'obtention du

Doctorat de l'Université de Valenciennes et du Hainaut Cambrésis

Discipline: Informatique

par

Colombe HERAULT

Adaptabilité des Services Techniques dans le Modèle à Composants

Composition du jury

Président : Pr René Mandiau, LAMIH, Université de Valenciennes et du Hainaut Cambrésis

Rapporteurs : Pr Laurence Duchien, LIFL, Université de Lille I
Pr Pierre Paradinas, CEDRIC, CNAM Paris

Examinateurs : Pr Arnaud Fréville, LAMIH, Université de Valenciennes et du Hainaut Cambrésis, directeur de thèse
MdC Nadia Bennani, LAMIH, Université de Valenciennes et du Hainaut Cambrésis, encadrant de thèse
Dr Thierry Coupaye, France Télécom R&D
MdC Didier Donsez, LSR, Université Joseph Fourier de Grenoble
MdC Sylvain Lecomte, LAMIH, Université de Valenciennes et du Hainaut Cambrésis, encadrant de thèse

Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines — UMR CNRS 8530

Remerciements

Je tiens à remercier Arnaud Freville de m'avoir accueilli au sein de l'équipe ROI du LAMIH et d'avoir accepté d'être mon directeur de thèse.

Je tiens à remercier particulièrement Nadia Bennani et Sylvain Lecomte, qui ont encadré mon travail au cours des quatre dernières années. Ils se sont montrés très complémentaires et ont su m'apporter conseils, aide et encouragements tout en gardant un œil critique sur mon travail afin de mieux faire progresser nos idées.

Je remercie énormément Laurence Duchien et Pierre Paradinas pour m'avoir fait l'honneur de rapporter mon travail. Leurs remarques et questions m'ont permis d'améliorer la qualité de ce manuscrit.

Je tiens à remercier René Mandiau, Thierry Coupaye et Didier Donsez d'avoir accepté de participer à mon Jury. Une mention spéciale pour Didier car son enthousiasme, parfois débordant, m'a donné la motivation pour entamer une année de DEA.

Je remercie évidemment tous les membre de l'équipe ROI et tout particulièrement ceux de SID : Hocine, Marie, Sergiy, Thierry. Grâce à eux, mon travail de recherche fût fructueux et mes pauses thé nombreuses et joyeuses.

Le prototype présenté dans ce manuscrit est l'œuvre collective de Hocine Grine, Philippe DosSantos, Décamps, Roux et Wemelbeke et moi-même. J'ai aussi reçu l'aide amicale de Samuel Marcaille. Je les remercie pour leur contribution.

Merci aux thésards de l'équipe ROI, aux copains de la fac et à Yohan pour les moments de détente qu'ils m'ont offert. Merci à mes amis, spécialement Laurette sur qui on peut toujours compter, et à ma famille qui m'ont soutenue et reconfortée. Je tiens à remercier mes parents, qui m'ont donné l'envie d'apprendre et le curiosité intellectuelle. Merci à Maman et à Jean-Claude d'avoir "rerelu ma taise". Merci à Alain qui chaque jour m'a donné la force et l'envie de continuer.

"Etudiant poil aux dents..."
Renaud Séchan

Table des matières

Liste des tableaux

Table des figures

Introduction

1 Contexte

L'évolution des systèmes d'information se caractérise par deux grands phénomènes : la multiplication et la diversification des informations et des services disponibles ainsi que l'hétérogénéité des terminaux participant aux échanges de ces données et de ces services autour de réseaux communs [BAN 02]. Les applications peuvent varier de grands systèmes d'information pour les entreprises à des applications de consultation de contenu Web sur un téléphone portable. Les machines sur lesquelles ces applications s'exécutent peuvent être de puissants serveurs d'information, des ordinateurs personnels, des assistants personnels digitaux (PDA), des téléphones portables ou encore des cartes à puce, interconnectés grâce à des réseaux filaires, WIFI, satellitaires, etc. Dans ce contexte, les applications se complexifient. Il est donc nécessaire de réduire les temps de développement et de mise à jour des services distribués sur les réseaux et provenant de fournisseurs différents, tout en garantissant une bonne interopérabilité de services.
C'est la promesse faite par la programmation à base de composants. Cette approche est largement usitée dans le domaine des applications distribuées, notamment grâce aux modèles à composants EJB et CCM. Elle permet une meilleure analyse de la structure des applications complexes et délimite les différentes étapes de leur développement qui peuvent ainsi être confiées à des spécialistes. Les applications sont plus rapidement développées et plus robustes.
Les modèles à composants dits "métiers", introduisent la notion de services techniques. Ces services ont la particularité d'être utilisés par l'ensemble des applications d'une plateforme pour leur apporter une meilleure qualité de service mais ne font pas partie, à proprement parler, de la logique applicative. Par exemple, les services de persistance, transaction, sécurité, nommage sont des services techniques. Les services techniques, développés par des spécialistes du domaine, ont pour avantage d'être sûrs et optimisés. Ces services sont rendus de façon transparente au développeur, grâce à l'encapsulation du code applicatif dans un conteneur. Ce conteneur repose sur un bus logiciel qui fournit les services techniques. Les services techniques facilitent le travail du développeur d'application car celui-ci bénéficie de ces services, sans pour autant devoir les gérer, et ils limitent l'entrelacement du code, en assurant une meilleure évolutivité. Cependant, les services techniques sont programmés en vue d'être exécutés dans un contexte assez spécifique : sur un serveur d'application, dans un réseau fixe. Ils ne sont donc pas forcement adaptés aux nouveaux terminaux et aux nouveaux réseaux.

2 Travaux préliminaires

Notre travail a débuté sur cette constatation, dans le cadre du projet COMPiTV (RNRT - 2001/2003). Ce projet, depuis terminé, avait pour but l'évaluation de la mise en place d'architectures à base de composants dans le cadre de services interactifs sur la chaîne d'exploitation de télévision numérique ([CHI 02], [COMb]). Il regroupait l'université de Lille 1 (LIFL/GOAL), l'université de Valenciennes (LAMIH/ROI), Canal+ Technologie et Gemplus Card International.

La télévision numérique permet d'associer des logiciels interactifs à la diffusion d'émissions

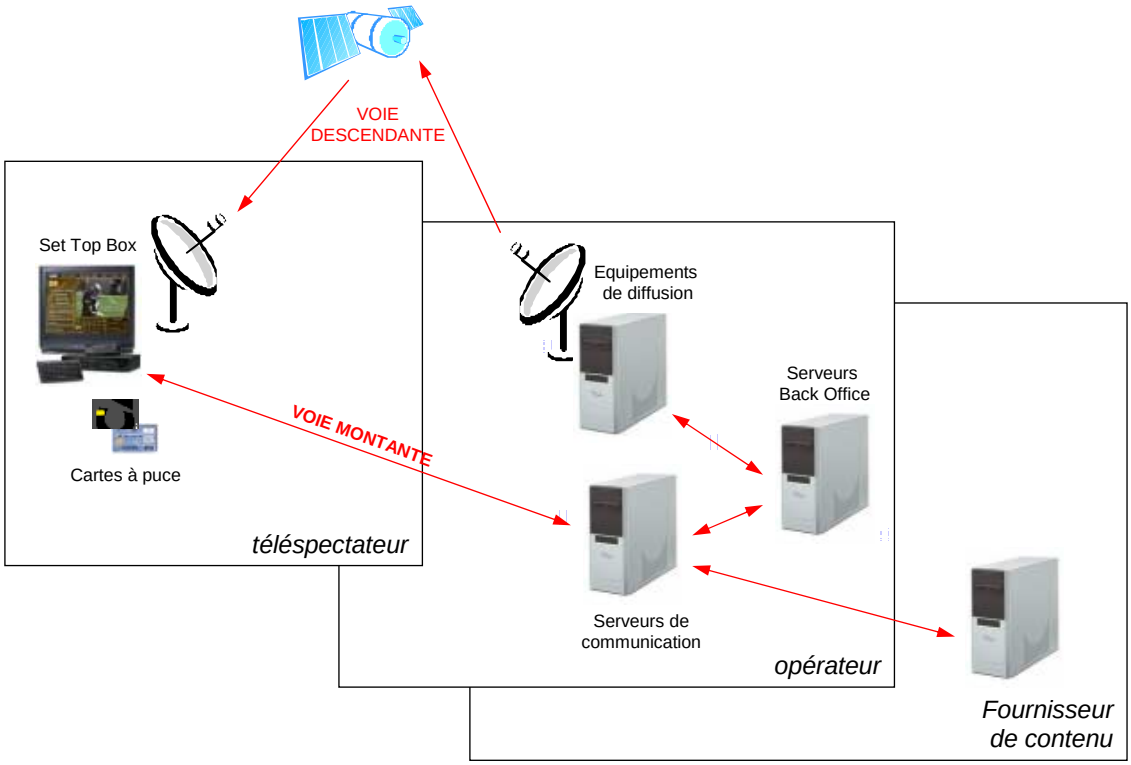

FIG. 1 – *Architecture de la télévision interactive.*

télévisées. Par exemple, un téléspectateur peut consulter la météo de sa ville, interroger ses comptes bancaires ou faire des paris sur une course hippique dont il est en train de voir la retransmission. Notre première étude s'est portée sur l'adaptation du service technique de persistance utilisé par ces logiciels interactifs, en fonction de son environnement d'exécution. L'infrastructure réseau de la télévision numérique a pour particularité d'intégrer trois types de communication très différents (voir figure 1) :

- la voie descendante de l'opérateur vers le téléspectateur via le satellite, cette liaison fournit un débit important et constant mais elle est mono-directionnelle et partagée par tous les téléspectateurs, ce qui peut entraîner un temps de latence assez long;
- la voie montante permet l'échange entre le téléspectateur et l'opérateur via une ligne téléphonique. Cette liaison permet un échange d'informations privilégié avec le téléspectateur mais a un débit limité et sa disponibilité n'est pas garantie;

– entre les serveurs de l'opérateur et du fournisseur de contenu, les informations sont échangées grâce à un réseau filaire avec un bon débit et une bonne disponibilité.

Les communications entre les machines de la télévision numérique reposent donc sur des réseaux hétérogènes ayant des contraintes structurelles qui ne permettent pas d'envisager des communications bidirectionnelles permanentes entre les participants de l'application.

De plus, l'infrastructure de la télévision interactive fait interagir trois types de participants sur les machines desquels le service de persistance doit s'exécuter :

– le fournisseur de contenu de télévision interactive qui possède des serveurs applicatifs;
– l'opérateur de télévision qui possède des serveurs "back office" (qui font la liaison avec le fournisseur de contenu), des serveurs de diffusions (qui diffusent les informations aux utilisateurs par voie descendante) et des serveurs de communications (qui gèrent les échanges avec les utilisateurs par voie ascendante),
– les téléspectateurs qui possèdent un récepteur de télévision numérique (dont la configuration matérielle est proche de celle d'un PC il y a cinq ans) et des cartes à puces (celles liées à l'abonnement de télévision numérique qui authentifient l'utilisateur et celles de paiement bancaire).

Ces machines possèdent des capacités (de mémoire, de traitement, etc) très différentes. Dans le cadre de ce projet, nous avons proposé d'adapter le service de persistance pour qu'il anticipe la réplication de certaines données, des serveurs vers le récepteur de l'utilisateur. Ainsi l'utilisateur peut consulter ces informations avec un temps de latence réduit, sans attendre la diffusion par voie descendante de l'information ou la connexion par voie montante. Nous proposions aussi d'ajouter un service de négociation de bande passante entre les différents diffuseurs de contenu permettant de facturer son utilisation. En dernier lieu, cette étude concluait à la nécessité d'un cadre de conception plus générique des services techniques adaptables [HéR 01].

3 Contributions de la thèse

Depuis, nos travaux ont conduit aux constatations suivantes : les solutions permettant l'adaptation des intergiciels ou des applications leur confèrent une plus grande efficacité et une meilleure portabilité dans les environnements hétérogènes. Cependant, ces différents types d'adaptation ne sont pas suffisants :

– L'adaptation d'un intergiciel a des répercutions sur le fonctionnement de toutes les applications qui fonctionnent au dessus de cet intergiciel. Une telle adaptation ne peut donc pas prendre en compte les besoins particuliers de chaque application.
– Bien qu'on soit capable de fournir les concepts et les outils pour la faciliter, l'adaptation des applications doit être gérée par le développeur de l'application qui doit avoir connaissance de tous les éléments de l'application et des différents comportements de l'application. Le développeur n'étant pas spécialiste des services techniques, il ne peut pas adopter la même démarche pour les services techniques. De plus, il faut maintenir la séparation du code fonctionnel et du code technique.

C'est pourquoi l'adaptation des services techniques peut venir en complément de l'adaptation des applications. Les deux types de code étant développés de façon indépendante, on peut modifier le comportement de l'un sans modifier le comportement de l'autre et offrir un comportement global résultant de l'adaptation des deux types de code.

Nos travaux se concentrent donc sur l'adaptation des services techniques. Nous notons trois éléments problématiques autour de cette thématique :

- Dans les modèles industriels tels que les EJB ou CCM, la conception, à base d'objet, des services techniques ne facilite pas leur adaptation. Il est difficile d'identifier les différentes tâches accomplies par le service et de modifier son comportement.
- Le code grâce auquel un composant applicatif bénéficie d'un service technique est enfoui dans le code de son conteneur et de l'intergiciel sur lequel il repose (dans le cas des modèles à composants reposant sur un intergiciel). Il n'est donc pas possible d'ajouter un nombre indéterminé à l'avance de services techniques et, pour un service technique donné, il est même assez difficile de changer d'implantation.
- Enfin, il n'existe pas de mécanisme pour gérer dynamiquement (c-à-d au cours de l'exécution de l'application) l'adaptation des services techniques.

Nous proposons donc de faire évoluer la tâche du développeur de services techniques en lui proposant un nouveau modèle de conception et de développement. Grâce à celui-ci il pourra représenter les différents comportements ou "personnalités" de son service, identifier les tâches élémentaires et créer différents assemblages autour d'une base commune, chacun étant associé à une personnalité particulière. Les services techniques développés selon cette nouvelle méthodologie sont "combinés" avec les composants applicatif afin d'être adaptés à la fois aux besoins de l'application (si celle-ci a besoin d'un service en plus, en moins, d'une personnalité particulière) et à l'environnement d'exécution. Enfin la gestion de cette adaptation est confiée à un service de gestion dynamique de l'adaptation qui va permettre de localiser, choisir et "combiner" les personnalités adéquates.

Nos travaux ont permis les contributions suivantes :

1. Dans un premier temps, nos efforts se sont concentrés sur un cadre de conception pour des services techniques adaptables. Les services techniques étaient jusqu'alors développés selon les principes de la programmation orientée objet. Nous proposons de les développer grâce au modèle à composants et plus particulièrement le modèle Fractal. Ce modèle offre des outils génériques de conception, avec plusieurs niveaux d'abstraction et un modèle à composants hiérarchique. Grâce à une plus grande modularité de services techniques, nous espérons accroître leur adaptabilité. Elle est facilitée notamment par le partage de modules entre les différentes personnalités d'un service.

2. Pour faciliter la manipulation des services techniques lors de l'adaptation, nous définissons un niveau de description des services techniques basé sur l'expression de la qualité de service. Cette description permet de caractériser les performances des services techniques en fonction de l'environnement d'exécution et peut être utilisée

pendant l'adaptation pour choisir la personnalité du service technique la plus adéquate.

3. Nous redéfinissons l'assemblage que forme le composant applicatif et les services techniques. En effet, jusqu'à présent il était implicite : les services techniques associés aux composants étaient ceux fournis par l'ORB et il n'était pas possible d'en changer. Nous proposons de permettre aux composants applicatifs de bénéficier d'autant de services techniques que nécessaire et les plus adéquats.

4. Enfin, nous présentons un service permettant l'adaptation dynamique des services techniques. Grâce aux informations qu'il collecte sur l'environnement d'exécution des services techniques et en fonction des besoins des applications, ce service répertorie et fournit des services techniques adaptés aux applications. Pour cela, un annuaire de services techniques utilise la couche descriptive évoquée dans le point 2. Un "contrat" manipule l'assemblage du composant applicatif et des services technique évoqué dans le point 3.

4 Plan du document

La suite de ce rapport de thèse comporte quatres grandes parties, chacune divisée en chapitres.

- La première partie se divise en deux chapitres : "Contexte et Problématique" et "Etat de l'art et Positionnement". Le premier chapitre situe le contexte de nos travaux : les nouvelles applications distribuées à base de composants, le modèle à composant grâce auquel elles sont développées ainsi que les intergiciels sur lesquels les composants reposent. Ce chapitre décrit ensuite notre problématique : fournir à ces applications des services techniques adaptés à l'hétérogénéité des environnements d'exécution dans lesquels elles évoluent.
  Le second chapitre présente un état de l'art des solutions d'adaptation dans le domaine de la programmation par composants et notre positionnement par rapport à ces solutions.

- La partie II regroupe l'ensemble de nos propositions théoriques (chapitre 1) et leur validation à travers leur mise en œuvre (chapitre 2). Dans le chapitre 1, nous proposons d'abord un cadre de conception pour les services techniques. Construits à base de composants Fractal et enrichis d'un niveau descriptif, les services techniques ainsi conçus sont plus facilement manipulables. Puis nous décrivons des composants de gestion (moniteur, annuaire, coordinateur, contrat) qui gèrent l'adaptation dynamique des services techniques par rapport à l'environnement d'exécution et aux besoins de l'applications.
  Dans le second chapitre, nous montrons, à travers l'exemple d'un service technique de transaction, les bénéfices offerts par le nouveau modèle de conception des services

techniques. Enfin, nous détaillons le prototype des composants de gestion réalisé.

– La partie III résume les apports et les limites des solutions présentées. Elle conclut avec les perspectives de ce projet.

– Enfin, la partie IV contient les annexes et la bibliographie de ce document.

Première partie

Problématique et Etat de l'art

1

Contexte et Problématique

Nos travaux ont pour but l'adaptabilité des services techniques; ces services sont mis à dispositions des développeurs d'applications pour améliorer la qualité de service de leur code (ex: service de transaction, de persistance, etc). Ils sont utilisés par toutes les applications de la plate-forme et sont donc dits "transversaux". Fournis par la plate-forme à composants, ils sont déclenchés automatiquement avant et après l'exécution d'une méthode d'un composant. Afin de gagner du temps de conception et améliorer leur fiabilité, les services techniques sont développés par des spécialistes du domaine et réutilisés par les différents développeurs d'applications. Une utilisation plus systématique de ces services est faites depuis l'avènement du modèle à composants qui les intègre à son modèle d'exécution. Le principe de service technique et son usage sont détaillés dans les sections 1.2.2 et 1.2.3.

Notre motivation pour l'adaptabilité des services techniques vient du fait que l'on veut améliorer la qualité de service des nouvelles applications distribuées (exemple section 1.1.3). Pour cela, on veut pouvoir agrémenter ces applications de services techniques. Mais ces applications avec leurs services techniques s'exécutent dans des environnements très hétérogènes et changeants (description section 1.1). Or les applications comme les services techniques conçus à base d'objets ne sont pas adaptés à ce genre d'environnement (voir section 1.4.4). Concernant les services techniques, développés à base d'objet, il n'est pas facile de modifier leur comportement en cours d'exécution. De plus, le code permettant d'associer les services techniques aux composants applicatifs est souvent enfoui dans l'architecture de la plate-forme et leur utilisation est figée. Il est donc difficile d'ajouter un service ou encore d'en changer d'implantation. Ainsi il est nécessaire de fournir une plate-forme proposant des applications et des services techniques capables d'adapter leur comportement à leur environnement.

L'un des modèles qui facilite le développement d'applications adaptées à leur environnement est le modèle à composants (voir section 1.2). Le modèle à composants facilite l'expression des tâches, ce qui facilite l'adaptation. D'ailleurs, il existe déjà plusieurs solutions pour l'adaptation des applications (voir la section de l'état de l'art 2.1.4) qui peuvent être complémentaires de la solution que nous proposons pour les services techniques. Par ailleurs, le modèle à composants, grâce aux intergiciels sur lesquels il s'exécute, fournit des

services techniques qui sont un premier pas vers l'amélioration de la qualité de services. Mais comme nous l'avons déjà dit, ces services ne sont pas adaptés à ces environnements. Le rapport entre services techniques et modèles à composants ici est double: d'une part c'est le modèle à composants qui fournit les services techniques aux composants applicatifs et d'autres part c'est aussi un modèle qui semble être un bon candidat pour la conception des codes adaptables car il facilite l'expression des tâches.

Afin de définir le contexte de nos recherches, dans le chapitre suivant, nous détaillerons donc la notion d'environnement hétérogène et proposerons un exemple d'application dans ce contexte (section 1.1). Nous rappellerons les grands concepts de la programmation par composants ainsi que le rapport entre services techniques et modèles à composants (section 1.2). Nous évoquerons la notion d'intergiciel (section 1.3) qui bien que permettant au développeur de s'abstraire de la complexité de la couche matériel, ne peut pas totalement cacher les variations de qualité de service. Ensuite nous identifierons les différents types d'intergiciels qui peuvent convenir à nos besoins. Enfin nous poserons les différents éléments de la problématique (section 2.12).

1.1 Environnements hétérogènes et besoin d'adaptabilité

Les applications de l'informatique ubiquitaire peuvent être de type client/serveur ou peer-to-peer. Outre les serveurs, elles reposent sur des terminaux nomades, que l'utilisateur emmène avec lui afin de l'assister dans sa vie quotidienne ou dans son travail. Ces terminaux sont typiquement de type PDA, c'est-à-dire de petite taille, fonctionnant sur batterie, communiquant grâce à des réseaux sans fil de faible portée. Mais ils peuvent aussi être des cartes à puces, plus sécurisées, ou des téléphones portables, qui permettent facilement de communiquer grâce au réseau GSM.
Dans cette partie, nous montrons l'hétérogénéité des environnements d'exécution des nouvelles applications. Pour cela nous caractérisons d'abord la notion d'environnement d'exécution. Ensuite, nous montrons combien les environnements d'exécution fournis par les nouveaux terminaux sont hétérogènes : différentes capacités de stockage, réseau, puissance de calcul, etc. Enfin, nous détaillons un exemple d'une nouvelle application déployée sur des machines hétérogènes.

1.1.1 Définition du contexte d'exécution

La notion de "contexte d'exécution" est assez vague, ici nous essayons d'en donner une vision un peu plus fine mais qui ne peut être exhaustive dans la mesure où il existe une grande diversité d'information pour la caractériser selon qu'on s'intéresse plus à la qualité des interfaces graphiques, à la rapidité d'exécution, etc.
On nommera *contexte d'exécution* l'ensemble des caractéristiques de l'environnement dans lequel une application s'exécute. Le contexte d'exécution peut englober différents facteurs que l'on peut regrouper en deux grandes catégories : le profil de l'utilisateur et les caractéristiques logicielles et matériel de la machine, ces dernières étant composées de : la

localisation de la machine exécutant le programme, le type de terminal et de ses caractéristiques physiques, la connectivité du réseau, les ressources logicielles, les périphériques accessibles.
Il n'est pas possible d'en faire une liste exhaustive, car ces informations sont très nombreuses et de nouvelles données peuvent s'ajouter. Les sections suivantes reprennent chacune de ces 5 catégories et en détaillent les caractéristiques afin de mieux cerner le problème. Des travaux ont pour objet l'étude et la surveillance de toutes ces données, comme par exemple SAJE ([SAJ], [LES 03]), modèle dans lequel chacune des ressources est réifiée et fournit l'ensemble des objets pour les observer. Dans le contexte de la programmation par composants et particulièrement des services techniques, nous nous intéresserons plus particulièrement aux caractéristiques logicielles et matérielles de la machine. En effet, la plupart du temps, l'utilisateur n'a pas les connaissances, ni le besoin (qu'il aurait pu exprimer à travers le profil utilisateur) de gérer les services techniques, ces derniers étant écrits par des développeurs spécialisés.

La localisation de la machine exécutant le programme

caractéristiques :

- pays, ville, etc
- localisation GPS

De cette localisation peuvent dépendre les données et les services fournis à un utilisateur. Par exemple, dans une application mobile de proximité [THI 03], l'acheteur potentiel reçoit des informations sur les biens qu'il désire acquérir et qui sont proches de lui physiquement.

Le type de terminal et ses caractéristiques physiques

alimentation
caractéristiques :

- batterie (type, autonomie)
- secteur

Cette donnée est importante surtout pour les assistants digitaux ou les ordinateurs portables qui peuvent alterner entre une alimentation par batterie ou une alimentation sur secteur. En effet, une utilisation importante du processeur ou des capacités de stockage peut rapidement épuiser une batterie. Si l'on constate que la batterie restante est faible, on veillera à ne pas utiliser des services grands consommateurs de temps CPU et d'accès mémoire.

stockage
caractéristiques :

- différents types de stockage (disque dur, ROM, RAM, Flash)
- capacité,
- espace utilisé,

– vitesse d'accès

Les capacités de stockage d'une machine à court terme ou à long terme sont déterminantes respectivement pour deux facteurs : la rapidité d'exécution d'un programme et la capacité de la machine à stocker à long terme des informations (base de données, proxy). Concernant la rapidité d'exécution, on devra veiller à ce que le temps d'exécution des services ne devienne pas trop important pour l'usager. Quant à la capacité de stockage, elle détermine la quantité d'informations qui pourra être rendue persistante par les services, par exemple des données distantes répliquées pour améliorer le temps d'exécution, un profil utilisateur basé sur des informations qu'il fournit ou sur ses habitudes, etc.

CPU

caractéristiques :

– modèle,
– vitesse,
– charge,
– taille de cache

Les caractéristiques et le taux d'occupation du CPU vont déterminer principalement la vitesse d'exécution d'une application. Un service nécessitant trop de travail de la part du CPU, soit ne fonctionnera pas, soit fonctionnera trop lentement pour que l'utilisateur soit satisfait. Il faut donc veiller à ce que les services exécutés le sollicitent raisonnablement.

lecteurs

caractéristiques :

– CD,
– DVD,
– SmartCard,
– zip,
– etc

La présence ou non de lecteurs conditionne les capacités de stockage d'une machine. Comme pour le stockage, cette caractéristique déterminera le nombre d'informations potentiellement stockables.

interfaces homme/machine

caractéristiques :

– microphone,
– clavier,
– écran,
– carte son
– carte graphique
– souris, touchpad

Les interfaces de communication sont aussi appelées partie "présentation". C'est d'elles que dépend notamment l'interface graphique d'une application. La présence ou non de ces interfaces de communication va déterminer le niveau d'interaction possible entre l'utilisateur et sa machine . La carte graphique influence fortement la qualité de rendu des applications à haut niveau graphique.[1]

Le réseau

caractéristiques :

- Les différentes normes,
- Largeur bande passante
- Bande passante utilisée
- Connexion ou non

Dans le domaine des applications distribuées, les informations sont échangées entre les modules logiciels à travers les connexions réseau. Ces connexions conditionnent donc la cohérence et l'efficacité de l'application. La norme de la connexion réseau, sa largeur de bande passante et sa disponibilité déterminent sa qualité de service : le taux d'erreur, le taux de déconnexion, taux de transfert, etc. Ces informations sont donc à prendre en compte dans la conception d'un service. Par exemple, lorsque la qualité du réseau est trop faible, on doit envisager des solutions qui permettent d'utiliser le réseau le moins possible, comme la duplication des données.

Les ressources logicielles

caractéristiques :

- Bus logiciel
  - Type de bus logiciel,
  - version
- Multi-thread ou pas,
- Jre.

Pour s'exécuter, les composants logiciels se reposent principalement sur un bus logiciel. D'autre logiciels, librairies ou ressources logicielles (ex : base de données) sont nécessaires à l'exécution de certains composants. Leur version peut aussi influer sur leur rapidité ou les fonctionnalités disponibles.

Les périphériques accessibles

caractéristiques :

- imprimante,
- scanner,

1. Ces informations font partie de l'environnement d'exécution des applications, c'est pourquoi nous les avons mis dans cette classification. Cependant, comme nous l'avons dit précédemment nous ne nous intéresserons pas particulièrement à la partie présentation et donc à ces données.

– webcam.

Ces informations peuvent être utilisées pour ajouter de nouvelles fonctionnalités à l'application et les adapter au périphérique (ex : ajout d'une interface d'impression).

1.1.2 Contextes d'exécution hétérogènes

L'émergence simultanée des outils informatiques personnels portables et des réseaux sans fil a considérablement augmenté le nombre de terminaux connectés aux réseaux. Aujourd'hui, un utilisateur peut être connecté en permanence à des applications distribuées à son domicile ou au travail, en utilisant des ordinateurs puissants, mais également des ordinateurs portables comme des assistants personnels ou des téléphones mobiles (qui, bien que connectés au réseau et offrant des capacités de stockage et de traitement, sont bien moins puissants), ou encore d'autres outils informatiques présents à la maison comme les terminaux de télévision numérique, les consoles de jeux qui partagent beaucoup de caractéristiques avec les ordinateurs portables. De plus, il semble que l'on n'aille pas vers une convergence de ces outils mais plutôt que l'utilisateur préfère avoir plusieurs outils spécialisés qui peuvent partager les informations.
Le tableau 1.1 présente, pour exemple, les caractéristiques principales de quelques machines sur lesquelles les applications à base de composants sont susceptibles de fonctionner. Les machines décrites dans ce tableau sont représentatives des besoins hétérogènes

	ordinateur personnel	assistant numérique personnel	récepteur numérique de télévision
fréquence du CPU (en MHz)	3000	400	800
mémoire vive	2 Go DDRS-DRAM	64 Mo RAM	32 Mo SDRAM
type de connexion réseau	ADSL (128Kbps-8Mpbs)	IrDA (9Kbps-115Kbps), Bluetooth (2Mbps-12Mbps), WI-FI (1Mbps-54Mbps)	ligne ascendante : ligne téléphonique classique (56Kbps) ou ADSL (128Kbps-10Mbps), ligne descendante : réception satellitaire ou hertzienne
disque dur	250 Go	4 Go	20 Go, 16 Mo de mémoire Flash

TAB. 1.1 – *Caractéristiques principales de machines représentatives de l'hétérogénéité des machines actuelles*

en terme de machines, dans les nouvelles applications :

– un ordinateur personnel présent dans les bureaux et les foyers. Grâce à sa grande puissance de calcul et sa bonne connectivité, il permet d'exécuter de nombreuses applications complexes, utilisant le réseau.

- un assistant numérique personnel (ou PDA) utilisé de plus en plus dans les déplacements. Il est souvent équipé de connection WI-FI, IrDA ou Bluetooth qui permettent de créer des réseaux spontanés.
- un récepteur de télévision numérique (ou Set Top Box) déjà largement utilisé par les abonnés de la télévision satellitaire, bientôt pour la télévision numérique hertzienne. Il permet aussi d'exécuter des applications associées aux émissions ainsi que d'enregistrer des vidéos ce qui nécessite une grande capacité de stockage.

Il existe une grande disparité entre les caractéristiques de ces différentes machines, que ce soit en terme de :

- capacité de traitement : dans les exemples, de 400 MHz à 3 GHz pour le processeur et de 32 Mo à 2 Go pour la mémoire vive
- capacité de stockage de données qui varient entre 16 Mo de mémoire flash pour un récepteur numérique simple à 250 Go de disque dur pour un ordinateur personnel.
- capacité réseau qui sont vraiment hétérogènes. Elle peut varier en terme de largeur de bande passante (ex : ADSL de 128 Kbps à 10 Mbps) mais aussi en taux d'erreur ou en fréquence de déconnexions.

1.1.3 Exemple d'application en milieu hétérogène

De nouvelles applications destinées à fonctionner sur ces machines hétérogènes sont de plus en plus développées, elles tirent parti de leurs nouvelles capacités : la facilité de déplacement, la connectivité des réseaux spontanés, les nouveaux types de connexion ayant des débits de réception importants ou les interfaces homme/machine plus familières que celles des PC. Pour illustrer notre propos, prenons comme exemple une application de paris à domicile (voir figure 1.1). Cette application permet à l'utilisateur de consulter les renseignements nécessaires aux paris (horaires, participants, cotes, etc) et de passer des paris. Précédemment, nous avons identifié deux situations d'adaptation de l'application à son contexte d'exécution qui sont présentes dans cette application.

Dans le premier cas, une même application est déployée sur plusieurs machines, de façon indépendante. Ce cas se traduit pour notre application par le fait que l'application peut être déployée sur un ordinateur personnel ou sur un récepteur de télévision numérique. L'application déployée sur l'ordinateur personnel est utilisée à la maison avec une connexion permanente ADSL : l'utilisateur parie quand il le veut car l'application est en contact permanent avec le serveur de renseignements; les informations disponibles pour l'utilisateur sont donc à jour et il est aisé d'assurer la sécurité et d'exécuter les transactions nécessaires à la validation du pari et au paiement. En effet, l'accès aux bases de données distantes et aux services effectuant des traitements de données est facilité par une bonne connectivité au réseau. Par ailleurs, cette même application est déployée sur un récepteur satellite, grâce auquel l'utilisateur consulte les renseignements par voie satellite et passe les ordres de paris par ligne téléphonique. Ce mode de connexion convient bien à cette application dans la mesure où la taille des informations reçues (informations sur les courses) est importante comme celle de la bande passante descendante (réception satellitaire) et celle envoyée (validation de pari) est faible comme celle de la voie ascendante (ligne téléphonique).

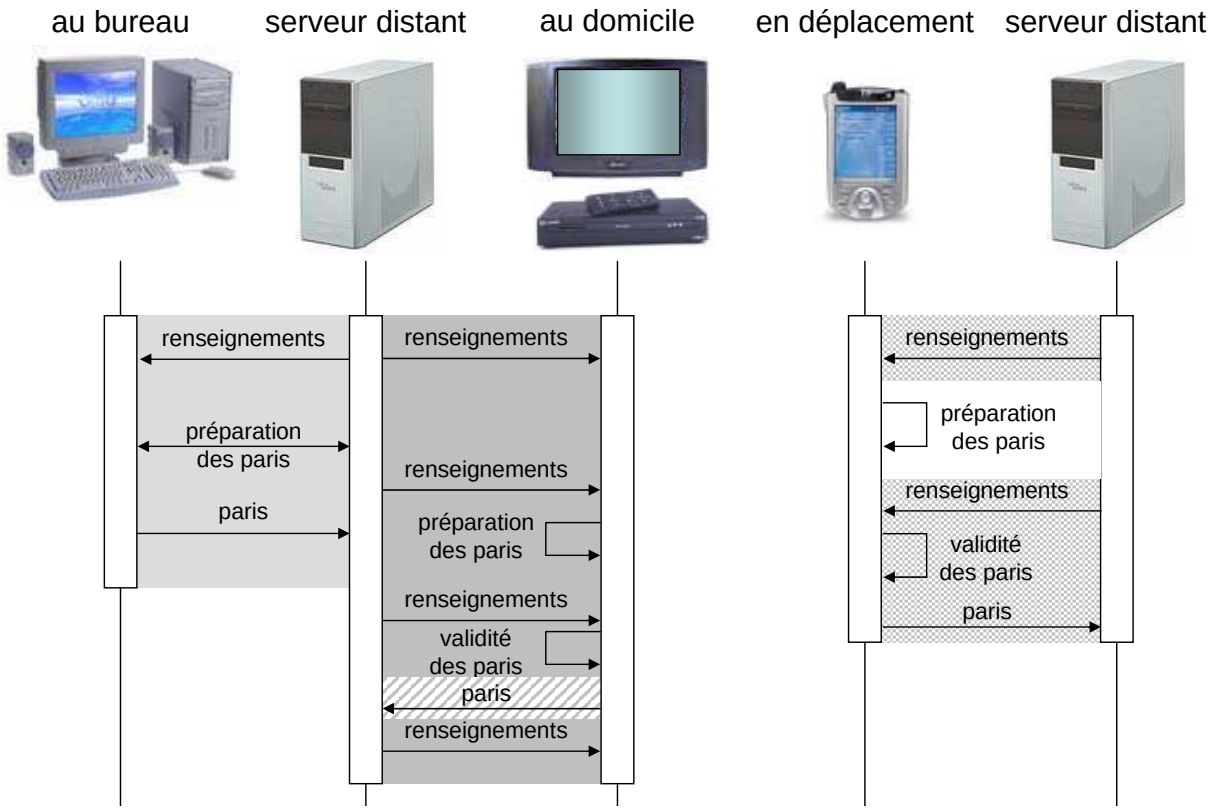

FIG. 1.1 – *Exemple d'application en contexte hétérogène*

Dans le second cas, l'adaptation se fait lors d'un changement de contexte d'une application. Dans notre exemple, l'application de paris est déployée sur un PDA qui possède une connexion WI-FI. Dans ce cas, le contexte alterne entre des périodes de connexion et déconnexion avec un PC (son ordinateur personnel ou celui du PMU). Concernant la consultation des données, lorsque le PDA est synchronisé avec un PC, le programme consulte les données qui sont disponibles sur le serveur de données distant puis il les duplique sur le PDA. Ainsi, lorsque l'utilisateur déconnecte son PDA du réseau, il peut consulter les derniers renseignements dupliqués et préparer ses paris. Lors d'une future synchronisation, l'utilisateur pourra passer ses paris après que le programme ait préalablement vérifié que les cotes n'ont pas changé depuis la dernière synchronisation. Les données du paris sont mises à jour si nécessaire et le pari est alors passé avec les conditions de sécurité nécessaires.
Les caractéristiques de l'environnement influencent notamment la vitesse d'exécution et la fréquence des pannes d'une machine. En conséquence, les applications écrites sous forme de composants pour fonctionner sur des serveurs ne sont plus adaptées. Pour développer des applications portables sur ces terminaux, il devient nécessaire de prendre en compte l'hétérogénéité de leurs caractéristiques. Pour cela, il est nécessaire de concevoir des applications adaptables.

1.1.4 Adaptabilité

Une définition générale de l'adaptabilité est "*la qualité de ce qui est adaptable*", ce qui est adaptable étant "*ce qui peut s'ajuster à des conditions particulières ou nouvelles*". Une définition dans le domaine des systèmes distribués est que "*l'adaptation est l'opération qui consiste à apporter des modifications à un logiciel ou à un système informatique, dans le but d'assurer ses fonctions et, si possible, d'améliorer ses performances, dans un environnement précis d'utilisation*" [gra].

Selon le moment où l'adaptation intervient (conception, déploiement, exécution) et qui l'opère, on distingue différents types d'adaptations :

- l'adaptabilité *statique* : elle intervient avant l'exécution (pendant la conception ou le déploiement),
- l'adaptabilité *dynamique* : elle intervient tout au long de l'exécution,
- l'adaptabilité *ayant des connaissances sur son environnement d'exécution* : c'est la plate-forme elle-même qui fournit les renseignements nécessaires à l'adaptation lors du déploiement,
- l'*auto-adaptabilité* : elle est initiée par le système lui-même.

L'adaptabilité statique et dynamique s'excluent. Un système ayant des connaissances sur son environnent d'exécution peut être adapter dynamiquement ou statiquement. Un système capable de s'auto-adapter a des connaissances sur son environnement qui lui permettent de faire des choix d'adaptation, il peut être statique ou dynamique. Lorsqu'on parle d'auto-adaptation, on sous-entend fréquemment qu'il s'agit d'auto-adaptabilité dynamique. Un système qui n'est pas auto-adaptable peut être adapté par un intervenant extérieur : un opérateur humain ou un autre système.
L'adaptabilité statique convient à des systèmes déployés dans des environnements particuliers mais n'évoluant pas ou peu pendant l'exécution du système [BRU 01a], par exemple dans un environnement ayant un CPU, une taille mémoire particulière.
L'adaptabilité dynamique convient aux systèmes déployés dans des environnements d'exécution évoluant lentement, par exemple un système où la place mémoire libre diminue lentement à cause du nombre d'informations stockées de plus en plus grand.
L'auto-adaptabilité peut être très avantageuse dans certains environnements très dynamiques tels que les réseaux mobiles où la qualité de transport est versatile. Mais passer de l'adaptabilité statique à l'auto-adaptabilité implique une perte de performance dans la mesure où elle alourdit le travail qui incombe au système : avoir une connaissance de l'environnement d'exécution et effectuer les adaptations.

Enfin on peut distinguer deux grandes façons d'adapter le code [MCK 04] :

- adaptation comportementale : au sein du code lui même, en le paramétrant on modifie son comportement;
- adaptation architecturale : entre les modules logiciels que l'on réassemble en changeant les liaisons.

C'est souvent l'adaptation architecturale qui est choisie car elle ne demande pas d'avoir de connaissance du comportement interne du composant. Avec les modèles à composants

hiérarchiques on a une certaine convergence des deux types d'adaptation car le comportement d'un composant et sa composition sont en grande partie en correlation.

1.1.5 Conclusion

Les serveurs d'entreprise et les ordinateurs mobiles et communicant à des réseaux sans fil, fournissent aux nouvelles applications des environnements d'exécution hétérogènes. Afin de préciser le problème, nous avons défini la notion de contexte d'exécution comme étant l'ensemble des caractéristiques de l'environnement d'exécution de l'application dont nous avons donné une liste non exhaustive. Ce contexte d'exécution peut grandement varier d'une machine à l'autre, selon les contraintes économiques, la portabilité, ou encore le type du réseau environnant. Nous avons détaillé l'exemple d'une application de pari à domicile développée sous la forme de composants. Nous avons vu que cette application, déployée sur les différentes machines pouvait adopter un comportement différent, tirant parti des informations environnantes, de la capacité du réseau, etc. Pour avoir une application, comme celle décrite ci-dessus, c'est-à-dire adaptée à son environnement, il est nécessaire de choisir un modèle de conception et de développement qui permet de gérer facilement la distribution de l'application et facilite son évolution. Le modèle à composants, que nous présentons dans la section suivante, a de tels caractéristiques.

1.2 Modèle à composants : étude de l'adéquation aux nouvelles applications distribuées

En corrélation avec la diversification des terminaux (terminaux mobiles, récepteurs de télévision numérique, consoles de jeux, etc), on constate une multiplication d'applications exploitant leurs nouvelles capacités (mobilité, interfaces homme/machines conviviales, etc). Comme nous l'avons vu pour l'exemple de la télévision numérique ou celui de l'application de paris à domicile, on aimerait proposer un intergiciel qui permette à tous les intervenants de l'application (fournisseur de contenu, développeur de l'application, diffuseur, etc) de coopérer autour d'architectures réseaux et matériels hétérogènes.
Nous allons montrer que la programmation orientée objet, jusqu'ici utilisée, ne suffit plus au développement rapide d'applications hautement distribuées et sûres. Le modèle à composants semble l'un des mieux adaptés à ce genre d'applications : il préconise la conception modulaire de l'application, facilitant la réutilisabilité et l'exécution distribuée de l'application. De plus, les plate-formes à composants fournissent des services techniques qui apportent une meilleure qualité de service sans alourdir le développement de l'application.
Dans cette partie, nous allons tout d'abord caractériser le type d'application ciblé par notre travail. Ensuite, nous introduirons les concepts de la programmation par composants et nous étudierons différentes implantations de ce modèle. Enfin, nous nous attarderons sur un des concepts familiarisé par la programmation par composants : les services techniques.

1.2.1 Caractéristiques des nouvelles applications distribuées

Grâce à l'efficacité récente des nouveaux terminaux mobiles et des réseaux qui les relient, de nouvelles applications sont identifiées : les applications temps-réel, les applications multimedia, les applications hautement distribuées ou les applications mobiles.
Exemple particulièrement représentatif de ces nouveaux types d'applications, les "applications de proximité" ([HéR 02] et [THI 03]) ont pour but de proposer des services dépendant de la localisation de l'utilisateur, dans des réseaux très mobiles : par exemple, acheter un bien proche de l'utilisateur, recevoir ou échanger des informations le concernant afin d'effectuer son choix.
Les applications ciblées par notre travail sont à la croisée de deux grands types d'applications :

- les applications d'entreprise (en anglais : corporate computing) : ces applications sont déployées sur des terminaux très ciblés, elles sont fixes, sûres et rapides d'utilisation. La complexité et la taille importante du code développé impliquent un temps de vie assez long. Elles sont écrites selon le modèle trois tiers, amélioration du modèle client/serveur décrit dans [ORF 99]. Une application développée selon ce modèle est constituée de trois parties : une partie client sur le terminal qui effectue la présentation des résultats, une partie logique applicative qui effectue un traitement sur les données stockées dans des bases de données, une troisième partie regroupant les bases de données.
- les applications de l'informatique ubiquitaire (en anglais : ubiquitous ou pervasive computing) : elles sont déployées sur de nombreux terminaux, légères, migrent facilement et sont facilement et dynamiquement découvertes par leurs utilisateurs, elles ont un temps de vie assez court. Dans de nombreuses applications destinées aux environnements mobiles et hétérogènes, des réseaux pair à pair ou une variante de ce modèle - le pair à pair hybride - remplacent de plus en plus souvent les architectures classiques Client/Serveur. Dans ces réseaux, les différents participants de l'application n'ont pas de rôle prédéfini : ils peuvent successivement passer du rôle de client à celui de serveur, pour fournir soit des données, soit des services applicatifs. Ainsi le rôle d'un terminal ne dépend pas uniquement de ses caractéristiques en terme de connexion et de puissance de calcul. Chaque participant de l'application (ou pair) expose aux autres participants les données et services qu'il veut partager. Lors d'un échange de données ou de services, le participant intéressé prend spontanément le rôle de client, le participant fournisseur des données ou des services prend le rôle de serveur. Le problème rencontré est alors que l'application n'est pas forcément conçue pour prendre en compte les particularités de ces machines qui jouent le rôle de serveur. Ce type d'application regroupe les Web services, les applications pair à pair, etc.

Ces nouvelles applications ont les fonctionnalités des applications ubiquitaires et la qualité de service des applications d'entreprise. Elles ont pour avantage de pouvoir être exécutées dans de nombreux environnements d'exécution, même très contraints; elles ont un temps de développement très court; elles sont très fiables.
Ces avantages en terme de qualité de service pour l'utilisateur impliquent du point de vue de la programmation de nombreuses contraintes.

En effet, comme les applications ubiquitaires, elles sont déployées dans des environnements d'exécution extrêmement distribués, très sujets aux pannes et aux problèmes de localisation d'informations et de services.
De plus, concernant le temps de développement, ces nouvelles applications ayant les mêmes fonctionnalités que les applications ubiquitaires, on peut donc espérer qu'elles aient un temps de développement similaire. Or, on constate que 50% des applications mobiles développées en 2002 étaient obsolètes à la fin de la même année, ce qui traduit bien le peu de temps alloué pour développer ce genre d'application.
Enfin, comme pour les applications d'entreprise, les applications visées doivent être sûres et fiables. En effet, ces applications permettent des traitements sensibles tels que les virements bancaires ou des échanges de données personnelles.

1.2.2 Un modèle de développement pour les nouvelles applications distribuées : le modèle à composants

La taille, la complexité et les contraintes d'adaptation à l'environnement d'exécution de ces applications grandissant, la programmation orientée objet, bien que représentant une grande avancée par rapport à la programmation procédurale, ne suffit plus. On a besoin d'un modèle prenant en compte l'ensemble du cycle de développement de l'application, facilitant le développement, la réutilisation et l'intégration des applications dans des systèmes distribués et hétérogènes. L'un des modèles répondant à ces attentes est le modèle à composants : il permet de construire une application robuste en assemblant des briques logicielles.
Après être revenus brièvement sur la programmation orientée objet, nous détaillerons les buts de la programmation par composants, ses concepts et les différentes solutions implantant ce modèle. Nous nous attarderons particulièrement sur les services techniques, fournis de façon transparente aux composants par la plate-forme, car ils sont au cœur de notre travail.

Programmation orientée objet

Au cours des dernières décennies, la programmation orientée objet (POO) s'est montrée être un paradigme puissant pour la réalisation d'applications évolutives et à grande échelle. Elle nous fournit les abstractions nécessaires à la représentation modulaire des éléments du problème modélisé par le programme. Selon le principe d'encapsulation, chaque élément apparaît sous la forme d'un objet possédant des données (représentées par les attributs) et des opérations accessibles à travers des interfaces (les méthodes). Ce principe garantit la " bonne " manipulation des objets. Les objets ayant des caractéristiques et des comportements en commun sont les instances d'une même classe, moule de conception permettant la réutilisation. La POO introduit deux autres concepts : le polymorphisme (permet à deux objets de partager une même interface sans toutefois l'implanter de la même façon) et l'héritage (permet de spécialiser et d'étendre classes et interfaces).
Si la POO permet une meilleure modularité et réutilisabilité que la programmation procédurale, grâce notamment aux concepts de classe et d'interface, ces caractéristiques ne s'étendent pas aux objets. En effet, il est difficile d'adapter un objet à son environnement

en lui apportant des services tels que la persistance ou la distribution car la POO ne préconise pas la séparation des aspects qui vise à dissocier les services réalisés par l'applications des propriétés ou aspects extra-fonctionnels. Par conséquent, la dispersion du code transversal aux applications est favorisée. Ainsi, lorsqu'on veut ajouter la persistance à un objet, il faut modifier son implantation, ce qui est d'autant plus difficile pendant l'exécution. Enfin la POO ne propose pas d'outils et de mécanismes pour exprimer et implanter les associations ou composition d'objets. Les appels de méthodes sont donc éparpillés dans le code, les réassemblages sont donc fastidieux.

Buts de la programmation à base de composant

Paradigme introduit dans les années 90, la programmation à base de composant ([HEI 01], [SZY 02], [RIV 00]) a pour but d'améliorer la réutilisabilité, la sûreté, la flexibilité des applications. Pour cela, elle se base sur la notion de composant et par là même la notion de composition ou d'assemblage. Elle propose de concevoir une application comme un assemblage de briques logicielles pré-fabriquées.
D'après Councill et Heineman dans [HEI 01], le but de l'ingénierie logicielle basée composant (CBSE) est de "créer des assemblages précis de composants bien documentés, de qualité et dignes de confiance". Ainsi les auteurs soulignent le fait que ce type de conception s'intéresse non seulement aux phases de développement du composant mais aussi de sa phase de conception, de modélisation (basée sur des outils de conception, outils visuels, UML, Model Driven Architecture de l'OMG) et d'assemblage (basée la plupart du temps sur des langages de description, ADL en anglais) qui sont confiés à des spécialistes. De cette façon, la réutilisation de composants fiables est facilitée. Certains modèles permettent même le réassemblage, pendant l'exécution, des composants, ce qui permet d'adapter les applications.
Szypersky, dans [SZY 02], met en évidence l'aspect économique de la programmation à base de composants. Il décrit un composant logiciel comme permettant la réutilisation pratique de "parties" logicielles et l'amortissement d'investissements sur plusieurs applications. Pour lui, la programmation par composants permet de concevoir les applications comme une composition de composants achetés, génériques, économiques, rapidement intégrables, développées par des spécialistes du domaine, donc sûres avec des composants développés expressément pour l'application particulière. Ces derniers sont plus chers, plus long à développer mais complètement spécifiques à l'application.

Concepts de la programmation à base de composants

Si, comme nous l'avons vu précédemment, l'ingénierie logicielle orientée composant est bien entendu basée sur les notions de composants et d'assemblage, un ensemble d'autres concepts communs aux différents modèles à composants peut aussi être distingué (notons qu'ils ne sont pas forcément tous implantés par les modèles) : interface, conteneur et services techniques qui sont liés à la séparation de la logique applicative et du code technique, introspection.

Définitions de la littérature Il existe de nombreuses définitions d'un composant, que l'on pourra notamment retrouver dans [SZY 02], elles varient selon l'approche, de la définition générale d'un composant applicable à n'importe quel domaine d'ingénierie à une définition très technique où le composant s'apparente pratiquement à un processus. Les définitions auxquelles nous nous reportons sont indépendantes d'un modèle particulier et font ressortir les particularités du domaine de l'ingénierie logicielle.
Dans [HEI 01], Councill et Heineman donnent les définitions suivantes :

- *Un composant logiciel est un élément logiciel qui se conforme au modèle à composants et peut être indépendamment déployé et composé sans modification selon un standard de composition.*
- *Un modèle à composants définit des standards spécifiques d'interactions et de composition. Une implantation de modèle à composants est dédiée à un ensemble d'éléments logiciels exécutables requis pour supporter l'exécution de composants qui se conforment au modèle à composants.*
- *Une infrastructure de composants logiciels est un ensemble de composants logiciels interagissants conçus pour qu'un système logiciel ou un sous-système construit en utilisant ces composants et interfaces satisfasse clairement les spécifications de performance définies.*

Ces définitions mettent en évidence la distinction qu'il est nécessaire de faire entre un composant logiciel, son modèle à composants et l'infrastructure sous-jacente. Notons que dans cette partie, nous définissons tout d'abord la notion de composant ainsi que ses propriétés, puis nous discutons des différents modèles à composants existants, ainsi que de leurs implantations.
Dans cette définition du composant logiciel, les termes "*indépendamment déployés*" nous

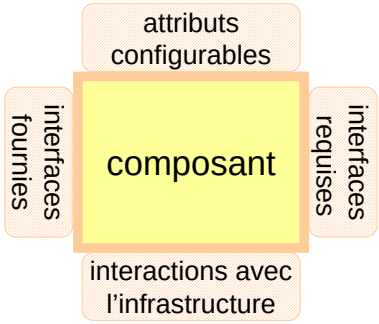

FIG. 1.2 – *Vue extérieure d'un composant*

ramènent au fait que le composant est un modèle logiciel au déploiement autonome, il n'est donc pas nécessaire d'avoir des connaissances sur le comportement interne des autres composants pour le déployer; cependant, il est assemblé ou "*composé*" avec d'autres composants selon des règles prédéfinies par le standard. Dans la définition d'une infrastructure de composants logiciels, les auteurs introduisent la notion de "*performance*", une interface est vue comme un contrat entre le composant client et le composant serveur, garantissant une certaine qualité de service. Une interface regroupant plusieurs méthodes, l'interface

requise par un composant client est un sous-ensemble de l'interface fournie par le composant serveur (voir figure 1.2). Communément, on distingue deux types d'interfaces : synchrones et asynchrones. En mode asynchrone, les échanges entre composants sont non bloquants et se font par l'intermédiaire d'une "boite aux lettres" : MOM ou "message oriented middleware".
La définition d'un composant logiciel donnée par Szyperski dans [SZY 02] est : "Un composant logiciel est une unité de composition avec des interfaces spécifiées contractuellement et des dépendances de contexte explicite. Un composant logiciel peut être déployé indépendamment et est soumis à la composition par une tierce partie." Comme dans la définition précédente, l'auteur souligne à la fois les aspects techniques d'un composant avec la notion d'interface, de dépendance au contexte explicite (qui se traduit sur la figure 1.2, par la notion d'interaction avec l'infrastructure), mais aussi les aspects économiques avec la séparation des métiers, le développement et le déploiement/composition étant faits par deux parties différentes.

Compléments Les définitions que nous avons extraites de la littérature sont des définitions très générales du composant. On y retrouve la notion d'unité de déploiement indépendant, la notion d'assemblage selon des standards définis dans le modèle. Néanmoins, nous identifions deux notions importantes que nous aimerions voir apparaître dans la définition d'un composants : la notion d'instance de composant et la séparation de la vue interne de la vue externe d'un composant.
En effet, les définitions étudiées plus avant ignorent la distinction à faire entre composant et instance de composant qui est similaire à celle faite entre une classe et un objet. Une instance de composant est obtenue à partir d'un composant, dont on peut dire qu'il est le moule. Une instance de composant a un état, fonction des valeurs de ses attributs et de son cycle de vie. Elle peut être partagée par plusieurs applications et plusieurs instances d'un même composant peuvent cohabiter dans une application. Cette simplification est souvent faite pour alléger le discours. Néanmoins rétablir cette distinction permet de mieux comprendre les différents types de réutilisation possible (réutilisation d'un composant ou d'une instance de composant), le partage de composants entre différentes applications, etc.
De plus, ces définitions assez générales ne font pas de distinction entre la vue interne et la vue externe du composant, en effet, on n'y fait pas allusion au "contenu" du composant mais l'on définit sa vue externe. En réalité, un composant expose une ou plusieurs vues externes de lui même correspondant aux différents usages que l'on peut en faire. Les vues externes d'une instance de composant sont offertes par le conteneur qui encapsule l'instance et permet l'utilisation de la logique applicative (le contenu) à travers des interfaces. Dans les modèles à composants actuels, la notion de conteneur n'apparaît explicitement souvent qu'au moment du déploiement des instances. En effet, les conteneurs des différents services techniques ne diffèrent que par les politiques des services techniques qu'ils fournissent aux composants qui sont décidées au moment du déploiement. On les associe donc aux instances de composant. Un conteneur fournit les mécanismes d'interception des appels de méthodes faits sur les composants à travers leurs interfaces, que ce soit des appels entrants ou des appels sortants. De plus, ils fournissent des interfaces, parfois pas

explicitement déclarées par le développeur, qui permettent au système de connaître et modifier l'état d'une instance de composant (la valeur de ses attributs, son contenu, etc). Il est alors capable d'offrir ces informations à l'infrastructure sous-jacente afin qu'elle puisse manipuler le composant. Les appels de méthodes interceptés sont réifiés par le conteneur à travers l'utilisation des différents services techniques.
Un composant et ses instances ont bien entendu les mêmes interfaces. Une interface regroupe un ensemble d'opérations ; c'est un contrat implicite entre le fournisseur qui implante l'interface et le client qui l'utilise. Ainsi, deux instances de composants interagissant à travers une même interface ignorent le fonctionnement l'un de l'autre. Une vue interne, cachée à ses utilisateurs, révèle un contenu plus complexe qui peut être basé sur des objets, des bases de données, etc. Dans certains modèles, dits hiérarchique ou récursifs, un composant peut lui même être composé d'autres composants.

Services techniques

Techniquement, les composant s'exécutent souvent au dessus d'une couche appelée bus logiciel. Ce terme générique regroupe les courtiers d'objets (en anglais Object Request Broker ORB) et les bus de message (en anglais Message Oriented Middleware, MOM), qui fournissent des mécanismes nécessaires aux appels à distances d'objets, respectivement synchrones et asynchrones. Le bus logiciel permet de cacher la distribution et l'hétérogénéité d'implantation des objets.
Le bus logiciel fournit donc notamment le service de distribution mais aussi d'autres "services techniques". Fournis par l'intermédiaire du conteneur, les services techniques facilitent le travail de développement d'un composant. Ils apportent aussi à la plate-forme une plus grande qualité de service.

Définition La notion de service technique (encore appelé service non-fonctionnel) est apparue de façon claire dans les spécifications des modèles à composants métiers (voir figure 1.3). Elle découle du principe de séparation du code fonctionnel du code non-fonctionnel : le développeur de composant se focalise sur la logique applicative de l'application et laisse à la plate-forme le soin de fournir de façon plus ou moins automatique un ensemble de services qui apportent une meilleure qualité de service à l'application globale. D'après [KIN 02] le code applicatif est le code qu'un développeur moyen sait écrire. Par opposition, un service technique est un service assez complexe à implanter pour qu'on en laisse le développement à un spécialiste du domaine et dont l'utilisation peut apporter une meilleure qualité de service à un grand nombre d'applications.
Les services techniques sont fournis avec la plate-forme à base de composants. Ils sont utilisés par les conteneurs des composants lors des invocations de méthode (entrantes ou sortantes). Leur utilisation intervient avant l'exécution de la méthode et après. Par exemple, avant l'exécution d'une méthode, le service de sécurité peut effectuer une authentification, le service transactionnel peut démarrer une transaction. Après l'exécution d'une méthode, le service transactionnel peut effectuer une validation de la transaction ou un abandon (selon que l'exécution de la méthode s'est bien passée ou non).

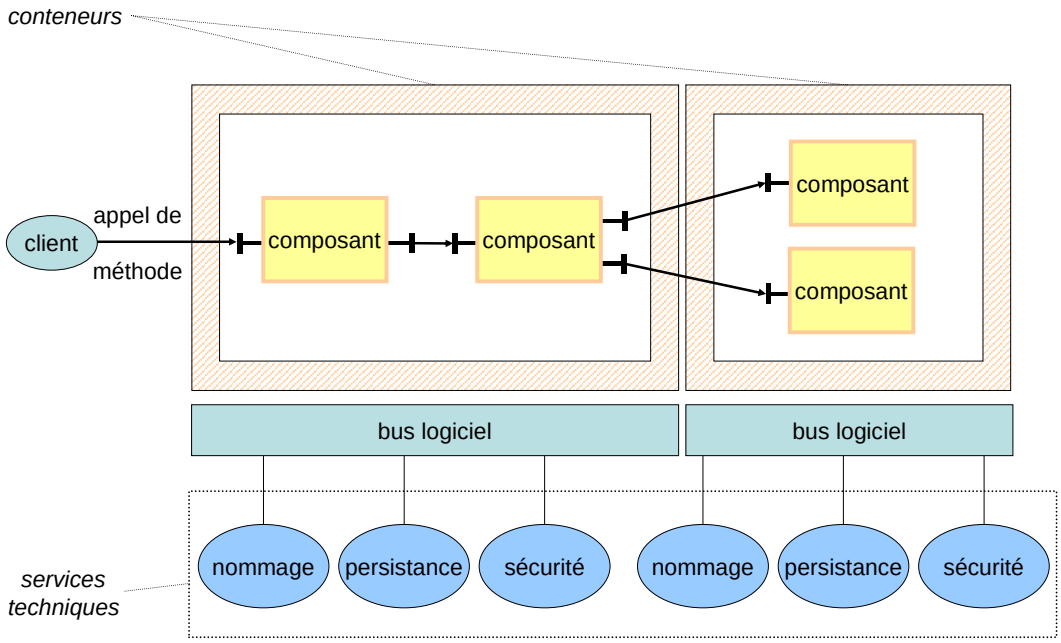

FIG. 1.3 – *Services techniques dans les modèles à composants métiers*

Classification des services techniques En parcourant les spécifications des modèles à composants, on peut identifier un certain nombre de services techniques. Ils sont communément regroupés en 4 grandes catégories :

- les services d'interaction : ils décrivent les interactions entre les composants. Ce sont le service de distribution, le service de transport synchrone ou asynchrone.
- les services de cycle de vie : ils permettent de gérer et de trouver les instances de composant. Ce sont les services de gestion de cycle de vie, de nommage et de courtage.
- les services d'information : ils sont liés à la persistance et au traitement des données. Ce sont les services de persistance, de sécurité et de transaction.
- les services de gestion de qualité de service : ils assurent un niveau de fonctionnement minimal du système et sont totalement intégrés dans la plate-forme. Ce sont les services de tolérance aux pannes, synchronisation, équilibrage de charges.

Cette liste est non exhaustive, elle présente les grandes catégories de services techniques. Chacun de ces services peut regrouper plusieurs fonctionnalités, par exemple, le service de sécurité fournit un mécanisme d'authentification, de cryptographie, etc. Dans la partie suivante, nous présentons les implantations majeures du modèle à composants. Les plates-formes décrites ne fournissent pas toutes les mêmes services techniques. Pour chacune, nous reviendrons sur les différentes solutions en terme de services techniques.

1.2.3 Modèles à composants et leurs implantations

La classification des modèles à composants est dictée par la modélisation trois tiers des systèmes d'information. Ainsi on distingue couramment deux grands types de modèles à

composants : les modèles à composants de présentation (ex : JavaBeans) et les modèles à composants métiers (ex : EJB, CCM, .NET), la partie donnée étant gérée par les services techniques des composants métiers. Cependant, un troisième type de composants est récemment apparu : les modèles à composants qu'on pourrait dénommer "génériques" (ex : Fractal, ArcticBeans, Avalon).

Composants de présentation

Les modèles de composants de présentation sont principalement dédiés au développement rapide d'interfaces graphiques et n'implantent pas les notions de conteneur. Le représentant principal de ce type de modèle est le modèle des JavaBeans ([jav], [DEM 03]) : principalement dédiés aux applets et aux applications côté client en Java, ils sont connectés dynamiquement sur un modèle d'émetteur/récepteur d'événements et garantissent les propriétés de persistance, d'introspection et de configuration. Ces composants sont développés selon un moule assez restrictif mais qui permet de les manipuler facilement avec des outils visuels. Ce type de modèle ne nous intéresse pas particulièrement, dans la mesure où nous nous intéressons plutôt à la partie "logique applicative" et à l'extension de l'utilisation des services techniques dont ce type de composants ne fait qu'une utilisation limitée.

Composants métiers

Les composants métiers sont des composants logiciels gros grain, destinés aux systèmes d'information distribués. Afin d'en faciliter le développement, la distribution et la réutilisation de ces composants, leur conception sont basées entre autres sur le principe de séparation code applicatif (ou fonctionnel) du code technique (ou non-fonctionnel). Cela se traduit par le fait que lors de l'exécution d'une méthode d'un composant, des services techniques tels que la distribution (ORB pour les communications synchrones ou bus à message pour les communications asynchrones), la persistance, la sécurité et d'autres sont mis en œuvre par l'intermédiaire du conteneur de façon transparente. Ce code technique étant développé à part du code applicatif et étant manipulé par le conteneur, l'application écrite par le développeur ne contient pas ou peu de code relatif à ses services.
On distingue principalement trois modèles à composants métiers les plus utilisés dans l'industrie : EJB, CCM et .NET. Chacun de ces modèles correspond à une vision particulière de la programmation : les composants Corba Component Model (CCM) de l'OMG sont destinés au développement d'applications à grande échelle et fiables pour les entreprises. Les composants Entreprise JavaBeans (EJB) de Sun proposent une solution pour un développement rapide d'applications déployées sur Internet. Quant aux composants .NET de Microsoft, ils permettent de développer des applications "de bureau".

Enterprise JavaBeans de Sun La spécification des EJB définie par Sun ([EJB01], [MON 01]) autour des technologies Java a de nombreuses implantations aussi bien commerciales qu'en libre source : JonAs [jon], JBoss [JBoa], BEA WebLogic, IBM WebSphere, Oracle9iAS, etc. En effet, la spécification EJB ne précise par la manière d'implanter un composant mais plutôt un moule de conception utilisé par le développeur pour définir des

blocs fonctionnels et des interfaces précises. Le composant obtenu est réutilisable dans n'importe quel environnement Java et configurable de façon limitée mais simple selon l'environnement : il est possible de définir la politique de certains services techniques au moment du déploiement. La gestion de ces services techniques est déléguée au conteneur du composant qui repose sur un serveur EJB (souvent dans les implantations le conteneur et le serveur ne sont pas distincts).
Ainsi, le modèle EJB définit peu d'abstraction, il fournit plutôt une solution fermée (en Java avec un nombre limité de services techniques) mais facile et rapide à implanter pour des applications spécifiques tels que les applications de commerce électronique.

CORBA Component Model de l'OMG Les implantations du modèle CCM [CCM99] ne sont pas nombreuses : OpenCCM , EJCCM et MicoCCM. Le modèle CCM définit beaucoup plus de concepts abstraits et prend en compte le cycle de vie entier du composant, de la conception à l'exécution en distinguant :

- modèle abstrait qui définit la notion de composant et l'indépendance entre la description des interfaces et l'implantation dans divers langages de programmation),
- modèle de programmation qui décrit entre autres un langage de description des propriétés techniques,
- modèle de déploiement qui permet de décrire assemblage et déploiement ainsi que de fournir des outils d'administration,
- modèle d'exécution qui définit le comportement du conteneur.

Ce modèle repose sur la plate-forme CORBA de l'OMG pour la gestion des requêtes et pour les autres services techniques qui sont plus nombreux que dans le modèle EJB.
Ce modèle offre une solution abstraite de haut niveau, dont de nombreux concepts sont réutilisables pour n'importe quel type de composants. Il définit une solution moins spécifique que celle fournie par les EJB, avec notamment des conteneurs de composants plus "génériques". Cependant les implantations de ce modèle sont très récentes et ne fournissent pas toujours toutes les fonctionnalités attendues. De plus, ce modèle ne fournit pas de guide strict pour la conception de composants, ce qui ne facilite pas forcément le travail de développement.

.NET de Microsoft Dernière génération de composants Microsoft initialement destinés au développement d'applications sous le système d'exploitation Windows, les composants .NET ([THA 01]) définissent un modèle indépendant de l'OS qui devrait permettre leur implantation sous d'autres environnements, tels que Linux, grâce notamment à l'utilisation de CLR (common language runtime), intégrant les différents standards du domaine (http, XML, SOAP, WSDL, UDDI). Ce modèle vise à unifier la programmation des produits "de bureau" et les services Internet ainsi que les produits destinés aux serveurs vendus par Microsoft. Un composant .NET a accès aux services fondamentaux de la plate-forme tels que le service d'authentification .NET Passport, la gestion des exceptions ou un service d'alerte.

Services techniques dans les modèles métiers Dans les modèles à composants décrits ci-dessus, les services techniques gérés par le conteneur sont en nombre limité et leurs

	EJB	CCM	.NET
transaction	JTA / JTS	OTS / Concurrency Service	MTS
persistance	JDBC / JCA / JDO	Persistent State Service	ADO .NET
nommage / courtage	JNDI	Naming Service / Trading Service	LDAP / UDDI
sécurité	authentification / autorisation	Security Service	Passport
événements	JMS	Event Service	service intrinsèque

TAB. 1.2 – *Services techniques dans les modèles à composants métiers*

API sont strictement définies (voir tableau 1.2) : les EJB utilisent les services spécifiés par Sun, les composants CCM réutilisent les services de CORBA (ex : transaction, sécurité, persistance, tolérance aux fautes, etc) et les composants .NET intégrent les services techniques des services web ainsi que de nouveaux services développés par Microsoft de façon spécifique. Malgré l'utilisation de services reposant sur les mêmes concepts et parfois les mêmes modèles, les plate-formes n'utilisent pas les mêmes implantations ni les mêmes API. Pour certains services techniques, il n'est possible de choisir ni l'implantation, ni la politique. Cependant une certaine liberté est tout de même laissée au déployeur de l'application dans la mesure où il peut définir, dans un cadre strict, les besoins du composant en terme de service technique. Il peut le faire de deux façons : choisir parmi des types prédéfinis de composants ou choisir au moment du déploiement la politique d'un service. Par exemple, un composant EJB peut être session, entité ou message. Ses services techniques ne sont pas gérés de façon similaire (ex : un composant session n'est pas persistant). De plus, un serveur EJB fournit :

- *un service transactionnel* : lui aussi associé à tous les composants, ce service fournit uniquement des transactions simples. Ce service est implanté de façon spécifique par le fournisseur de la plate-forme (en anglais "vendor") dans le conteneur du composant, il n'est donc généralement compatible qu'avec un moniteur de transaction particulier. Le développeur choisit si une transaction doit être associée à l'exécution de ce composant ou pas.
- *un service de persistance* : le développeur définit si ce service est géré par lui même de façon explicite dans le code ou par le conteneur du composant;
- *un service de nommage et courtage* : le service de nommage permet de retrouver grâce à son nom symbolique un composant, qu'il s'exécute ou non sur la même JVM (Java Virtual Machine). Le service de courtage fournit un service équivalent en prenant toutefois comme paramètre les propriétés du composant. On peut faire une analogie entre le service de nommage et de courtage avec respectivement les pages blanches et les pages jaunes de l'annuaire téléphonique.
- *un service de sécurité* : associé à tous les composants, il fournit les modèles d'authentification et d'autorisation préconisés par le kit de développement Java 1.2 (JDK1.2).

- *service de gestion de cycle de vie* : une implantation basique de ce service est fournie avec la plate-forme. Le développeur a la possibilité de le modifier : il peut ajouter des états et modifier les règles de passage d'un état à un autre. Ceci doit toutefois être fait au moment du développement du composant.
- *un service de gestion des évenements* : utilisé spécifiquement pour invoquer de façon asynchrone les "message-driven beans", il repose sur JMS qui définit une norme pour l'échange de messages. Ce service garantit la bonne délivrance des messages, leur durabilité ainsi que la gestion des problèmes réseaux sous-jacents. Enfin, l'utilisation de JMS permet une bonne interopérabilité avec les MOMs (Message Oriented Middleware).

On retrouve le même type de solution dans .NET et CCM que dans les EJB. Par exemple, pour le service de transaction, ces modèles laissent toujours le choix au développeur de l'application entre les transactions gérées par lui-même ou gérées de façon automatique par le conteneur, la première solution étant moins coûteuse en terme de temps d'exécution mais plus difficile de conception.
En conclusion, pour la plupart de ces services, le développeur exprime de façon explicite ses préférences dans des fichiers de déploiement. Cependant, ces personnalisations sont limitées : il n'a pas le choix du modèle ou de la version du service technique.

Composants génériques

Un nouveau type de modèle à composants émerge. On peut qualifier les modèles de ce type de modèles à composants génériques car ils ont pour but initial de fournir des composants pour la conception et le développement à la fois d'intergiciels, d'applications d'entreprise et de services web. Des modèles tels que Fractal ([BRU 04], [Fra]), Avalon ([ava]) ou ArcticBeans ([AND 01], [KAR 03]) résultent du retour d'expérience des premiers modèles à composants industriels. En effet, la conception à base de composants ayant eu du succès dans ce domaine, on a voulu l'étendre à un nouveau domaine, celui des intergiciels ainsi que proposer une solution unifiée pour les applications. Pour cela, on a défini des modèles fournissant plusieurs niveaux d'abstraction et implantant la notion de conteneur ouvert. Ainsi, ces modèles fournissent des composants à plusieurs niveaux de complexité, plus évolutifs grâce une grande flexibilité du conteneur. Ainsi, potentiellement, on peut définir une large gamme de composants : allant initialement des composants basiques, petits grains, sans services techniques, légers à exécuter et donc adaptés à la conception d'intergiciels jusqu'aux composants métiers, gros grains, avec services techniques. De plus, ces modèles introduisent la notion de composants hiérarchiques : composants qui se composent eux mêmes d'autres composants. Par extension, on peut concevoir des composants applicatifs métiers, utilisant des composants techniques plutôt légers. Cependant le développement de composants métiers n'étant pas le but premier de ces modèles et ces derniers étant très récents (par exemple le modèle Fractal date de janvier 2002), il n'existe pas dans la hiérarchie de composants proposée de tels composants. Si ces modèles sont bien adaptés, de par leur conception, à évoluer vers les composants métiers, il reste donc beaucoup de travail autour de la définition de l'utilisation et de la gestion des services techniques.

Fractal Le but du modèle à composants Fractal ([Fra], [BRU 02]) est d'offrir un cadre architectural global pour le développement d'applications à base de composants. Il est inspiré du modèle RM-ODP [ISO 95a], modèle pour la description haut niveau de système de traitement distribué ouvert. Contrairement à d'autres modèles tels que les EJB, CCM et Avalon qui offrent un modèle concret du composant, Fractal regroupe une hiérarchie ou une famille de modèles. On y distingue le modèle de traitement abstrait qui est la racine de la famille de modèle et définit peu de concepts (composant, contrôleur, contenu, signal, nom et valeur). Un composant y est défini comme la composition d'un contrôleur (ou membrane, à rapprocher de la notion de conteneur) et d'un contenu, et le contrôleur d'un composant comme l'incarnation du comportement de contrôle associé à ce composant. De plus, ce modèle définit un ensemble de propriétés de base du modèle dont les propriétés d'encapsulation, d'abstraction et de récursion. La récursion signifie que le contenu d'un composant est composé d'autres composants (appelés sous-composants) qui sont sous le contrôle du contrôleur du composant englobant. Le modèle est complètement récursif et autorise l'imbrication des composants à un niveau arbitraire, une composition pouvant être modifiée dynamiquement. Enfin ce modèle permet à un composant d'être partagé par plusieurs compositions de composants.
La hiérarchie des modèles de Fractal étant extensible, il est possible de lui ajouter d'autres modèles de traitements, de programmation ou d'ingénierie, par exemple le modèle de traitement concret et le modèle de programmation associé au framework Julia (i.e. l'implantation de référence de Fractal) qui peut être utilisé pour la programmation de composants Fractal en Java. Le modèle de traitement concret est un raffinement du modèle de traitement abstrait. Dans ce modèle, le contrôleur encapsulant le composant permet de contrôler la composition et les liaisons d'une composition de composants. En effet, ces liaisons sont explicites et accessibles. Ainsi le système a une représentation de lui-même.

ArcticBeans Le modèle ArcticBeans [AND 01] est basé sur le modèle à composants OOPP (Open-ORB Python Prototype [AND 02]) fondé sur le modèle récursif RM-ODP). Il a pour but l'adaptation des services techniques de sécurité et de transaction, pour cela il repose sur la notion de réflexivité. Chaque service (transaction et persistance) y est conçu sous la forme d'un MOP (Meta Object Protocol). Pour l'adaptabilité du service de sécurité, il propose le language Obol qui permet de décrire les besoins d'une application en terme de sécurité sous forme de programme plutôt que de fichiers de déploiements. Ce langage est exécuté par un module nommé Lobo qui est inclus dans le conteneur des composants OOPP afin de déterminer la politique de sécurité. Pour le service de transactions [KAR 03], ce modèle propose des mécanismes réflexifs qui permettent l'ajout de nouvelles politiques de transaction. Ces mécanismes vérifient si ces manipulations sur les polices de transactions sont corrects. Grâce à la vérification de la cohérence des transactions, le système permet de façon sûre à un composant d'avoir en cours plusieurs transactions concurrentes et de types différents.

Avalon Ce modèle développé par Apache pour le développement de ses outils pour le web promeut l'inversion de contrôle afin d'améliorer la sécurité et la séparation des préoccupations [ava]. Cela se traduit par le fait qu'un composant y est vu comme une entité

passive qui joue un rôle particulier exprimé à travers des interfaces. Le modèle définit plusieurs types d'interfaces : "configurable", "serviceable" (pour un objet qui utilise des composants), "initializable", "disposable", "stoppable". Aussi choisi comme base pour l'élaboration d'intergiciels et de système pour serveurs, Avalon a été par exemple utilisé pour développer OpenORB de CORBA. Comme Fractal, le modèle à composants Avalon est récursif mais il ne permet pas le partage de composants. De plus, il ne définit pas plusieurs niveaux d'abstraction du modèle ce qui induit une API minimale plus importante. En effet, dans Fractal, le noyau de l'API est minimale car il ne reprend que les concepts des plus haut niveaux d'abstraction. Ce qui n'est pas le cas d'Avalon.

Services techniques dans les composants génériques Comparativement au conteneur des modèles à composants métiers (EJB, CCM), le contrôleur proposé dans les modèles à composants génériques propose peu de services techniques, principalement la gestion de cycle de vie.
Le modèle Avalon ne définit pas la notion de service en tant que tel : le conteneur s'occupe du déploiement de l'ensemble des composants d'une application, mais ne fournit pas d'autres services.
Comme décrit dans la partie consacrée à ce modèle, ArcticBeans se focalise sur les services techniques. L'approche adoptée est de fournir un middleware fournissant les mécanismes nécessaires à l'ajout de services techniques sous forme de MOP. Une solution spécifique à chaque service est choisie, ceci ayant pour but de fournir une solution plus efficace. Aucune approche globale n'est proposée, il n'y a donc pas de cohérence dans l'adaptation des services techniques et la solution proposée n'est pas réutilisable pour un autre service.
Concernant le modèle Fractal, ne founissant que le service de gestion de cycle de vie, il laisse néanmoins une totale liberté quant à l'ajout de services techniques. Pour cela, il fournit des fonctionnalités minimales permettant l'interception d'appel entrant ou sortant sur un composant, grâce aux notions d'intercepteur et de sous-contrôleur. Un intercepteur peut être placé sur une interface client ou une interface serveur, il "intercepte" les appels de méthodes et effectue des appels sur les sous-contrôleurs qui lui sont associés. Les sous-contrôleurs fournissent différentes fonctionnalités d'introspection du composant (ex : sous-contrôleur de contenu du composant ou de ses liaisons, etc.). Fractal laisse toute possibilité d'ajouter de nouveaux intercepteurs et sous-contrôleurs mais ne donne pas de conseils quand à la conception des services techniques et leur utilisation.

1.2.4 Conclusion

La diversification des nouveaux terminaux s'accompagne d'une explosion des besoins applicatifs. Ces nouvelles applications doivent à la fois fonctionner dans des environnements spécifiques (machines nomades, sans fil, ou récepteur de télévision numérique, etc) et fournir à l'utilisateur une grande qualité de service (sécurité, tolérance aux pannes, etc). Le modèle à composants donne un cadre complet pour la conception d'applications distribuées et facile à faire évoluer. En effet, outre les caractéristiques positives des objets, un composant est conçu selon le principe de séparation des préoccupations, ce qui permet une conception modulaire de l'application, chaque module logiciel étant réutilisable sur n'importe quelle machine du réseau supportant la plate-forme et paramétrable selon les

besoins de l'application. Les modèles à composants métiers en particulier fournissent une solution pour des applications distribuées et robustes grâce aux services techniques qu'ils fournissent de façon transparente aux composants, sans surcharger le travail de développement.
Dorénavant, dans ce mémoire, le terme *application* désignera une application à base de composants logiciels (elle pourra aussi être désignée sous l'appellation d'*assemblage de composants logiciels*). Cela siginifie : une application composée d'un ensemble de composants logiciels dont la distribution est implicitement gérée par l'intergiciel sous-jacent et qui a recours aux services techniques fournis par cette même plate-forme. Cette application peut être a priori exécutée sur n'importe quelle machine capable de gérer l'intergiciel choisi.
Notons que dans cette définition d'une application, nous insistons sur la notion de service technique. En effet, comme nous l'avons vu précédemment, écrits indépendamment du code technique, ces services permettent d'apporter une plus grande qualité de service à l'application et d'adapter les applications à leur environnement. Ces services sont fournis par l'intergiciel sur lequel se repose la plate-forme à composants. Les intergiciels ont pour but de masquer l'hétérogénéité des machines sous-jacentes. Ceci facilite le développement des couches supérieure mais ne permet pas aux applications d'ajuster leur comportement à leur environnement.

1.3 Intergiciels

Notre travail se place dans le contexte des intergiciels, et plus particulièrement des intergiciels à base de composants. Conçus afin de combler le lacunes du modèle orienté objet, la programmation à base de composants introduit notamment la notion de services techniques sur laquelle se basent nos travaux.

1.3.1 Définition de la notion d'intergiciel

Afin de faciliter la conception, la programmation et la gestion d'applications distribuées, la notion d'intergiciel (en anglais middleware) fournit un cadre simple et cohérent de programmation ainsi qu'une infrastructure puissante pour l'exécution de ces applications.
Un intergiciel est une couche logicielle, souvent distribuée, situé entre les applications et le système d'exploitation, les protocoles et le matériel sous-jacents voir figure 1.4 (inspirée de [SCH 01], [GEI 01]). Il permet de s'abstraire de la complexité et de l'hétérogénéité de l'environnement d'exécution de l'application. Cet environnement se caractérise par l'hétérogénéité des protocoles de communication de ses réseaux, des machines (à la fois les terminaux, serveurs de traitement et serveurs de données), des systèmes d'exploitation et des langages et donc globalement de la qualité de service rendue. Afin de masquer cette hétérogénéité, l'intergiciel cache la gestion des protocoles de communication, du partage et de la réplication des données et des services, des pannes réseaux, etc.

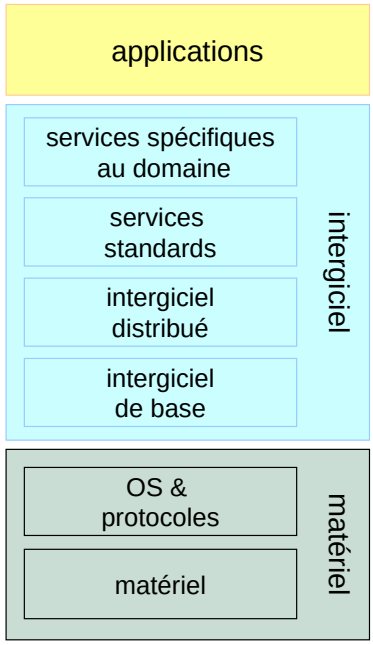

FIG. 1.4 – *Différentes couches du système*

1.3.2 Evolution des intergiciels

Les premiers intergiciels, destinés aux applications d'entreprise au sein d'un réseau assez fermé, avec des machines clairement identifiées bien qu'hétérogènes, avaient pour but de partager des informations entre des serveurs et des stations de travail en fournissant des outils d'invocation de méthode à distance et de gestion des fichiers ainsi qu'un annuaire de services.
Depuis, les applications utilisant les intergiciels ont évolué. Les applications d'entreprise sous plus ouvertes : elles doivent intégrer les données et services de leurs filiales, ainsi que ceux d'applications externes avec lesquelles elles coopèrent. Non seulement la taille des systèmes va en s'agrandissant mais ces systèmes doivent aussi permettre l'imbrication de nombreux modèles d'interaction, chaque participant à l'application voulant garder une certaine autonomie et avoir des preuves de la qualité du service rendu par ses partenaires. Parallèlement, un second type d'applications fait son apparition : les applications du web et les applications pair à pair. Celles-ci ont un nombre de participants très variable et difficilement prévisible. Malgré ces nouvelles contraintes, on demande à ces applications de garder une qualité de service élevée, comme par exemple un temps de réponse rapide. Les terminaux participant à ces applications sont non seulement hétérogènes mais il est difficile de les connaître a priori. De plus, les réseaux utilisés comme Internet ou les réseaux spontanés sont très sujets aux pannes. Enfin, il existe des problèmes de confiance entre les participants de l'application car ils sont difficilement identifiables et les réseaux pas forcement sécurisés.

1.3.3 Les tendances

Les premiers efforts de standardisation des intergiciels furent DCE [DCE], CORBA [COR95] et RMODP [ISO 95a]. Puis, la mise en œuvre de ces standards déboucha sur la conception des plate-formes génériques pour la gestion d'objets distribués CORBA et DCOM/COM+. Dernièrement, plusieurs tendances sont identifiées dans la recherche sur les intergiciels [dso] :

- les *intergiciels orientés service* tels que Jini ou OSGi qui ont pour but de faciliter la découverte et le déploiement de services, ils reposent souvent sur la notion d'annuaire,
- les *intergiciels orientés événement* bien adaptés pour les plates-formes sans contrôle centralisé, sont capables de surveiller le déclenchement d'événements et d'y réagir,
- les *intergiciels orientés message ou MOM* tels que IBM's MQseries [IBM] sont basés sur le principe suivant : l'expéditeur d'un appel de méthode n'attend pas la réponse du serveur pour continuer son travail. le récepteur n'a pas forcement besoin de fonctionner lors de l'envoi car les appels sont sauvegardés en attente de leur traitement.
- les *intergiciels orientés objet* comme CORBA ou DCOM/COM+, ils sont nés du besoin d'étendre le paradigme de la programmation orientée objet aux systèmes distribués. Ils étaient initialement basés sur l'interaction des objets grâce à des communications synchrones ainsi que des systèmes d'annuaires d'objets. Les communications synchrones ayant montré leurs limites, ces systèmes intègrent des services étendant les protocoles d'interaction vers les communications asynchrones. Une des évolutions de ce modèle est représentée par les *intergiciels à base de composants* comme CCM, EJB, .NET. Ces intergiciels offrent à la fois un cadre de conception qui accroît la réutilisabilité et l'évolutivité des applications mais aussi une plateforme d'exécution qui simplifie l'utilisation des services techniques. Ces services techniques, utiles à la plupart des applications quelque soit leur domaine, ne leur procurent pas de nouvelles fonctionnalités mais une meilleure qualité de service.
- les *intergiciels réflexifs* basés sur le concept de réflexion, ils ont pour but d'améliorer la portabilité, la (re)configurabilité grâce à leur flexibilité et leur adaptabilité.

Ces différents types d'intergiciels ne s'excluent pas les uns les autres : un intergiciel peut appartenir à plusieurs de ces catégories. Dans les intergiciels orientés service, événement et message, on s'intéresse plus aux interactions entre les modules de l'application. Primitivement, les intergiciels orientés objet ou composant définissent un modèle de conception d'application. Quant aux intergiciels réflexifs, ils regroupent des intergiciels dont les concepts intègrent le principe de réflexion.

Par exemple le modèle Fractal, que nous avons choisi comme modèle de référence pour cette thèse, est à la fois orienté composant et réflexif [BRU 04]. Il définit un modèle à composants hiérarchique, où un composant est une composition de composants et extensible car un composant Fractal peut avoir plusieurs niveaux de complexité. Ce modèle est aussi réflexif dans la mesure où les composants ont des capacités d'introspection, c'est-à-dire qu'ils ont une représentation d'eux-même et peuvent raisonner dessus, et des capacités d'intercession, c'est-à-dire qu'ils savent modifier leur propre état.

1.3.4 Notre problématique concernant les intergiciels

Notre but dans cette thèse est d'apporter aux nouvelles applications de l'informatique distribuée une solution pour qu'elles puissent s'exécuter dans des contextes hétérogènes avec une qualité de service optimale. Nous voulons donc étudier les caractéristiques d'un intergiciel capable de s'adapter à la qualité de service fournie, tout en dissimulant la complexité de l'environnement d'exécution des applications au développeur. Pour cela, nous voulons définir une solution générique pour la conception, le développement et l'exécution de services techniques s'adaptant à leur environnement d'exécution et aux besoins des applications. Ces services devraient s'adapter tout en maintenant l'esprit de la programmation par composants qui a pour but de faciliter le travail du développeur et en ne surchargeant pas non plus le travail de l'administrateur système.
C'est pourquoi dans cette thèse, nous nous intéresserons à deux types d'intergiciels : les intergiciels à base de composants et les intergiciels réflexifs. Le modèle à composants offre une solution efficace pour la conception et la mise en œuvre des nouvelles applications distribuées. Cependant, il ne permet pas de gérer de façon totalement optimale l'hétérogénéité des contextes d'exécution. C'est le problème auquel tente de répondre le second type d'intergiciels que nous étudierons : les intergiciels réflexifs.

1.4 Besoin d'adaptabilité des services techniques

Dans les sections précédentes, nous avons défini notre cadre de travail : de nouvelles applications, conciliant la diversité des environnements d'exécution des applications légères type application nomade et la grande qualité de service des applications d'entreprise. Nous avons aussi vu combien le modèle à composants est un modèle adapté au développement de ce type d'applications. En effet, il est possible à la fois tirer partie de la force d'abstraction du modèle à composants et se reposer sur des implantations des modèles à composants fournissant de puissants services techniques.
Nous avons vu que la diversification des outils informatiques, ouvrant de nouvelles possibilités, a créé de nouveaux domaines d'applications. Ces applications ont de nouvelles caractéristiques telles que leurs contextes d'exécution hétérogènes et leur besoin d'une grande qualité de service. Or ces applications sont de plus en plus souvent écrites grâce au modèle à composants permettant notamment une grande réutilisabilité et une gestion simple de la distribution de l'application. Ainsi les intergiciels à base de composants métiers qui étaient utilisés jusqu'alors presque exclusivement sur des serveurs dédiés à leur exécution, doivent maintenant être exécutés dans des environnements pour lesquels ils n'étaient pas initialement prévus. Il est donc nécessaire à présent de fournir des intergiciels robustes et adaptables quel que soit leur environnement d'exécution [DUR 04] : qui fonctionnent en mode dégradé si les ressources sont faibles et qui, au contraire, exploitent les ressources si elles sont importantes, apportant une plus grande qualité de service notamment grâce à leur personnalisation. Il est aussi nécessaire que l'offre en terme de service technique réponde aux besoins particuliers exprimés par les nouvelles applications.
Dans ce chapitre, nous détaillerons comment l'environnement d'exécution d'un service

technique et les besoins de l'application à base de composants en services techniques devraient conditionner l'offre en terme de services techniques. Puis nous récapitulerons les différentes possibilités d'adaptation nécessaires pour faire varier cette offre. Enfin, nous étudierons les limites des services techniques dans les modèles à composants usuels.

1.4.1 Influence de l'environnement d'exécution

Jusqu'à présent, parmi les plates-formes à composants, seules les plates-formes à composants métiers offraient des services techniques. Les modèles à composants clients ou "légers", s'exécutant sur les terminaux, n'en bénéficient pas. Or les composants métiers sont conçus pour fonctionner dans des contextes d'exécution robustes type serveur. Cependant, on assiste à une évolution car on aimerait pouvoir exécuter des composants métiers sur des machines de type terminaux, par exemple dans les applications P2P. Ces terminaux doivent donc être en mesure d'exécuter les services techniques tels que la persistance, les transactions, ou encore la sécurité. Cependant, contrairement aux machines serveurs, les terminaux, notamment les terminaux nomades ont des caractéristiques qui fluctuent beaucoup durant l'exécution (ex : bande passante, services disponibles, puissance CPU disponible, etc). De plus, d'un terminal à un autre, ces capacités sont très hétérogènes.
Cela a une influence sur l'exécution des applications et des services techniques qui s'exécutent dans ces environnements variables. Même si, grâce aux intergiciels, le code des applications et des services techniques est simplifié, l'intergiciel ne peut masquer les variations de qualité de service des couches matérielles sous-jacentes. Ces différents terminaux ne sont pas toujours capables d'exécuter la version "basique" des services techniques qui est fournie avec la plate-forme. En effet, ces services peuvent parfois requérir des grandes capacités de la part des machines qui les exécutent. Cela a pour conséquence les variations de qualité de service rendue qui peut même aller jusqu'à la non exécution du service.
Prenons pour exemple le service de sécurité : il est nécessaire à l'exécution en toute confiance des composants sensibles en fournissant les mécanismes d'authentification et d'autorisation. Ainsi le système garantit l'intégrité des données ou des demandes de service, leur origine et leur non divulgation à un tiers non autorisé. Le service de sécurité est déployé à la fois du côté serveur et du côté client. Plusieurs niveaux de sécurité peuvent être fournis par ce genre de service. Dans le domaine de l'authentification, un utilisateur peut simplement s'identifier à l'aide d'un mot de passe et d'un identifiant. Il est possible aussi d'ajouter à ce système de base la gestion de groupe d'utilisateurs. Dans le domaine de l'intégrité et du secret des données, le développeur peut utiliser des techniques perfectionnées comme la cryptographie. Dans l'application décrite dans la partie 1.1.3, certains composants sensibles nécessitent l'utilisation du service de sécurité, comme par exemple le composant de paiement lors de la validation du pari. Ce composant est réparti du côté client, il effectue la gestion de la carte de crédit, et du côté serveur il prend contact avec l'organisme bancaire. La partie du service de sécurité se trouvant du côté client effectue des traitements plus ou moins complexes comme la cryptographie. Ce genre de traitement est assez coûteux en terme de calcul. En fonction de la machine que le client utilise, il n'est donc pas possible d'avoir le même niveau de sécurité. Sur des terminaux ayant de faibles capacités de traitement (assistants digitaux personnel, téléphone GSM), l'utilisateur bénéficiera donc d'un service de moindre qualité ou plus lent, au contraire, sur un

ordinateur personnel, l'utilisateur aura un service très efficace. Cet exemple montre que l'application, en fonction de son environnement d'exécution a des besoins différents en terme de service technique.

1.4.2 Influence des besoins spécifiques des nouvelles applications

Nous avons vu, dans la partie précédente, que l'offre de la plate-forme en terme de services technique devrait dépendre de l'environnement d'exécution. Un second facteur entre en jeu : les besoins du composant applicatif en terme de services techniques. Selon son domaine applicatif ou encore ses besoins ponctuels, une application n'a pas toujours besoin des mêmes services techniques. Les services techniques sont écrits pour fédérer une partie du code transverse aux applications avec une grande expertise. Donc certains de ces services sont conçus pour une utilisation assez pointue.
Prenons pour second exemple le service transactionnel ([HéR 04c], [KAR 03]) qui a pour but de rendre fiables les applications dans des domaines tels que l'informatique hautement distribuée ou des applications sensibles (ex : gestion d'un hôpital). L'utilisation d'un service transactionnel garantit la cohérence des données, la reprise sur panne et le contrôle d'accès concurrents. Or, il existe différents modèles de gestion des transactions (plates, imbriquées, à flot de tâches, avec des contraintes de temps, pour les applications mobiles, etc.) et chaque modèle transactionnel s'utilise dans des contextes applicatifs bien précis [JAJ 97], sur des terminaux ayant des capacités spécifiques. Une même application peut même avoir des besoins qui changent au cours de l'application. Dans notre application, un parieur, à la maison, veut connaître les cotes des chevaux pour une course. Il exécute une transaction simple pour récupérer les informations qui sont centralisées sur le serveur de l'entreprise de paris. Ensuite, il a besoin d'informations sur la santé des chevaux ainsi que leurs cotes et les horaires des courses. Il exécute une transaction plus complexe à longue durée de vie pour réunir des informations qui sont distribuées : sur les serveurs de l'entreprise de paris, sur le serveur du journal spécialisé dans les courses et sur le serveur de l'entreprise qui gère les courses. Enfin, il va assister à la course, utilise son PDA pour consulter les dernières informations de façon urgente. Il exécute alors une transaction mobile, qui ne lui garantit peut être pas l'exactitude de l'information mais la fournit rapidement. Cette adéquation entre les besoins de l'application et les services techniques est réellement au centre des préoccupations de la communauté intergiciel, comme le prouve le développement d'une plate-forme spécifique à son environnement telle que real-time CORBA [rea] qui fournit des services temps-réel pour les environnements ayant ce genre de contraintes.

1.4.3 Adaptation nécessaire à 4 niveaux

Nous identifions quatre niveaux d'adaptation spécifiques aux services techniques :

– Tout d'abord, tous les composants applicatifs n'ont pas besoin de tous les services techniques. Une première adaptation consiste donc à ne fournir au composant applicatif que les services techniques dont il a besoin. Dans les modèles à composants usuels, ceci est possible uniquement pour certains services.

- De plus, si l'on considère qu'un service technique regroupe plusieurs fonctionnalités (comme par exemple le service d'annuaire qui regroupe nommage et courtage, ou celui de sécurité qui regroupe authentification, cryptographie, etc), le composant applicatif n'a pas forcement besoin de toutes ces fonctionnalités. Notons qu'il ne serait pas judicieux de les dissocier totalement, d'une part parce qu'il existe entre eux une certaine unité sémantique, d'autre part parce qu'ils partagent un certain nombre de données ou de fonctionnalités en commun.
- Chaque service technique peut par ailleurs être implanté de plusieurs façons (ou version), par plusieurs modèles. Par exemple, le service transactionnel [HéR 04c] peut implanter les transactions plates, imbriquées, à flot de tâches, avec des contraintes de temps, pour les applications mobiles, etc.
- Enfin, il serait possible de tirer avantage de la personnalisation de certains services en offrant plusieurs vues d'un même service ou en la paramétrant. Par exemple, un service de persistance de données pourrait offrir des vues personnalisées d'une base de données en fonction des besoins de l'application, ou permettre de paramétrer la taille de la base de données en fonction de la place mémoire disponible.

1.4.4 Manque d'adaptabilité des services techniques dans les modèles à composants industriels

Dans les modèles à base de composants industriels tels que les EJB, CCM ou .NET, il n'existe pas de possibilité simple d'adaptation des services techniques. En effet, avant l'exécution de l'application, le choix d'utiliser ou non un service, le choix du modèle théorique ou encore celui de l'implantation de ce modèle théorique sont difficilement possibles. Les services techniques proposés dans ces plates-formes sont figés une fois pour toute, lors du développement de la plate-forme. Effectivement, ces modèles n'ont pas été conçus afin de répondre à cette problématique mais plutôt avec l'idée de simplifier le travail du développeur. Ceci se traduit par plusieurs facteurs qui empêchent l'adaptabilité des services techniques.
Pour un service donné (persistance, transaction, etc), il n'existe pas d'API unique; chaque modèle possède sa propre API. Afin d'éviter les problèmes de maintien d'interopérabilité entre deux composants fonctionnant sur deux ORB différents, on fixe le modèle de service technique. Il n'est donc pas possible que deux composants applicatifs sur deux ORBs différents utilisent des modèles de services techniques différents.
De plus, les services techniques sont référencés par les ORBs sous forme d'objets notoires : les classes à utiliser pour instancier les services techniques sont décrites dans des fichiers de déploiement, lors de l'exécution, il n'est donc plus possible de modifier l'implantation d'un service.
Enfin, la gestion des interactions entre les services techniques et leur ordre d'exécution sont totalement fixés par le modèle. On pourrait par exemple vouloir qu'un service de sécurité s'exécute avant tout autre service.
Ainsi, que ce soit de façon statique ou dynamique, ces modèles ne permettent pas l'adaptation des services techniques et ne tirent donc pas avantage des spécificités de leur environnement d'exécution.

1.4.5 Vers une solution unifiée pour des services techniques adaptés à leur environnement et aux besoins de l'application

Comme nous l'avons vu dans les exemples précédents, pour assurer un même service technique, il existe différents modèles théoriques, implantations, versions, fonctionnalités et chacun convient mieux à un environnement particulier ou aux besoins de l'application. Ainsi, un grand nombre de services techniques devra être adapté afin d'apporter la meilleure qualité de service possible pour chaque situation; c'est-à-dire profiter au mieux des ressources de l'environnement tout en tenant compte des besoins de l'utilisateur et de l'application.
L'adaptation des systèmes informatiques n'est pas un problème nouveau : on a créé des systèmes pour adapter les langages, les systèmes à base d'objet, de composants et dernièrement s'est même dégagé un nouveau champs de recherche, l'AOP, technique basée sur l'insertion de code à des points clefs. Dans la prochaine partie, nous allons étudier ces différentes propositions pour en dégager les techniques que nous pourrions retenir pour l'élaboration d'un mécanisme de gestion de l'adaptabilité des services techniques pour plate-forme à composants.

2

Etat de l'art et Positionnement

2.1 Concepts et projets autour de l'adaptabilité

Ce chapitre est consacré à un état de l'art des systèmes à base de composants adaptables. Comme souligné par F. Duclos [DUC 02a] et P.K. McKinley and al. [MCK 04], la plupart des travaux de recherche menés autour des systèmes intergiciels adaptables sont à la croisée de trois concepts : la réflexivité, la séparation des préoccupations et la programmation par composants (voir figure 2.1 tirée de [MCK 04]). Ces trois concepts de programmation fournissent les outils nécessaires à la conception de système adaptables génériques avec un bon niveau d'abstraction.

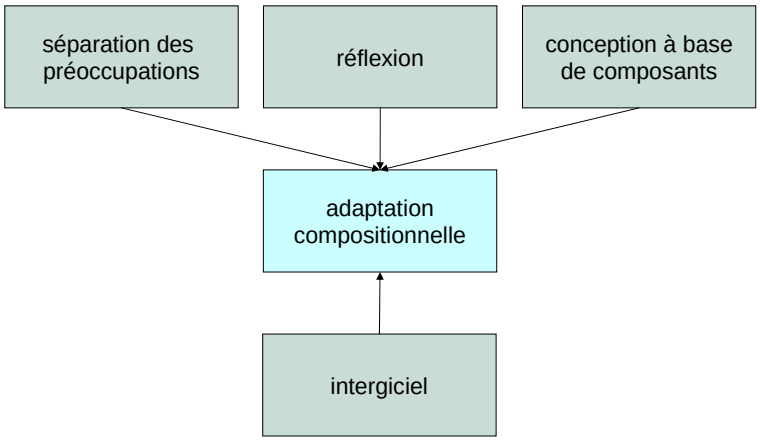

FIG. 2.1 – *Principales technologies pour l'adaptation compositionnelle*

– L'adaptabilité d'un système est souvent mise en oeuvre grâce au concept de réflexivité car un système réflexif a une connaissance et est capable d'agir sur lui même. Le concept de réflexivité, largement appliqué dans le domaine, est à la base de notre proposition.
– La séparation des préoccupations est un principe de conception d'application qui vise à dissocier les différents éléments de code transversaux d'une application puis

à les associer de façon claire et explicite. La séparation des préoccupations apparaît à différents niveaux : séparation des différentes fonctions, séparation du code fonctionnel du code technique. Elle a pour but de simplifier le développement et la maintenance de l'application et améliore la réutilisabilité du code. Ce concept est mis en œuvre dans de nombreux domaines tels que la programmation par aspects ([KIC 97b], [PAW 04], [AOS]), les filtres de composition ([M.A 93], [AKS 94], [BER 01], [Coma]), la programmation adaptative ([LIE 94], [LIE 95],[MEZ 98],[MEZ 00]).

– Bien que ne se réclamant pas du domaine de la séparation des préoccupations, la programmation à base de composants favorise un certain niveau de séparation des préoccupations. Elle fournit les concepts et les outils pour une programmation modulaire. L'approche par composants couvre tout le cycle de vie de l'application et permet aux différents intervenants de la chaîne de production d'exprimer leur point de vue. De plus, dans les modèles à composants métiers, on sépare les préoccupations fonctionnelles des préoccupations techniques.

Dans les parties suivantes, nous expliquerons la notion de réflexivité. Ensuite, nous dégagerons les grands domaines de la séparation par préoccupation et évoquerons d'autres techniques d'adaptation appliquées au domaine des intergiciels telles que l'utilisation des patrons ou encore la transformation de code. Nous détaillerons les différents projets visant à fournir des systèmes adaptables. Nous les avons classés selon qu'ils ont pour but d'adapter l'intergiciel lui-même, le code applicatif ou le code technique. Enfin nous positionnons nos travaux par rapport aux notions présentées.

2.1.1 Réflexivité

Un système réflexif est un système qui incorpore une représentation de lui-même qui lui permet de raisonner et d'agir sur lui-même ([MAE 87], [SMI 84]). La réflexivité permet à un système d'exposer certains de ses détails de fonctionnement, choisis à propos, sans pour autant en réduire sa portabilité. Elle permet de modifier son comportement améliorant ainsi son adaptabilité.
La réflexivité repose sur les concepts de réification (ou introspection) et d'intercession. La réification (ou introspection) est, pour un système, l'action de représenter et d'exposer sa structure interne en terme d'entités de programmation durant son exécution. L'intercession permet d'agir sur cette représentation et de modifier le comportement du système.
On distingue deux types de réflexivité [KON 02] :

– *comportementale* : capacité d'un système à fournir une représentation complète de sa propre sémantique en terme d'aspects internes de son environnement d'exécution. Elle permet de modifier le comportement des processus de l'environnement sous-jacent, en prenant en compte les propriétés non-fonctionnelles et la gestion des ressources. Par exemple, une application distribuée réflexive peut choisir d'utiliser un protocole de communication qui est bien adapté à son environnement d'exécution ;
– *structurelle* : capacité d'un système à fournir une réification complète de lui-même en terme de structure : hiérarchie des objets, connexion des objets, état, type. Grâce à la réflexivité structurelle, il est possible de modifier les fonctionnalités du programme. Par exemple, un objet réflexif peut donner des renseignements sur les

méthodes qu'il fournit.
Dans un système réflexif, on distingue deux niveaux (voir figure 2.2). Un niveau de base qui s'intéresse à la logique applicative, et un niveau méta qui décrit le traitement réflexif. Les deux niveaux sont dépendants : toute modification de l'un des niveaux se répercute sur l'autre.
Dans les systèmes réflexifs orientés objets, les objets représentant le traitement réflexif sont appelés des méta-objets. Le protocole d'interaction supporté par les méta-objets est appelé "meta-object protocol" (MOP) [KIC 91]. La réflexivité n'est pas un concept nouveau,

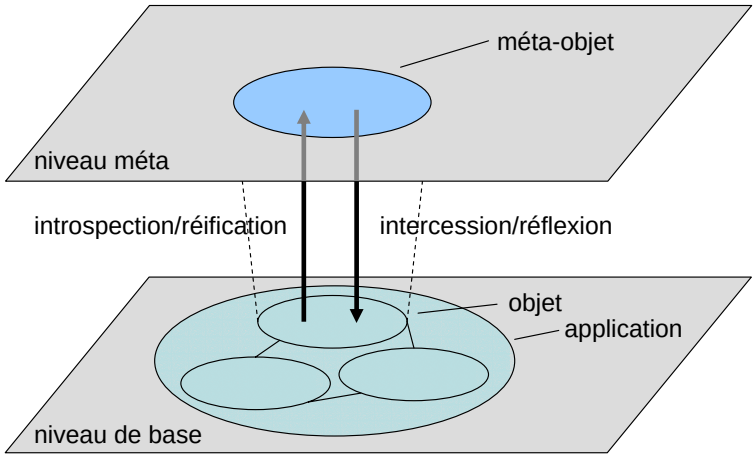

FIG. 2.2 – *Système réflexif.*

en effet il fut défini par Brian C. Smith, en 1982 [SMI 84]. Cependant, on s'est rendu compte qu'il pouvait être appliqué dans de nombreux domaines, dont les intergiciels. On peut citer [BLA 00], [VEN 02] et [MAS 92] qui proposent différents systèmes permettant l'adaptabilité grâce à la réflexivité.

2.1.2 Séparation des préoccupations

Les systèmes informatiques étant de plus en plus grands et complexes, les approches classiques pour le développement et la maintenance des applications ne suffisent plus. Parallèlement à l'approche par composant, l'approche par séparation des préoccupations (SoC, Separation of Concerns) a émergé.
Elle fut évoquée pour la première fois par D. Parnas en 1972 dans [PAR 72]. Puis, Dijkstra introduisit véritablement le terme en 1976 dans [DIJ 76]. Depuis de nombreux travaux reprennent cette approche [LOP 95], notamment la Programmation Orientée Aspect (POA).
Le développement de l'approche par séparation des préoccupations est parti d'un constat : dans une application, on peut identifier une préoccupation principale dispersée et entrelacée avec plusieurs préoccupations qui lui sont transversales. Par exemple, dans le serveur Web Tomcat, l'analyseur syntaxique de XML est entrelacé avec une fonctionnalité de traçage [GRI].
Dans cette approche, l'idée générale est de diviser la complexité d'un problème en identifiant des sous-problèmes plus simples et faiblement couplés entre eux que l'on nomme des

préoccupations. Puis on recompose ces préoccupations afin de résoudre le problème initial. Les préoccupations peuvent être de différents ordres : préoccupations transverses (ex : qualité de service, distribution, tolérance aux pannes, etc) ou préoccupations fonctionnelles. Cette approche repose donc sur deux principes : la modularité et la composabilité. La séparation des préoccupations a pour avantage de simplifier le développement en rendant le code plus lisible et plus compréhensible, ce qui lui procure une plus grande robustesse. De plus, la maintenance est facilitée car l'étude du code est plus simple. Enfin cette approche améliore la réutilisabilité et l'adaptabilité car le couplage entre les préoccupation est faible et explicite.
Aujourd'hui, le représentant le plus important de ce domaine de recherche est la Programmation Orientée Aspect, auquel nous consacrons la section suivante. Autour de la POA gravite un certain nombre de travaux de recherches dont le but est de permettre l'adaptation du code par séparation des préoccupations : les filtres de composition ([M.A 93], [AKS 94], [BER 01], [Coma]), la programmation adaptative ([LIE 94], [LIE 95], [MEZ 98], [MEZ 00]), la programmation générative (generative programming [CZA 00]), programmation intentionnelle (intentional programming [SIM 95]), la programmation orientée sujet (subject-oriented programming [HAR 93]). Dans les différents projets de recherche évoqués, la granularité d'une préoccupation peut être plus ou moins fine (objet, composant, aspect), selon que la séparation des préoccupations est appliquée à une classe, un composant, etc.

Programmation Orientée Aspect

La Programmation Orientée Aspect (POA, en anglais Aspect Oriented Programming) est l'application de l'approche de séparation des préoccupations la plus reconnue. Ses principes furent introduits par des chercheurs du XEROX PARC3, dont G. Kiczales, en 1997 [KIC 97a].Elle permet la séparation des préoccupations au moment du développement de l'application.

Principes de la POA La POA définit un aspect comme étant une entité logicielle qui capture une fonctionnalité transversale à une application (définition tirée de [PAW 04]). Parallèlement au développement des aspects, on définit des *points de jonction* (*pointcut* en anglais). Conceptuellement un point de jonction définit le moment de l'exécution de l'application où l'on veut appliquer un ou plusieurs aspects. Concrètement, il peut correspondre, par exemple dans AspectJ ([KIC 01], [GRI]), aux appels et réceptions de méthode. Il peut aussi intervenir au niveau d'un constructeur, une classe ou une interface, un attribut, un bloc de code, une instruction, etc.
Un aspect peut être associé à une ou plusieurs *coupes* (ou coupe transversale, en anglais *crosscut*) qui regroupent les points de jonctions. Bien que liés par définition, le point de jonction et la coupe sont de nature différente :
"Une coupe est un élément de code défini dans un aspect, alors qu'un point de jonction est un point dans l'exécution d'un programme." [PAW 04]

Afin de définir le comportement d'un aspect, on définit aussi des *codes advice*. Si la coupe définit où insérer l'aspect, le code advice définit ce que fait l'aspect. Chaque code

advice est associé à une coupe et donc à un ensemble de points de jonction auxquels seront associés le code advice. Il existe trois types de comportement :

- *before* : le code de l'aspect est exécuté avant la méthode associée au point de jonction,
- *after* : le code de l'aspect est exécuté après la méthode associée au point de jonction,
- *around* : le code de l'aspect est exécuté avant et après la méthode associée au point de jonction.

Après le développement du code de base de l'application (souvent selon les préceptes en programmation orientée objet mais ce n'est pas une obligation), des coupes et codes advices, un tisseur d'aspect (aspect weaver) greffe le code des aspects sur le code de base de l'application.
Ce tissage peut être statique ou dynamique.

- Le tissage *statique* est effectué par le compilateur qui prend en entrée le code de base de l'application ainsi que les aspects et fournit en sortie l'application étendue avec les aspects. Le tissage statique permet d'avoir de bonnes performances d'exécution cependant il ne permet pas aux aspects d'apparaître de façon distincte dans l'application finie. On ne peut donc plus ajouter, retirer ou changer les aspects après compilation. AspectJ fournit un tissage statique.
- Le tissage *dynamique* intervient pendant l'exécution. Il permet donc une plus grande flexibilité (ajout, retrait changement d'aspects) qui a cependant un coût car le tissage dynamique intègre en plus une phase d'adaptation qui consiste à préparer l'application pour quelle puisse recevoir les nouveaux aspects. JAC et JBoss AOP fournissent un tissage dynamique.

Implantations de la POA La section précédente présentait les concepts de la POA, ces concepts sont mis en œuvre dans des projets tels que AspectJ [KIC 01], JAC [PAW 03], JBoss AOP [JBoa], ApsectWerkz ou Prose.
La plupart des projets majeurs autour de la POA sont implantés en Java. Notons que ce n'est pas à cause d'une dépendance de la POA par rapport à un quelconque langage mais plutôt parce que le langage Java est très utilisé actuellement dans les applications distribuées.

- **AspectJ** [KIC 01] est une extension syntaxique du langage Java qui permet l'introduction de la notion d'aspect . Il permet un tissage statique des aspects par compilation.
  AspectJ fut crée en 1998 par G. Kiczales et son équipe de Xerox PARC. Depuis 2002, le projet a rejoint la communauté Open Source Eclipse.
  Le projet AspectJ est donc l'un des projets implantant les concepts de la POA les plus aboutis. Il fournit d'ailleurs de nombreux outils tels que le plug-in d'Eclipse AJDT qui permet de visualiser les points de jonction associés à une coupe.
  AspectJ a pour avantage de permettre un niveau de coupe très précis : on peut effectuer des coupes au niveau d'une méthode, d'un constructeur, attribut, exception, etc. Cependant, étant basé sur une extension du langage, AspectJ est limité au tissage statique, ce qui ne permet pas d'ajouter/retirer ou changer les aspects d'une application pendant son exécution. De plus, l'extension du langage est assez compliqué à maîtriser.

- **JAC** (Java Aspect Components) [PAW 03] n'est pas un langage, comme AspectJ, mais un framework. Ce framework open source implante l'API de l'AOP Alliance [aop] qui permet la définition des aspects, des coupes et des wrappers. Le mot wrapper est utilisé dans JAC et AspectWerkz, il est l'équivalent de code advice dans AspectJ ou intercepteur dans JBoss AOP.
  JAC fut conçu dès 1999 par les équipes CEDRIC du CNAM de Paris, le LIP6 de l'Université de Pierre et Marie Curie, le LIFL de l'université de Lille 1.
  JAC a pour avantage de permettre le tissage dynamique des aspects. De plus, il permet de configurer les aspects grâce à des fichiers de configuration afin que l'aspect soit adapté à des conditions particulières. Par exemple, un aspect implantant un service de transaction peut être configuré grâce à la signature des méthodes qui doivent s'exécuter dans une transaction. JAC a pour inconvénient que les points de jonctions y sont limités aux méthodes et aux constructeurs.
- **JBoss AOP** [JBob], comme JAC, est un framework qui permet d'ajouter dynamiquement à une application en Java des aspects en Java pur. C'est une extension du serveur d'application J2EE Open Source JBoss pour la programmation par aspect. Il permet d'associer à des EJB des aspects.
  Contrairement à JAC, les coupes sont exprimées en XML. On peut associer un point de jonction à une méthode, un constructeur ou un attribut. Comme dans JAC, on peut configurer les aspects grâce à des fichers de méta-données.
- **AspectWerkz** ([asp], [BON 04]) est un framework open source, sponsorisé par BEA. Il résulte de la synthèse de l'approche proposée par AspectJ, car très proche dans sa définition des coupes, et des frameworks JAC et JBoss AOP, car il permet l'adaptation statique et dynamique.
  AspectWerkz fut développé par Jonas Bonér et Alexandre Vasseur en 2002. Les aspects sont développés en Java pur à travers une API spécifique. Les coupes, sous la forme de fichiers XML ou de commentaires Javadoc spécifiques, sont associées à un constructeur, un attribut ou une exception.
  Notons que AspectJ et AspectWerkz devraient fusionner.
- **Prose** (PROgrammable extenSions of sErvices) [Pro] permet le tissage dynamique des aspects, il repose sur une extension de la JVM. Il permet des points de jonction au niveau des méthodes, des attributs et des exceptions. Les travaux autour de Prose sont menés depuis 2001 par G. Alonso au département d'informatique de l'ETH Zürich.

Il existe d'autres projets implantant les principes de la POA tels que DJ ([DJ], [ORL 01]), JMangler [jma], EAOP (Event-based AOP) [eao], CaesarJ [cae].

Conclusion Bien qu'ayant parfois des syntaxes ou un vocabulaire différents, tous les projets cités implantent les grands principes de la POA. Ces projets offrent une solution pour réutiliser le code transversale aux applications, comme les services techniques, grâce à des techniques de tissage très complètes. On peut tisser du code avant, après ou autour d'un appel de méthode, d'une lecture ou d'une écriture d'attribut, d'une levée d'exception, etc. On peut aussi exprimer l'ordre dans lequel on veut que les aspects soient exécutés.
Si AspectJ, historiquement première implantation, ne permet que le tissage statique, les

autres projets présentés permettent le tissage dynamique, souvent basé sur une manipulation de code Java. En plus de l'adaptation grâce aux aspects, des solutions telles que JBoss AOP et AspectWerkz ont pour avantage de permettre l'adaptation des aspects à travers des fichiers de configuration.
La programmation par aspect paraît donc être un bon candidat pour la conception de services techniques adaptables dynamiquement. Cependant nous relevons des limites tant au niveau de la conception que de l'exécution dans le cadre de notre travail.
La programmation par aspect fournit une philosophie de conception ainsi que des outils d'implantation. Cependant, ces outils sont basés sur la programmation orientée objet. Excepté dans les travaux menés par L. Seinturier et L. Duchien au LIFL de l'Université de Lille 1 ([PES 04], [BAR 02]) et sa collaboration avec N. Bouraqadi de l'Ecole de Mines de Douai [FAK 04], dans la majorité de ces projets, il n'existe pas la notion de composants comme nous l'entendons (même si le mot est parfois utilisé). On n'y voit pas apparaître les notions d'interface, de liaison. La plupart de ces solutions ne tirent donc pas partie de l'outils de programmation que pourrait être la programmation par composants.
De plus, afin de faire un choix entre les différentes implantations des services techniques, on aimerait en avoir une description. Ce niveau descriptif n'est pas présent dans les solutions présentées ou il se limite à l'expression des interfaces.
La POA a le potentiel pour fournir une solution pour l'adaptation des services techniques grâce à des techniques de tissage avancées mais qui seraient sans doute sous exploitées dans la mesure où, dans le cadre de la programmation par composants, les points de jonction des services techniques sont réduits aux appels de méthodes.
Pour la phase d'exécution, la POA donne des outils pour adapter dynamiquement mais pas auto-adapter. Les outils permettent de générer du code qui est adapté à l'environnement mais la génération n'est pas gérée par le système lui même. La programmation par aspect offre un cadre de conception très générique, convenant aussi bien aux aspects techniques que fonctionnel. Les services techniques y sont souvent assimilés à des aspects et il n'existe pas de distinction catégorique entre aspect technique et aspect fonctionnel. Il n'existe pas d'outils pour décrire les services techniques ou encore pour les retrouver ni pour représenter l'environnement d'exécution. Elle ne propose donc pas de solution complète pour gérer efficacement l'auto-adaptation des services techniques.

2.1.3 Le patron stratégie

Dans [GAM 95], E. Gamma propose un catalogue de patrons de conception qui permettent de simplifier la fabrication et l'évolution de programmes orientés objet. Le patron stratégie est un patron de conception (design pattern) comportemental qui permet l'adaptation de code sans que l'on ait besoin de modifier son client. Pour utiliser ce patron (cf figure 2.3), on définit une famille d'algorithmes (*StrategieConcreteA, B et C*) qui sont des variantes d'un même algorithme et qui sont encaspulée (par le constituant *Stratégie*). Dans ce patron de conception, les collaborations sont de deux types :

- entre *Stratégie* et *Contexte* : le *Contexte* peut offrir à la *Stratégie* une interface grâce à laquelle elle peut récupérer des données nécessaires à l'exécution de l'algorithme ou le *Contexte* peut se passer lui-même comme paramètre d'un appel à la *Stratégie*.

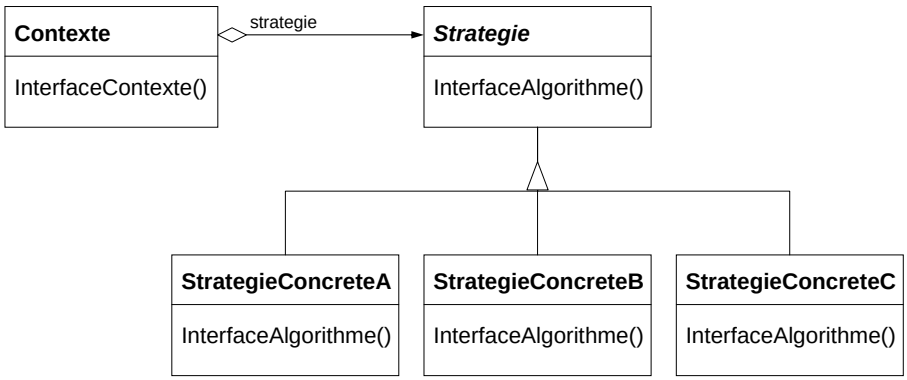

FIG. 2.3 – *Structure du patron de conception stratégie.*

– le *Contexte* transmet les requêtes de ses clients à la *Stratégie*.

Du point de vue de la conception, il est possible, grâce au patron Stratégie, de définir une famille d'algorithmes apparentés, pour lesquels l'héritage peut simplifier la réutilisation de fonctionnalités communes. Le recours aux déclarations conditionnelles (if, switch ou case) est limité car les différents comportements de l'algorithme s'expriment à travers plusieurs classes.

L'utilisation de ce patron permet l'adaptation dynamique du code en fonction de son contexte d'exécution. Cette adaptation est initiée par le client en fonction de critères tels que la vitesse d'exécution d'un algorithme. Son utilisation nécessite néanmoins que le client possède des connaissances sur les stratégies disponibles pour faire son choix. Elle peut engendrer une surcharge des communications entre Stratégie et Contexte dans la mesure où toutes les stratégies concrètes doivent implanter la même interface et que donc certaines des informations échangées ne seront pas forcement utilisées par les stratégies les plus simples.

Ce patron a été utilisé notamment dans les ORB TAO, ZEN, CIAO, DynamicTAO et UIC ainsi que dans le projet ACEEL que nous présenterons dans les sections suivantes.

2.1.4 Transformation de programme

La transformation de programme est l'une des techniques grâce auxquelles on peut adapter du code. Son principe consiste à modifier directement le code existant. Différentes techniques existent :

– La transformation au niveau du code source
  - Avant la compilation grâce à un pré-processeur, on produit du code source à partir de celui des classes et des préoccupations (ex : COMPOST [LUD 00], AspectJ [KIC 01]);
  - Durant la compilation grâce à un compilateur extensible, on agit sur l'arbre syntaxique pour ajouter les préoccupations au code des classes (ex : OpenC++ [CHI 95], OpenJava [TAT 99]);

  La transformation au niveau du code source permet une exécution du code assez

rapide mais elle fournit une approche statique qui n'est pas suffisante pour nos travaux. De plus elle nécessite de disposer du code source.
- La transformation du code intermédiaire (Javassist [CHI 00], JAC [PAW 03], Jabyce [LEN 04]) nécessite de disposer du code intermédiaire.
- La transformation du code binaire (ex : Detours [HUN 99])
  - Au moment du chargement;
  - Au moment de l'exécution du code grâce à la réflexivité;

  La transformation de code binaire fournit une approche dynamique qui permet de faire évoluer le code au cours de son exécution.
- La modification de la machine virtuelle (Prose [Pro], Guarana [OLI 99]) nécessite que l'utilisateur accepte ces modifications.
- La génération de code d'objets d'interposition est utilisée pour générer le code des conteneurs extensibles.

Le domaine de la transformation de code est très large. On retrouve donc ce concept dans de nombreuses solutions du domaine de la séparation des préoccupations, notamment en programmation orientée aspect. Toutes ces techniques demandent une grande connaissance de la part de l'utilisateur et l'on ne peut donc pas demander à un utilisateur de la plate-forme à composants de gérer leur utilisation directement. Par exemple, en POA on fournit aux utilisateurs les outils de tissage.

2.1.5 Travaux autour de l'adaptation

Le but des projets menés autour de l'adaptation est de proposer des systèmes informatiques, conscients de leur environnement d'exécution et qui tendent à améliorer la qualité de services qu'ils fournissent [GEI 01], [KON 02]. Pour cela, ils sont souvent basés sur la réflexivité et sur le principe de Séparation des Préoccupations.
Comme nos travaux, plusieurs projets tels que ArtcicBeans [Arc] ou JOTM [JOT], dont

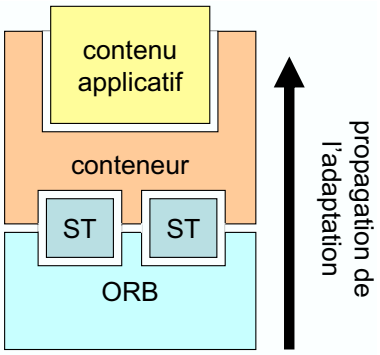

FIG. 2.4 – *Propagation de l'adaptation dans un système à base de composants.*

nous pouvons nous inspirer, s'intéressent directement à l'adaptabilité des services techniques. Nous aussi pouvons tirer profit d'autres solutions, qui bien que s'intéressant aux

autres éléments du système (voir figure 2.4), tournent aussi autour de l'adaptation des services techniques :

- L'**ORB** intègre la notion de service technique et son adaptation passe donc parfois par l'adaptation de ses services techniques;
- Les modèles à composants, qu'ils reposent sur un ORB ou pas (comme Fractal), peuvent intégrer la notion d'adaptation des services techniques à travers la notion de **conteneur** ouvert.
- Enfin l'adaptation des **applications à base de composants**, nous intéresse dans la mesure où nous proposons de voir un service technique comme une composition de composants et que, dans ce cas, l'application et le service technique ont la même structure.

Dans les parties suivantes, nous détaillons les principes de fonctionnement de certaines de ces solutions. La figure 2.6 est tirée de la thèse de Frédéric Duclos. Elle nous permet de comprendre les rapports existants entre les différents domaines (réflexivité, séparation des préoccupations, modèles de programmation par composants). Nous l'avons complétée avec le modèle à composant Fractal : celui-ci est réflexif, et prévoit l'ajout de services techniques. Les figures 2.5, 2.7 et 2.8, construites sur le même principe, montrent les rapports de la réflexivité et de la séparation des préoccupations (partie fixe du schéma) avec respectivement les ORB, les applications à base de composants et les services techniques. Nous y avons ajouté un espace "programmation par composants" pour représenter le fait que certaines solutions (i.e. ORB, application ou services techniques) sont développées à base de composants.

ORB adaptatifs

Les intergiciels tels que les EJB, CORBA, .NET permettent de masquer l'hétérogénéité et la distribution des ressources matérielles. Afin de répondre aux nouveaux besoins des systèmes temps réel, embarqués ou multimédia, on cherche à développer des intergiciels capables d'adapter la qualité de services qu'ils fournissent en fonction des ressources de leur environnement. Un intergiciel adaptable peut être porté sur de nombreuses plate-formes et fournit ainsi la transparence vis à vis de leur environnement nécessaire aux applications. Il existe donc de nombreux projets de recherche dans le domaine des intergiciels adaptatifs et il n'est pas possible de tous les étudier. Nous présentons les dernières recherches dans le domaine tels que DynamicTAO, UIC, OpenORB, OpenCORBA, Flexinet, etc.

- TAO [SCH 98] [TAO] est un ORB temps réel, conforme à la norme CORBA, à base de C++. Il est reconfigurable au démarrage grâce à des fichiers de déploiement. TAO améliore le service d'événement de CORBA en fournissant des mécanismes de planification et de distribution des événements en temps réel. TAO a été conçu par des chercheurs de l'équipe de F. Kon de l'Université de l'Illinois aux USA. Comme son successeur ZEN [KLE 02] et comme son implantation du modèle à composants CCM nommé CIAO [WAN 03], TAO n'est pas réflexif et utilise les patrons de conception "virtual component pattern" et "strategy pattern" de Gamma [GAM 95].

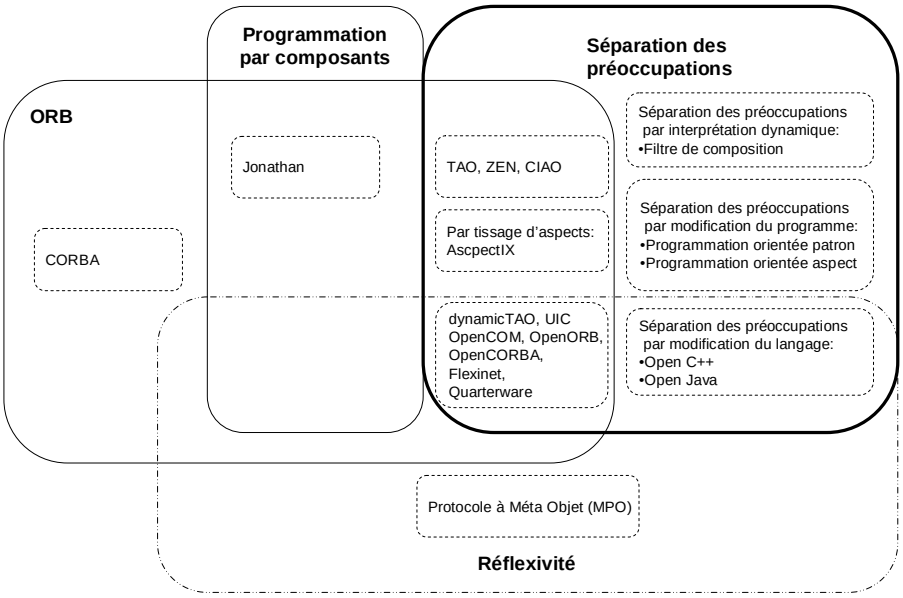

FIG. 2.5 – *Relations entre la réflexivité, la séparation des préoccupations et les modèles à composants et* ***les ORB****.*

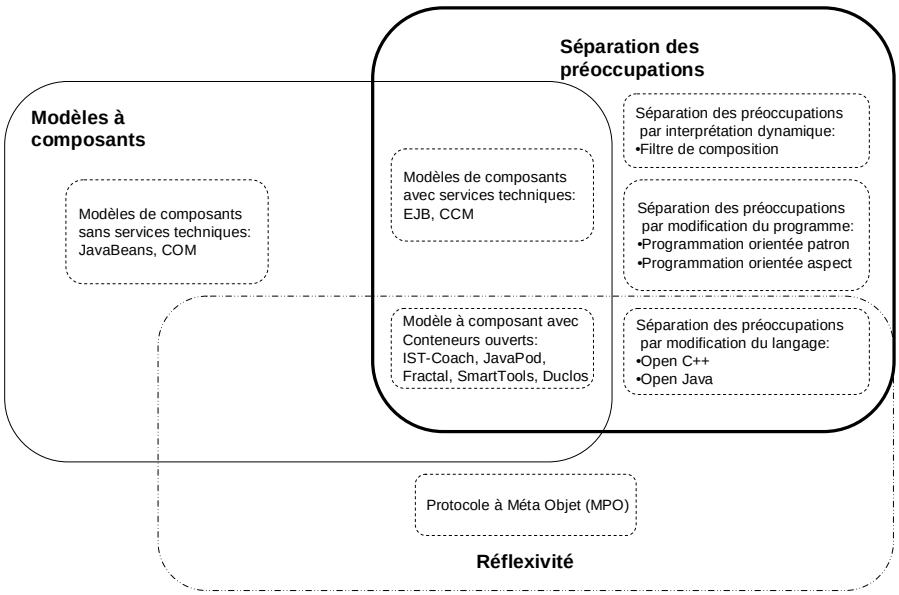

FIG. 2.6 – *Relations entre la réflexivité, la séparation des préoccupations et* ***les modèles à composants****.*

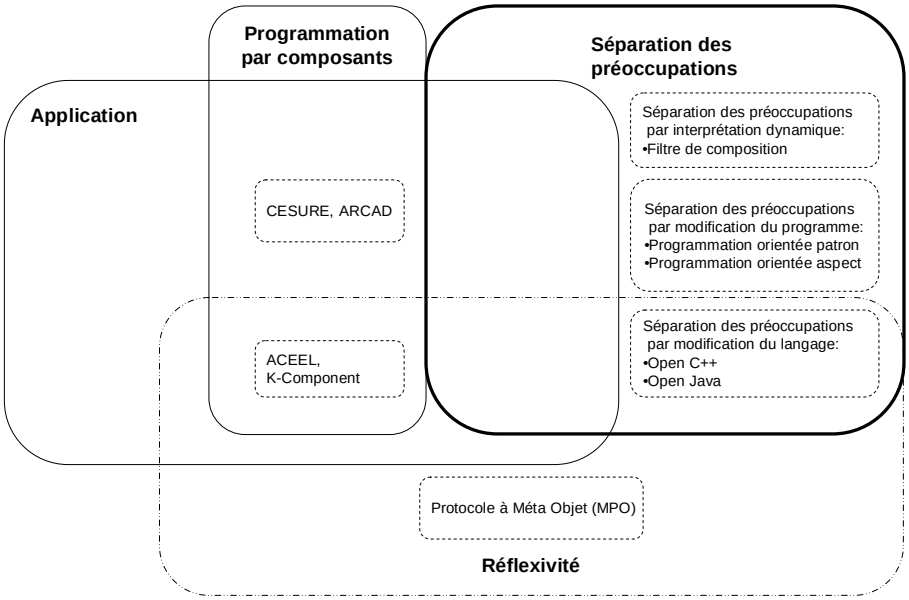

FIG. 2.7 – *Relations entre la réflexivité, la séparation des préoccupations et les modèles à composants et* ***les applications****.*

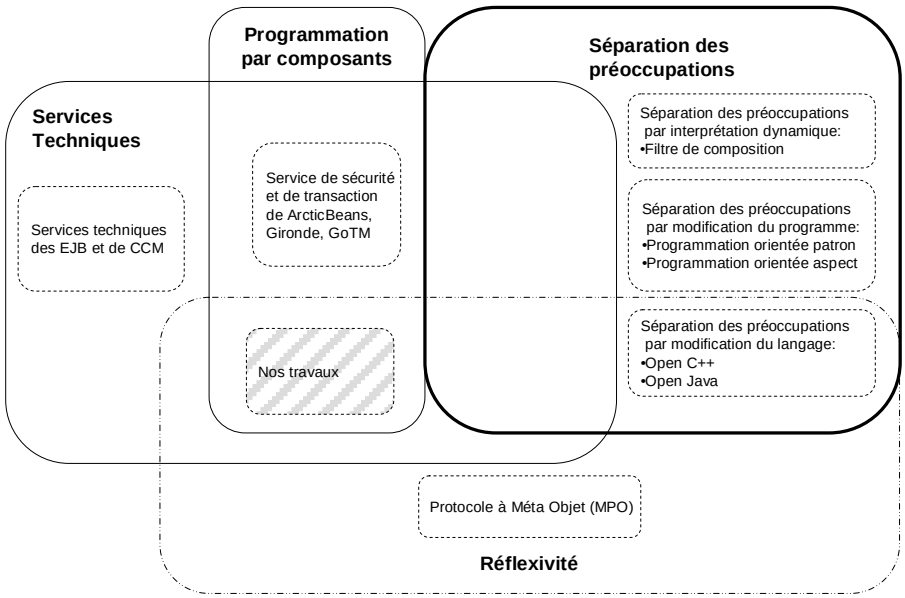

FIG. 2.8 – *Relations entre la réflexivité, la séparation des préoccupations et les modèles à composants et* ***les services techniques****.*

- Successeur de TAO, DynamicTAO [KON 00] repose sur la réflexibilité afin de permettre l'adaptation dynamique. Il fut développé entre 1998 et 2000 puis remplacé par LegORB [ROM 00] et UIC [ROM 01]. Selon les besoins, DynamicTAO peut adopter différentes stratégies pour implanter la gestion de la concurrence, le démultiplexage des requêtes, la planification ou encore la gestion des connexions.
- UIC (Universal Interoperable Core [ROM 01]) est un ORB générique destiné aux machines dont les capacités sont limitées. Il est constitué d'une base de taille limitée et de plusieurs personnalités telles que CORBA, Java RMI ou DCOM qui peuvent être adaptées soit statiquement pour des adaptations profondes avec des changements de fonctionnalités soit dynamiquement pour un ajout de fonctionnalités. Ainsi des objets UIC peuvent interagir avec des objets des différents standards tout en ne modifiant pas leur implantation.
- OpenORB [BLA 01] est destiné aux applications multimédia et mobiles. Il est développé depuis 2000 par les chercheurs de Exolab Group, organisation informelle travaillant au développement de logiciels d'entreprise open source. OpenORB est très flexible. Il permet à la fois des adaptations dynamiques profondes du système par réassemblage de ses composants grâce à un protocole à méta-objet mais aussi de petites adaptations au niveau d'un objet. Cet ORB n'est pas conforme à la spécification CORBA. Contrairement aux solutions présentées précédemment, il a pour inconvénient que ses adaptations ne sont pas transparentes aux applications.
  Le projet OpenORB a donné lieu à plusieurs travaux de recherche au sein de l'Université de Lancester tels que OpenCOM ([Opeb], [COU 04a]) implantation de OpenORB pour le modèle COM de Microsoft ou OOPP [AND 02] une implantation Python de OpenORB, OpenCCM [Opea] d'ObjectWeb mené par l'équipe GOAL du LIFL.
- OpenCORBA [LED 99] est un ORB CORBA en NeoClasstalk langage réflexif basé sur les méta-classes comme Smalltalk. Adaptable dynamiquement, il n'est pas capable de fournir une vision global de l'ORB comme le font DynamicTAO et OpenORB. Mais il permet, comme DynamicTAO de préserver un noyau de base autour duquel s'articule l'adaptation de certains mécanismes comme le mécanisme d'invocation ou de vérification de type.
- Flexinet [HAY 97] est un ORB CORBA adaptable dynamiquement proposé par Hayton et al. Il est écrit en Java et utilise la réflexivité et permet ainsi l'adaptation au niveau des interfaces.
- Jonathan est un ORB multi-standard en Java adaptable statiquement. Il fournit différentes personnalités qui sont décrites de façon statique. Jonathan fut développé par France Télécom R&D en 1996 et maintenant soutenu par Kelua et France Télécom R&D. Ce projet fait partie du consortium ObjectWeb. Jonathan est constitué d'un noyau minimal dont le but est de permettre la communication des composants. On peut ensuite ajouter les briques logicielles implantant les différents protocoles nécessaires. Jonathan fournit par défaut deux personnalités : David une implantation de CORBA et Jeremie qui permet de créer des applications à base de RMI, mais laisse la possibilité à chacun d'assembler son propre ORB.
- MobileJMS est un MOM implantant l'API JMS [KAD 04]. Il permet l'ajout ou

le retrait dynamique de services prédéfinis tels que les services de stockage et réémission, de compression et de fragmentation. Il permet aussi de changer l'ordre de ces services. Cette adaptation se fait en fonction des politiques des applications (contrat qui associe un ensemble de besoin de l'application avec une configuration particulière du contexte d'exécution), de la disponibilité des services et du contexte d'exécution (type de connexion réseau, bande passante disponible, etc). Elle est négociée entre les parties client et serveur. La négociation est faite au début d'un connexion JMS ou lors d'un changement d'environnement.

Il existe de nombreux autres projets d'ORB adaptables tels que Squirrel [KOS 01], AspectIX [GEI 98], Electra [MAF 95], Horus [REN 95], Isis [BIR 94], Orbix [Orbb], ORBacus ([ORBa], [ORB04]), JacORB [BRO 02], etc. L'article "A Taxonomy of Compositional Adaptation" de MCKinley et al. [MCK 04] définit et utilise une classification très complète de ces différents intergiciels.

L'étude de ces différents projets montre le large champs d'adaptation des intergiciels actuels. L'adaptabilité dynamique est souvent implantée grâce à l'ajout d'un niveau méta (ex : DynamicTAO, IUC, OpenCORBA, etc) qui permet d'avoir une représentation des éléments du système et de pouvoir ainsi les manipuler plus simplement.
De plus, on voit apparaître la notion de personnalité, configuration prédéfinie du système, par exemple dans UIC ou Jonathan. Ce genre de solution est souvent basée sur un noyau commun, dont l'adaptation est réalisée grâce à un réassemblage de composants ou à une réification au niveau des interfaces.

Cependant ces solutions n'intègrent pas l'idée d'un nombre indéterminé de services techniques [DUC 02a] et ne proposent pas véritablement de représentation adaptée des services techniques. Un service technique n'apparaît pas en tant que tel mais plutôt comme l'un des mécanismes de l'ORB. De plus, excepté pour MobileJMS, l'adaptation de l'ORB se répercute sur la totalité des applications qui s'exécutent au dessus de cet ORB. Ainsi, il n'est pas possible que deux applications, fonctionnant au dessus de cet ORB, bénéficient de services différents.

Conteneurs extensibles et ouverts

Jusqu'à présent les services techniques dans les modèles industriels apparaissaient sous la forme d'un nombre fixe d'objets notoires, c'est-à-dire des objets dont le système connaît, au moment du déploiement, le chemin d'accès qui ne peut ensuite plus changer. Ils sont appelés par le conteneur au moment d'un appel de méthode par l'intermédiaire de l'ORB. Le chemin pour retrouver ces objets notoires est fixé dans des fichiers de déploiement et il est impossible de le changer pendant l'exécution de la plate-forme [DUC 02a].
De plus, les appels aux différents services techniques, leur ordre et les paramètres passés étaient déterminés une fois pour toutes par le développeur du conteneur .
L'assemblage entre les composants applicatifs et les services techniques est prédéfini et n'apparaît pas de façon explicite dans le système : pour connaître les services techniques de la plate-forme, il faut se référer à la spécification du vendeur de plate-forme. Il en

résulte un manque de formalisation de la structure du conteneur.
Une façon de rendre l'assemblage plus souple (sans pour autant en fournir une représentation) est d'utiliser des conteneurs ouverts et/ou extensibles. Les conteneurs extensibles permettent l'ajout de services techniques au niveau d'un composant. Les conteneurs ouverts étendent cette idée en offrant une représentation explicite et formalisée des éléments du conteneur. Les modèles industriels n'en ayant pas la nécessité et par souci de rapidité d'exécution, il n'implantent pas ce type de conteneurs.

Projets Jacquard et IST-COACH Dans le cadre des projets Jacquard et IST-COACH [ist], P. Merle et M. Vadet redéfinissent la chaîne complète de description, de production, de déploiement et d'administration de conteneurs ouverts [VAD 01]. Leur vision d'un contrôleur est implantée dans la plate-forme à composant OpenCCM [Opea]. Ils proposent de faire coïncider la structure du conteneur avec ses rôles : celui d'interception, de coordination et de contrôle et que chacun de ces rôles soit rempli par un composant "système" (voir figure 2.9). Un composant de contrôle est associé à une fonction système (dont les services techniques font partie) et offre une abstraction au niveau supérieur de coordination. Les composants de coordination fournissent une vue cohérente des fonctions système à la couche d'interception. La couche d'interception réalise les indirections vers la couche de coordination.

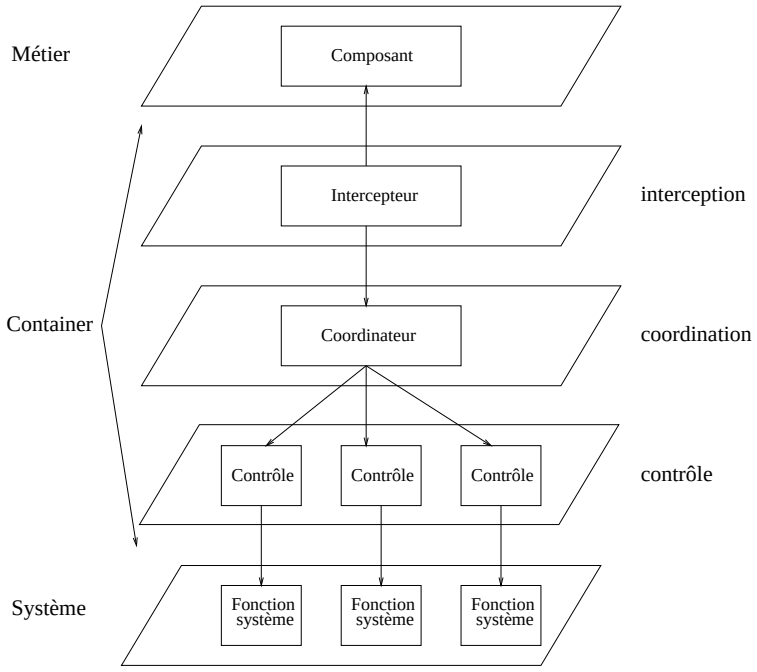

FIG. 2.9 – *Mécanismes clés du conteneur décrits dans [ist].*

En plus de cette vision formalisée et hiérarchique d'un conteneur, P. Merle et M. Vadet proposent des outils de génération de conteneurs. Ces outils permettent non seulement de créer un conteneur dont la configuration est spécifique à une instance de composant

mais aussi d'offrir une même configuration aux conteneurs d'un ensemble d'instances de composants ou d'un ensemble d'instances héritant d'un même type de composants, etc.

JavaPod JavaPod ([BRU 01b], [BRU 01a]), conçu par E. Bruneton, est une plate-forme à composants inspiré du modèle EJB. Cette solution repose sur une extension du langage Java qui intègre une certaine forme de réflexivité. Ainsi les conteneurs ouverts fournis permettent l'ajout d'un nombre illimité de services techniques sous forme de méta-objets.

Fractal Le modèle concret de Fractal, Julia, propose des conteneurs ouverts (ou contrôleur dans la terminologie Fractal) qui permettent de rendre flexibles les composants [BRU 04]. Fractal définit un contrôleur comme un ensemble d'intercepteurs et de sous-contrôleurs que l'on peut ajouter ou retirer selon les besoins (voir figure 2.10). Les intercepteurs, placés sur les interfaces requises ou fournies, permettent l'interception d'appels entrants ou sortants d'un composant. Ils interceptent la signature de la méthode ainsi que ses paramètres d'appel. Puis, ces intercepteurs délèguent le traitement aux sous-contrôleurs qui peuvent ainsi réifier le code : il peuvent exécuter une méthode de pré-traitement, puis la méthode du composant applicatif initialement appelée et enfin une méthode de post-traitement. Les méthodes de pré et post traitement font partie du sous-contrôleur et peuvent utiliser la signature et les paramètres de la méthode interceptée. Un appel de méthode sur un composant est ainsi facilement réifié.
La solution proposée par Fractal n'est que partielle : la relation entre le composant ap-

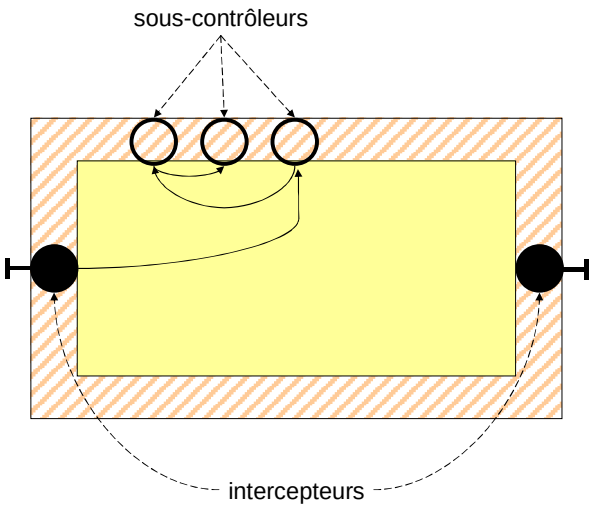

FIG. 2.10 – *Exemple de contrôleur Fractal.*

plicatif et ses services techniques doit se faire à travers le contrôleur, mais il n'y a pas de précision sur la façon de la faire.

SmartTools L'atelier de développement de composants génériques SmartTools ([COU 02], [COU 04b], [sma]) fournit un générateur de conteneurs extensibles. Cet atelier propose

un modèle générique de composants qui peuvent être projetés sur différents modèles de composants tels que les EJB. Le conteneur du composant est généré à partir d'un fichier de description XML.

Machine à Objets Extensibles Les travaux de F. Duclos ([DUC 02a], [DUC 02b]) proposent une machine à objets étendus pour définir des conteneurs extensibles. Ces objets étendus qui remplacent le composant et son conteneur dans les modèle à composants métiers, sont étendus grâce à des "règles" comparables aux points de jonctions de la POA. De plus, il propose un langage de description des services techniques et un langage d'utilisation de ces services qui sont à rapprocher des langages de coupes de la POA.

Jabyce La plate-forme Jabyce [LEN 04], de Romain Lenglet, a pour but de simplifier l'ajout/retrait générique de services techniques en nombre indéterminé. Jabyce ne fournit pas des conteneurs extensibles à proprement parler mais plutôt une solution permettant de gérer la liaison entre code applicatif et code technique. Sa solution est basée sur la transformation de programme au niveau du bytecode Java utilise le modèle à composants Fractal. Elle permet de concevoir de façon modulaire des transformateurs de programme qui sont composés ensemble pour réaliser des transformations complexes. Elle fournit aussi des outils de vérification et de correction du code transformé et de traitement des erreurs. Jabyce a permis entre autre de développer un transformateur de code permettant l'ajout du service de persistance Speedo.
Jabyce est un outil générique qui permet l'ajout statique d'un ou de plusieurs services techniques combinés et peut être utilisé pour la génération des conteneurs. Ces manipulations assez complexes étant laissées à la charge du développeur de conteneur, cette solution ne permet pas une gestion automatique des adaptations.

Conclusion L'approche par conteneur ouvert et/ou extensible est intéressante dans la mesure où elle permet de fournir exactement les services dont l'application a besoin de façon transparente au code applicatif. C'est donc l'un des moyens techniques pour arriver à l'adaptation des services techniques. Cependant, c'est une approche statique où l'on ne gère pas l'environnement d'exécution. Elle n'est donc pas suffisante pour implanter l'adaptation dynamique des services techniques.

Applications adaptatives

Dans les nouvelles applications à base de composants, les composants applicatifs sont déployés sur les différents terminaux de l'application. Le développeur d'application, s'il veut que son code soit adapté à son environnement d'exécution, doit donc adapter son application à chaque terminal. Les projets dont nous pouvons nous inspirer se sont intéressé à cette problématique : CESURE, ARCAD, ACEEL, Qinna. Ils ne sont pas basés sur l'adaptation par modification de leur langage d'implantation ou de l'intergiciel sous-jacent mais sur l'adaptation par (ré)assemblage des composants.

CESURE Le projet CESURE ([PEL 00],[BAL 00]) s'est principalement intéressé à l'aspect déploiement de cette problématique. Dans le modèle qu'il préconise (voir figure2.11), on fournit au serveur de composants la liste des composants applicatifs nécessaires à l'application (liste stockée par exemple sur une carte à puce) ainsi que les caractéristiques du futur environnement d'exécution de ce composant : chaque terminal connaissant son type (ex : PDA, téléphone portable, etc). Ensuite, grâce à un système d'annuaire et de bibliothèques de composants, le serveur applicatif distant fournit en retour les composants applicatifs adéquats.
Cette solution a pour avantage que seul le code nécessaire est embarqué dans la machine et ceci de façon automatique.
Néanmoins, si l'on met à part l'aspect distribué, cette solution permet un déploiement statique automatique d'une version adaptée à la description de son environnement d'exécution d'une application. Mais si l'environnement d'exécution évolue au cours de l'exécution, cette modification n'est pas détectée, l'application n'est donc pas redéployée. De plus, il n'y pas de système permettant de capitaliser les règles d'adaptation du code. Pour chaque application, le développeur doit donc définir chacun des ensembles des assemblages de composants correspondant à un environnement d'exécution précis.

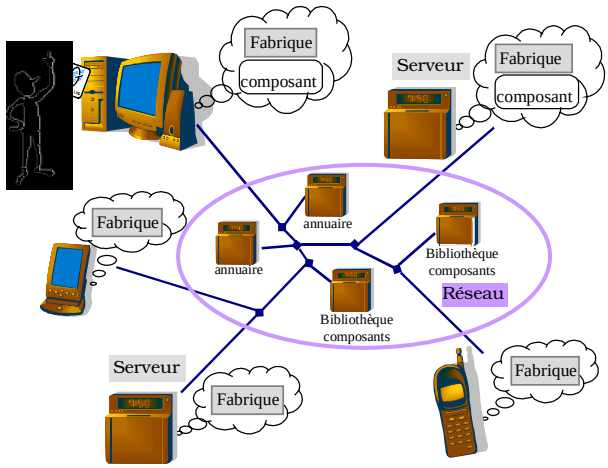

FIG. 2.11 – *Architecture d'une plate-forme CESURE*

ARCAD Dans ARCAD (Architecture Extensible pour Composants Adaptables [DAV 03] [DAV 04b]), Pierre-Charles David nous propose de voir l'adaptation comme une préoccupation : le code gérant l'adaptation y est écrit par le développeur du composant applicatif puis intégré au composant lui même. Le développeur décrit les règles d'adaptation selon un formalisme du type "événement, condition, action". Ces actions pouvant être de deux natures : reconfiguration de l'assemblage de composants et reparamétrisation des composants. Les informations concernant l'environnement d'exécution (correspondant aux conditions) sont obtenues durant l'exécution du système grâce à des outils puis représentées et interprétées. Cette solution a pour avantage d'être très évolutive, dans la mesure où l'on peut ajouter des composants applicatifs ainsi que des règles d'adaptation à n'im-

porte quel moment. De plus, elle offre un cadre assez général pour qu'on puisse adapter n'importe quel type de code. Néanmoins, l'adaptation étant totalement répartie entre les différents composants, il n'existe pas de cohérence dans l'évolution des différents composants. De plus, ce système n'offre pas de solution pour capitaliser les règles d'adaptation et les réutiliser. Ceci peut ne pas poser de problème dans la mesure où l'on s'intéresse au code applicatif : les règles d'adaptation y sont spécifiques. Mais lorsqu'il s'agit de code non-fonctionnel, potentiellement utilisé par tous les composants applicatifs, cela pourrait être moins efficace.

ACEEL Le modèle à composants ACEEL (Cherfour & André [CHE 02]) est destiné aux applications évoluant dans des environnements mobiles. Il a pour but de prendre en compte les variations de ressources (réseau, CPU, énergie).
Ce modèle reprend une définition basique d'un composant : c'est un objet dont les interfaces sont bien définies. Il ne prend pas en compte les notions de connecteur, de liaisons, de configuration.
Il est basé sur le principe de réflexivité (voir figure 2.12). Il se compose d'un niveau de base construit selon le schéma de conception "strategie" [GAM 95] où les différentes variantes d'un algorithme sont des classes implantant toutes la même interface. Le niveau méta gère l'adaptation grâce à des scripts du type condition/action. L'adaptation est déclenchée grâce à un système de détection-notification des changements de l'environnement d'exécution.
Le schéma de conception "strategie" semble adapté aux services techniques, cependant la

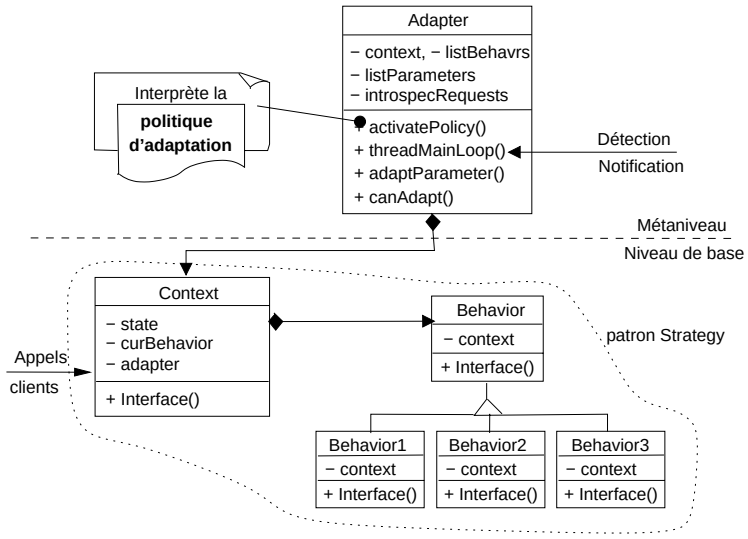

FIG. 2.12 – *Modèle à composants auto-adaptatif ACEEL*

notion de composant ici est vraiment "basique", par exemple il n'existe pas de liaison. De plus, on demande au développeur de décrire les adaptations et il doit connaître a priori toutes les variantes possibles du code.

Qinna Proposé par France Télécom R&D et le CITI de l'INSA Lyon, Qinna [TOU 05] est un modèle à composant, reposant sur le modèle Fractal qui intègre la gestion de la qualité de service. Il est destiné aux applications de type temps-réel. Dans ce modèle, on adjoint à chaque composant nécessitant la gestion de la qualité de service, nommé QoSComponent, des composants qui gère les adaptations ainsi qu'un ensemble de polices d'adaptations. Grâce aux renseignements qui sont connus sur l'environnement, le système crée des contrats entre les composants et négocie la qualité de service fournie par les composants.

Conclusion Dans les solutions présentées, les approches choisies concernant les sources fournissant les paramètres d'adaptation ont des similitudes. Comme expliqué précédemment, ces paramètres sont : l'environnement d'exécution et les besoins de l'application (voir tableau 2.1). Dans les quatre cas, c'est la machine sur laquelle doit s'effectuer l'exécu-

	CESURE	ARCAD	ACEEL	Qinna
informations sur l'environnement d'exécution	fournies par le terminal de façon statique	fournies par les outils ARCAD de façon dynamique	fournies de façon dynamique	idéalement devrait être fourni dynamiquement, mais pour l'instant simulé par un fichier XLM
besoins de l'application	fournis par le développeur de l'application	fournis par le développeur de l'application	fournis par le développeur de l'application	fourni par le développeur sous la forme de polices
type d'adaptation	statique au déploiement	dynamique grâce aux règles du développeur	dynamique grâce aux règles du développeur	dynamique grâce aux règles du développeur

TAB. 2.1 – *Comparaison des adaptations du code applicatif*

tion qui fournit les données d'environnement d'exécution : pour CESURE le type d'environnement d'exécution est fourni par le terminal de façon statique, pour ARCAD, ACEEL et Qinna ces informations sont détectées dynamiquement par un outil proposé par la plateforme. Il semble intéressant de s'inspirer de cette technique, car elle permet au système de rester autonome en fournissant des informations sur son propre fonctionnement. De plus, l'approche statique proposée dans CESURE apparaît plutôt comme une étape vers l'adaptabilité dynamique comme celle proposée dans ARCAD et ACEEL. La solution d'un outil de surveillance dynamique des données est donc à préconiser. Néanmoins, on peut arguer que certaines informations n'étant pas dynamiques, il serait inutile voire coûteux de vérifier leur évolution. Le modèle doit donc représenter deux types d'informations propres : statique ou dynamique.
Concernant les besoins de l'application, dans CESURE ils sont fournis par le développeur sous la forme d'une liste de descriptifs de composants applicatifs (stockée par exemple sur

une carte à puce) à l'utilisateur qui la restitue ensuite lors de l'utilisation. Dans ARCAD, ACEEL et Qinna, elle est exprimée par le développeur de l'application à travers ses règles d'adaptation. Dans les deux cas, on demande au développeur de l'application de fournir ces informations, ce qui lui demande un gros effort supplémentaire. Or, Ceci n'est pas vraiment dans la philosophie de la programmation par composants.

Services techniques adaptatifs

Nous avons vu dans la partie précédente, que dans les solutions actuelles destinées à adapter le code applicatif, on doit utiliser des règles d'adaptation spécifiques à chaque composant et à chaque environnement. Pourtant cela multiplie le développement de chaque composant par le nombre d'environnements possibles.
Une solution pour limiter au maximum la réécriture du code applicatif est d'identifier les parties de code communes aux différentes applications, de développer celui-ci de façon à ce qu'il soit adapté aux différents environnements et ensuite de le réutiliser dans chacune de ces applications.
C'est ce qui est fait par le modèle à composants qui distingue code applicatif et code technique et identifie le code commun entre les différentes applications comme étant le code technique. Cette distinction est faite pour une question de gain de temps de développement car le code technique est fourni avec la plate-forme. De plus, cela permet d'améliorer la sûreté du code technique car celui-ci est développé une seule fois pour toutes les applications par des spécialistes du domaine. Des travaux ont déjà été menés pour rendre les services techniques adaptables. Certains travaux, comme GoTM [ROU 03], Jironde [PRO 03a] ou ArcticBeans [AND 01], s'intéressent à l'adaptation d'un service technique en particulier (le service transactionnel pour ArcticBeans et JOTM, le service de sécurité pour ArcticBeans). Les solutions présentées dans cette partie présentent des services techniques qui gèrent eux même leur adaptation.

ArcticBeans Le modèle à composants ArcticBeans ([AND 01], [Arc]), que nous avons présenté dans la partie 1.2.3 de la partie 1, est basé sur le modèle à composants OOPP (Open-ORB Python Prototype [AND 02]) fondé sur le modèle récursif RM-ODP (ISO Reference Model for Open Distributed Processing [ISO 95a]). Il a pour but l'adaptation des services techniques de sécurité et de transaction, pour cela il repose sur la notion de réflexivité. Chaque service (transaction et persistance) y est conçu sous la forme d'un MOP. Pour l'adaptabilité du service de sécurité, il propose le language Obol qui permet de décrire les besoins d'une application en terme de sécurité sous forme de programme plutôt que de fichiers de déploiements. Ce langage est exécuté par un module nommé Lobo qui est inclus dans le conteneur des composants OOPP afin de déterminer la politique de sécurité. Pour le service de transactions [KAR 03], ces modèles proposent des mécanismes réflexifs qui permettent l'ajout de nouvelles instances de services techniques. Ces mécanismes ont la faculté de vérifier si ces manipulations sur les instances de services sont corrects. Grâce à la vérification de la cohérence des transactions, le système permet de façon sûre à un composant d'avoir en cours plusieurs transactions concurrentes et de types différents.

JOTM Le projet JOTM regroupe la recherche faite autour du service transactionnel dans ObjectWeb. Il contient les sous-projet notamment GoTM et Jironde.
GoTM [ROU 03] de l'équipe GOAL du LIFL à Lille s'intéresse au service transactionnel. Il ne propose pas de service adaptable à proprement parler mais propose plutôt un cadre pour la conception de services transactionnels en fonction des besoins. Il propose de voir le service transactionnel comme un assemblage de services de base (moniteur de transaction, gestionnaire de ressources, gestionnaire de concurrence, etc) permettant de construire plusieurs personnalités. En aval un adaptateur (wrapper), spécifique à chaque implantation du service technique, fait l'interface entre le service transactionnel et sa gestion par le conteneur.
Le projet Jironde ([jir], [PRO 03b]) propose d'implanter les différentes fonctionnalités du service transactionnel sous forme de sous-contrôleur (au sens Fractal). L'adaptabilité vient du fait qu'on peut au moment de la création du contrôleur (ou conteneur) n'instancier que les sous-contrôleurs nécessaires.

Conclusion Des projets ArcticBeans et JOTM présentés ci-dessus, dont le but est l'adaptation d'un service technique particulier, on peut retirer certains principes :

- l'utilisation de la modularité dans GOTM et Jironde : on verra par la suite que c'est l'un des principes que nous préconisons. Dans ces deux solutions, les services techniques eux mêmes sont modularisés.
- malgré le fait qu'il existe différentes implantations d'un service technique, la plupart du temps on peut en dégager les fonctionnalités principales qui se retrouvent dans les interfaces de chacune de ces implantations. Par exemple, pour le service transactionnel, l'interface permet de démarrer, valider ou annuler une transaction, avec des paramètres variant selon les implantations.
- le besoin de décrire les services techniques, leur niveau de qualité de service et leurs règles d'adaptation qu'on retrouve dans ArcticBeans.

Cependant, dans les différents projets présentés, une solution spécifique à chaque service est choisie afin de fournir une solution plus efficace de la gestion de l'adaptation. Aucune approche globale n'est proposée, il n'y a donc pas de cohérence dans l'adaptation des services techniques et la solution proposée n'est pas réutilisable pour un autre service. Ces solutions ne sont donc pas applicables directement à n'importe quel service technique ce qui limite leur réutilisation.

2.2 Vers une solution générale pour l'adaptation des services techniques

Afin de nous positionner, nous devons répondre à deux questions : "à quel niveau allons nous effectuer l'adaptation?" et "grâce à quelles techniques?".

Nous avons vu que l'adaptation peut intervenir au niveau de l'ORB, du conteneur ouvert, de l'application ou des services techniques eux mêmes. D'un côté, étant donné que

les ORB intègrent les services techniques et les conteneurs, l'adaptation des services techniques peut être vue comme une adaptation d'un sous ensemble de l'ORB. Cependant, les solutions existantes pour l'adaptation au niveau des ORB ne sont pas suffisantes car elle ne prennent pas en compte la possibilité d'ajout de services techniques.
D'un autre côté, l'adaptation des applications peut être vue comme une solution complémentaire à l'adaptation des services techniques, dans la mesure où elle ne porte pas sur le même code. L'adaptation, que nous préconisons, se fait à deux niveaux : le service technique lui-même et le point de jonction entre les services techniques et le conteneur.

Concernant les techniques d'adaptation, les solutions pour l'adaptation des applications ont pour inconvénient de ne pas tenir compte des spécificités des services techniques et de laisser beaucoup de travail au développeur. En nous intéressant plus particulièrement aux services techniques, nous pouvons proposer une solution plus efficace.
A l'inverse les solutions proposées pour l'adaptation des services techniques sont spécifiques à un service technique particulier. Nous préconisons donc une solution spécifique aux services techniques et qui puisse être appliquée à tous les services techniques.
Les techniques choisies sont donc : la réflexivité, la programmation par composants, les conteneurs ouverts et une adaptation du patron "stratégie" à la programmation orientée composants (voir figure 2.8). Nous allons expliquer pourquoi nous choisissons ces techniques à la base de notre solution :

- Aussi bien dans le domaine des ORB (dynamicTAO, UIC, Flexinet) que dans ceux des conteneurs ouverts (IST-coach, Javapod) et des applications adaptables (ACEEL), on a pu vérifier que la **réflexivité** était un concept puissant pour concevoir des système adaptatifs dynamiquement. Elle permet notamment de dissocier le code de l'adaptation du code du système. Nous gérons les adaptations de la liaison, jusqu'ici implicite, entre le conteneur et les services techniques à l'aide d'un système méta qui apportera la dynamicité nécessaire.
- Comme dans les travaux sur l'adaptabilité des applications (CESURE, ACEEL) ou des services techniques (ArcticBeans, Gironde), nous voulons adapter le code lui même à travers notamment l'utilisation de la **programmation par composants**. La programmation modulaire préconisée par le modèle à composants permet d'associer un élément de code (e.g. un composant) à une stratégie d'adaptation. De plus, elle permet la réutilisation de code entre les différentes versions d'un même service.
- Au niveau du point de jonction entre conteneur et services techniques, nous nous appuyons sur les conteneurs ouverts qui permettent l'ajout de services techniques. La notion de **conteneurs ouverts** apporte flexibilité et précision à la conception des conteneurs. Elle nous permet d'ajouter au composant les services techniques nécessaires. Cependant, elle n'apporte pas de solution à la gestion dynamique telle que nous allons la présenter.
- Enfin, si l'on considère différentes versions d'un même service comme étant autant de "stratégies", on peut utiliser une version "composant" du patron "stratégie", où les différentes stratégies sont implantées par des composants ayant la même interface.

2.2.1 Contraintes

Cette partie pose les contraintes fixées à nos travaux. Elles émanent principalement du fait que, travaillant dans le domaine de la programmation par composants, il est impératif d'en respecter la philosophie. Ce modèle est basé sur trois grands principes qui sont : l'interopérabilité, la réutilisabilité et la simplicité d'implantation.

- *interopérabilité* : les composants applicatifs doivent évidemment rester interopérables au sein d'un ORB mais aussi entre les ORBs. Il faut donc garantir l'interopérabilité des services techniques notamment si l'on a deux composants applicatifs utilisant deux versions ou deux implantations différentes d'un service technique.
- *réutilisabilité* : à la fois des composants applicatifs et des services techniques. La réutilisabilité des composants applicatifs signifie notamment que l'on ne doit pas affaiblir la séparation entre code fonctionnel et non-fonctionnel. La réutilisabilité des services techniques est pour l'instant très limitée, dans la mesure où ces services apparaissent sous la forme d'objet notoire; on peut donc espérer l'améliorer en rendant les services techniques plus modulaires.
- *Simplicité d'implantation*
  - Pour le *développeur de composants applicatifs* : dans ce type de modèle, on permet au développeur de délaisser les aspects fonctionnels pour se consacrer essentiellement à la logique applicative. On ne peut donc pas surcharger le travail du développeur de composants en lui demandant de définir une politique lourde d'utilisation des services techniques.
  - Pour le *développeur de code technique* : pour l'instant les développeurs de services techniques écrivent des objets et ne doivent pas décrire les services techniques ou leurs possibles adaptations. Il nous faut donc définir des solutions en terme de description et de développement facile à utiliser.
  - Pour l'*administrateur* : tout comme le développeur de services techniques, l'administrateur n'a pas besoin de gérer les services techniques. Il faut concevoir une solution qui requiert le moins possible son intervention et qui lui permet de capitaliser ses connaissances quant au comportement du système.

2.2.2 Améliorations visées

Dans un premier temps, l'amélioration visée est que les services techniques puissent fonctionner dans les différents environnements dans lesquels ils sont susceptibles de s'exécuter. En effet, les solutions existantes ne permettent pas de choisir la version de service technique adaptée à son environnement, il est possible donc que ces services techniques ne puissent pas du tout s'exécuter. Si les ressources nécessaires à son fonctionnement ne sont pas disponibles, le service risque d'entraîner une défaillance du système. Par exemple, un service de persistance ne prennant pas en compte la place mémoire lui restant allouée, pourra entraîner des disfonctionnements si les données qu'il doit stocker sont plus grandes que la place en mémoire libre restante. Il est donc nécessaire de fournir au système un service qui, certes pourrait fournir une qualité de service moindre, mais n'empêcherait pas la plate-forme de fonctionner et d'effectuer le service en mode dégradé.

Dans un second temps, on voudrait améliorer la qualité de service fournie. En effet, lors d'une augmentation des ressources, le système doit pouvoir bénéficier de services techniques de plus grande qualité, c'est à dire, par exemple, des services ayant un temps de réponse plus petit, un niveau de sûreté plus grand, une sécurité accrue, ou encore un nombre d'informations stockées par le système plus important. Ainsi on pourrait arriver à une amélioration globale de la qualité de service. Il est impossible d'énumérer ici tous les facteurs de qualité de service potentiellement améliorables dans la mesure où ceux-ci dépendent du service technique. De plus, mesurer la qualité de service d'un système est un problème assez complexe. En effet, cette qualité de service globale est une fonction de l'ensemble des facteurs, qui sont non seulement difficilement énumérables mais aussi parfois difficilement mesurables. Si le temps de réponse est une donnée facilement quantifiable, d'autres le sont plus difficilement, comme par exemple le niveau de sécurité.
En conclusion, le but fixé est donc de permettre au minimum à la plate-forme de s'exécuter sur les machines hétérogènes sur lesquelles les nouvelles applications distribuées s'exécutent. Nous montrerons qu'ils permettent aussi d'améliorer la qualité de service, cependant nous ne nous pencherons pas sur la question complexe de la mesure de la qualité de service.

2.2.3 Synthèse

Dans cette partie dédiée à la problématique, nous avons montré comment l'émergence de nouveaux types d'appareils portables et dotés de connexions sans fil au réseau s'est accompagnée de la création de nouveaux types d'applications. Or, ces applications temps-réel, multimédia, hautement distribuées ou mobiles ont un temps de mise sur le marché très court. Ce qui amène les développeur à choisir des solutions de développement facilitant la modularité et réutilisabilité du code développé. L'une de ces solutions est le modèle à composants. L'utilisation de ce modèle est un moyen facile de gérer la distribution et d'autres services tels que les services techniques. Cependant, les modèles industriels ne proposent pas de solution pour adapter le code aux particularités des environnements hétérogènes dans lesquels le code s'exécute. De ce fait, certaines applications ne sont pas réutilisables dans ces environnements. Une première solution utilisée a été de développer plusieurs solutions de l'application, en fonction des futurs environnements d'exécution. Une amélioration de cette solution a consisté à rendre automatique le passage d'une version à une autre de l'application. Néanmoins cette solution a pour inconvénient qu'elle nécessite que le développeur de l'application écrive lui même les différentes versions du code et décrive leurs adaptations.
Nous promouvons une solution qui peut être complémentaire à celle-ci : adapter non pas le code applicatif mais le code technique. Ainsi, le travail du développeur n'est pas surchargé et le travail d'adaptation est confié aux développeurs de services techniques qui sont des experts. Comme nous l'avons vu dans l'état de l'art, cette approche a été adoptée dans différents travaux de recherches. Néanmoins, les solutions proposées ne nous satisfont pas pleinement. En effet, certaines travaux proposent une adaptation spécifique à un service technique, ainsi le niveau d'adaptation est assez poussé et les services techniques obtenus offrent des mécanismes d'adaptation assez puissants. Cependant leur démarche est difficilement applicable à un autre service technique, les mécanismes obtenus ne sont

pas réutilisables et l'on ne garantit pas une cohérence de fonctionnement de la plate-forme si l'on ajoutait plusieurs de ces services. D'autres travaux permettent une adaptation des services techniques par la possibilité d'ajout ou de retrait des services, ils ne proposent pas de solution pour gérer automatiquement une plus grande adaptation.
La solution que nous décrivons dans le chapitre suivant a donc pour but de permettre à une plate-forme à composant de fournir et gérer des services techniques adaptables à leur environnement d'exécution et aux besoins de l'application. Cette solution prendra en compte les contraintes liées au domaine de la programmation par composant : inter-opérabilité, réutilisabilité, simplicité d'implantation. Elle ne doit pas surcharger de façon trop significative le système.

Il est nécessaire de mener une une réflexion concernant deux aspects : la conception des services techniques d'une part et la gestion de l'adaptation des services au cours des changements d'environnement d'autre part.

- Concernant le premier point, nous allons redéfinir les étapes de conception, développement et d'assemblage avec les composants applicatifs, afin de guider le développeur de services techniques dans la conception de services techniques adaptables, capables de "présenter" les différentes personnalités qu'ils peuvent implanter (voir partie II chapitre 1).
- Un service de gestion de l'adaptation doit conduire l'adaptation dynamique des services techniques décrits précédemment. Cette adaptation est régie par un diagramme d'état appelé boucle d'adaptation (voir partie II chapitre 2 section 2.1). Afin d'incarner cette boucle, le service d'adaptation fournit un mécanisme de détection de changement de l'environnement d'exécution, ainsi un service d'annuaire dont le but est de recenser les personnalités et choisir celle qui sera la plus adaptée à l'environnement présent et aux besoins exprimés par le développeur de l'application (voir partie II chapitre 2 section 2.2).

La validation de ces propositions est faite à travers la mise en œuvre d'un service transactionnel mutli-personnalités ainsi qu'un prototype du service de gestion de l'adaptation (voir partie II chapitre 3).

Deuxième partie

Propositions et validations

1

Un nouveau modèle pour la réalisation des services techniques

Nous avons montré le besoin d'adaptabilité des services techniques à leur environnement d'exécution et aux besoins des applications afin de fournir une meilleure qualité de service. D'après ce que nous avons vu concernant les services techniques et intergiciels à composants qui les fournissent, le processus de conception actuel des services techniques ne facilite pas leur adaptation. La conception orientée objet utilisée pour réaliser les service techniques ne permet pas de dissocier clairement les préoccupations et d'exprimer les relations qui existent entre ces préoccupations. Dans les intergiciels, le code associant services techniques et composants applicatifs est figé.
Afin de mettre en évidence les différents niveaux d'indépendance par rapport au modèle et donc de réutilisabilité de notre solution, nous avons suivi l'approche préconisée dans l'approche MDA (Model Driven Architecture [DAV 04a]). Pour cela, nous avons dégagé les différents niveaux d'abstraction de notre proposition (voir figure 1.1). Dans la phase de conception, nous cherchons à donner une représentation de nos services techniques indépendante d'un modèle et qui prend en compte leurs spécificités et la possibilité de leur adaptation. Pour cela, nous proposons la notion de personnalité de service technique, elle permet notamment de représenter la coexistence de plusieurs versions ou modèles d'un même service technique. Elle donne aussi une description de la qualité de service rendue par le service technique qui sera utilisée lors de l'adaptation.
Afin de pouvoir projeter les concepts définis dans cette première phase de conception, nous avons choisi un modèle de développement : le modèle à composants. Ce modèle nous fournit des concepts génériques pour modéliser notre système, notamment la conception modulaire. Cette conception des services techniques permet d'accroître la réutilisabilité de code entre différentes versions d'un même service technique et son adaptabilité en facilitant l'expression d'un service à travers un assemblage de modules logiciels.
La seconde phase de conception, affinage de la première, repose sur le modèle à composants Fractal qui permet principalement la conception récursive des composants. Ainsi, une composition de plusieurs modèles d'un même service peut être vue comme un seul composant, la gestion des services techniques en est simplifiée.
Basée sur Julia, implantation standard de Fractal, la phase d'assemblage des composants applicatifs et techniques est redéfinie tout en maintenant la séparation des codes

techniques et applicatifs et en facilitant l'ajout, le retrait ou le changement de services techniques. Notre proposition s'appuie sur la modularité du contrôleur (ou membrane) des composants de Julia. Nous redéfinissons un sous-contrôleur (module de base du contrôleur) comme étant un composant. Les services techniques sont intégrés aux sous-contrôleurs, ceci facilitant leur utilisation. Les services techniques gardent leur autonomie grâce à la possibilité offerte par Fractal à différents composants composites de partager un même composant. Ainsi ce service technique peut appartenir au contrôleur de plusieurs composants applicatifs. Dans les sections suivantes, nous revenons sur chacune de ces phases que

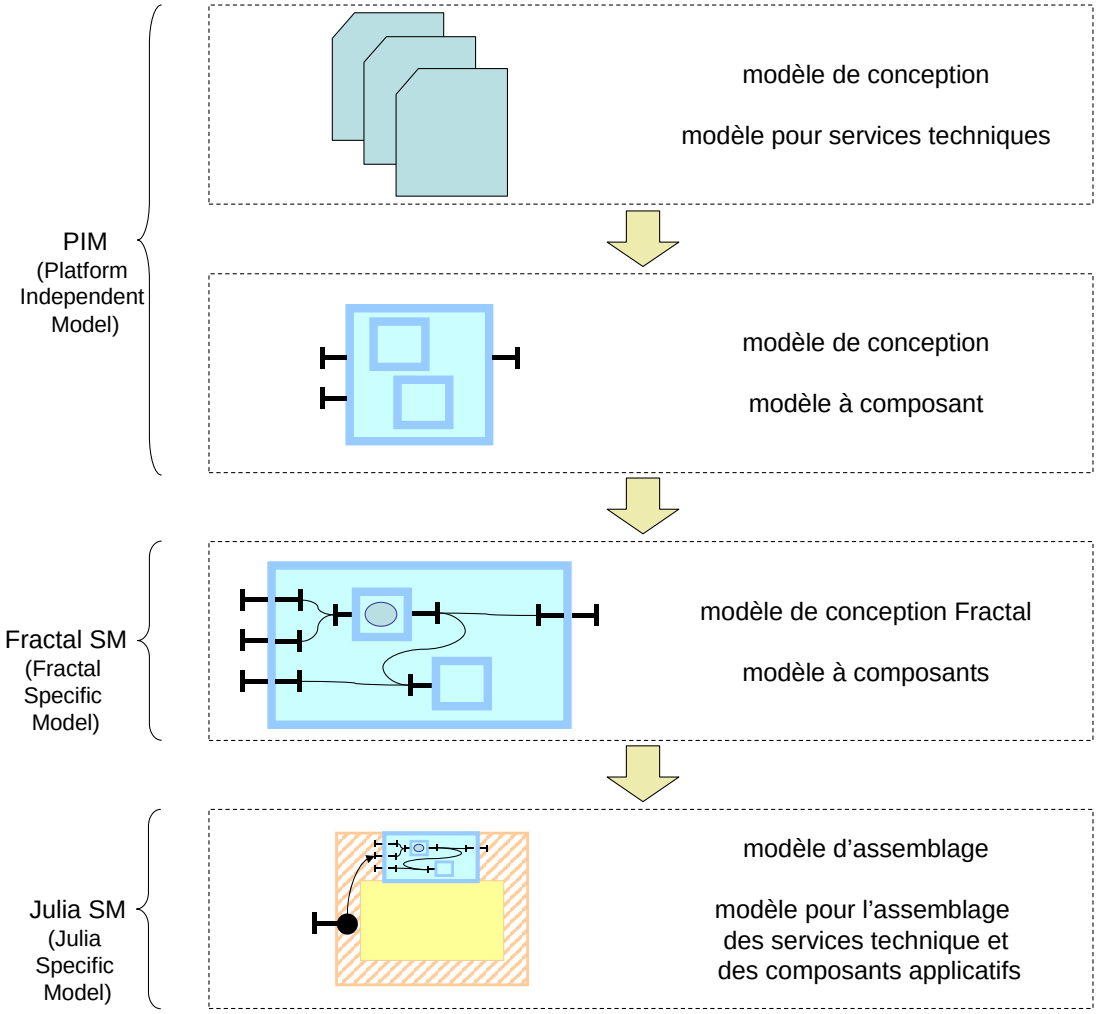

FIG. 1.1 – *Projection des différents modèles pour les services techniques.*

nous détaillons.

1.1 Modèle de conception : notion de personnalité

Cette phase de conception correspond à la première couche de la figure 1.1. Elle permet de déterminer les concepts architecturaux nécessaires à l'adaptation des services tech-

niques. L'objectif ici est de permettre qu'un service apparaisse sous la forme de plusieurs personnalités correspondant à des standards ou des implantations différentes. Cette représentation passe par une description des caractéristiques du service technique qui doit permettre à l'administrateur (pour l'adaptabilité statique) ou au système (pour l'adaptabilité dynamique) de choisir l'implantation adéquate du service technique.

1.1.1 Un service à plusieurs personnalités

Nous appelons l'entité représentant un modèle ou une version d'un service une P1S ou "Personnalité d'1 Service" [HéR 02]. Ici le terme service désigne le service en général sans distinction du modèle ou d'implantation, par exemple le service de transaction, de sécurité, de persistance. Le mot personnalité permet de signifier le fait qu'un service peut être rendu de plusieurs façons différentes, chacune transposée sous la forme d'une P1S distincte. Par exemple, un service d'annuaire peut être rendu de plusieurs façons : par nommage (le paramètre passé est le nom de l'objet recherché), par courtage (le paramètre passé est une description de l'objet), par un annuaire LDAP (on peut alors avoir des critères de recherche plus complexes). Chacun de ces services peut être représenté par une P1S, on peut même avoir plusieurs P1S pour un modèle s'il a plusieurs implantations différentes ou plusieurs versions.
Le concept de P1S permet alors de donner une abstraction pour représenter les implantations des services techniques. Cette représentation est indépendante de l'implantation du service.

1.1.2 Représentation de la qualité de service

Comme nous l'avons vu dans la partie 2.2.4 de la problématique, cette P1S doit nous permettre de faire un choix aux différentes étapes de l'adaptation (pour l'embarquement du code, son instanciation et son utilisation). Ce choix est basé sur des considérations telles que la taille du code, la mémoire vive nécessaire à son instanciation, sa rapidité d'exécution. Ces caractéristiques font partie de la qualité de service fournie par le code technique.
La définition donnée par l'ISO de la qualité de service est : *"un ensemble de qualités relatives au comportement collectif d'un ou plusieurs objets"* ([ISO 95b], [AAG 01]).
La norme ISO/IEC 9126 ([ISO 99a], [ISO 99b], [ISO 99c] et [ISO 00]) relative à la spécification et l'évaluation de la qualité de service des logiciels identifie deux aspects de la qualité de service : la qualité d'utilisation et les qualités interne et externe. D'un côté, la qualité d'utilisation permet de mesurer le degré de satisfaction de l'utilisateur c'est-à-dire jusqu'à quel point l'utilisateur a réussi à remplir ses objectifs. D'un autre côté, la qualité externe s'interesse aux propriétés du logiciel en lui-même et la qualité interne aux propriétés des différentes parties de ce logiciel.
La qualité de service que l'opérateur humain perçoit n'entre pas en jeu ici car le fonctionnement interne de la plate-forme est cachée à l'utilisateur. C'est donc la qualité interne de la plate-forme qui nous intéresse. La norme ISO définit six facteurs qui permettent de mesurer la qualité de service. Ces facteurs sont les mêmes pour la qualité de service interne et la qualité de service externe, seul le point de vue change :

– niveau des fonctionnalités : capacité du logiciel à fournir les fonctions qui répondent

aux besoins définis ou implicites. Ce facteur regroupe lui même les facteurs d'adéquation, d'exactitude, d'interopérabilité et de sécurité.
- fiabilité : capacité d'un logiciel à maintenir le niveau de performance spécifié. Ce facteur regroupe les facteurs de maturité, de tolérance aux pannes et de possibilité de récupération.
- convivialité : capacité d'un logiciel a être compris, appris, utilisé et attractif pour ces utilisateurs. Il regroupe les facteurs de facilité de compréhension, de facilité d'apprentissage, d'efficacité opérationnelle, attractivité.
- efficacité : capacité d'un logiciel à fournir des performances appropriées à la quantité de ressources utilisées. Il regroupe les facteurs de rapidité d'exécution et d'utilisation des ressources.
- facilité de maintenance : capacité d'un logiciel à être modifié à travers des corrections, améliorations, ou adaptation du logiciel aux changement d'environnement, de spécification. Il regroupe les facteurs de facilité à être analysé ou modifié, de stabilité, de testabilité.
- portabilité : capacité d'un logiciel à être transféré d'un environnement à un autre. Ce facteur regroupe les facteurs d'adaptabilité, facilité d'installation, co-habitation, facilité de remplacement.

La qualité de service fournie par une personnalité de services dépend de l'adéquation entre la qualité de service qu'elle requiert et la qualité de service fournie par son environnement. Nous devons donc représenter et quantifier trois qualités de service : celle fournie par le service technique, celle requise par le service technique mais aussi celle fournie par l'environnement. On choisira la personnalité en fonction du service qu'elle fournit ainsi que de l'adéquation existante entre la qualité de service requise par la personnalité et la qualité de service fournie par l'environnement d'exécution.

1.1.3 Contenu de la P1S

La P1S doit permettre de décrire la qualité de services d'une personnalité particulière d'un service technique ainsi que celle de l'environnement d'exécution auquel cette personnalité est adaptée (c'est la qualité de service requise). La P1S prend donc la forme d'une association entre une implantation du service technique et la description de ces qualités de service (voir figure 1.2).
Dans la description de la personnalité, certains facteurs sont indispensables au choix de la personnalité, d'autres optionnels, d'autres encore ne sont pas exprimés explicitement.

Facteur indispensable

Le niveau de fonctionnalité de la personnalité est le premier facteur de choix. Il fournit la fonction du service rendu (transaction, persistance, etc). En effet, la fonction du service rendu sert à choisir le service technique en fonction des besoins du composant applicatif. Au niveau du composant applicatif, c'est le renseignement minimal connu pour décrire les besoins du composant applicatif en terme de services techniques. Si l'on se réfère à la notion d'objet notoire qui était jusqu'alors utilisée pour retrouver les services techniques,

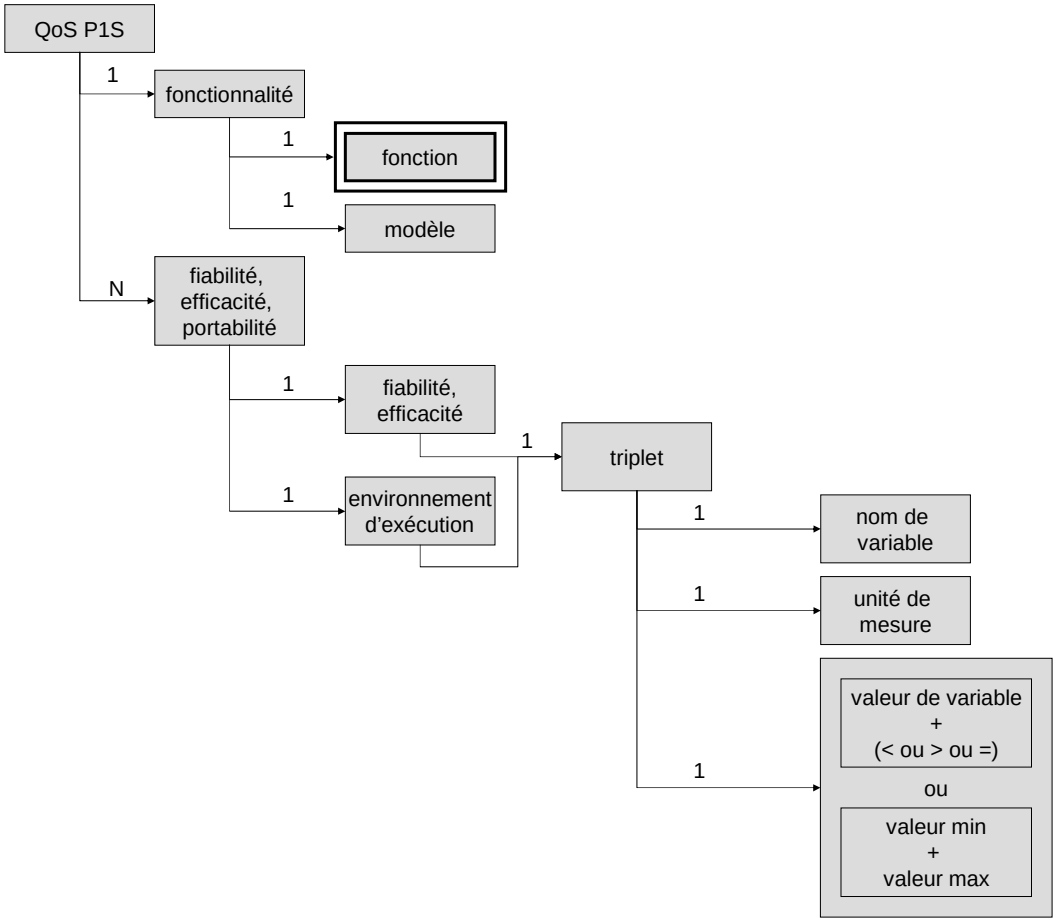

FIG. 1.2 – *Qualité de Service d'une P1S.*

les développeurs de composants applicatifs devaient uniquement connaître le type du service. La correspondance entre le nom du service technique et son emplacement était faite par le système.

Facteurs optionnels

D'autres facteurs permettent d'affiner ce choix :

- Le niveau de fonctionnalité est complété par le modèle de la personnalité car il permet au développeur de l'application d'affiner son choix de personnalité, sachant que certains modèles sont plus adaptés à certains environnements ou à certains types d'applications.
- Les facteurs de fiabilité, efficacité et portabilité permettent d'affiner le choix. Les facteurs de fiabilité et d'efficacité concernent le service fourni par le service technique (ex : la vitesse d'exécution du service, la fréquence des pannes, la version de l'implantation, etc). Le facteur de portabilité correspond à l'environnement d'exécution auquel la personnalité est adaptée (ex : la vitesse du CPU, la taille de la mémoire vive, la connectivité au réseau, etc). Nous avons décrit cet environnement

dans la partie 2 chapitre 3.

Pour être complet, la description des facteurs de fiabilité, efficacité et portabilité donne la possibilité de décrire non pas un environnement d'exécution et les facteurs de fiabilité et d'efficacité correspondantes mais un ensemble de couples de ce type (la cardinalité de l'association est notée N sur la figure 1.2). Chacun de ces couples regroupe :

- la description de la fiabilité et l'efficacité fournie par le service technique : si l'on a plusieurs services techniques pouvant correspondre aux contraintes du système réel, alors on pourra faire le choix du service technique offrant la meilleure qualité de service.
- la description de l'environnement d'exécution pour lequel la P1S est adaptée : en fonction de l'environnement d'exécution réel, on pourra chercher le service technique qui pourrait au mieux fonctionner.

Les descriptions de l'environnement d'exécution et celle de la qualité de service fournie prennent la forme d'une liste de triplete contenant

- le nom de la variable,
- l'unité de mesure
- le triplet (valeur de la variable, opérateur, $>$ ou $<$ ou $=$) ou le couple (la valeur minimum , valeur maximale de la variable).

Facteurs non explicités

Les facteurs de convivialité et de facilité de maintenance ne sont pas prépondérants. Concernant le facteur de convivialité, l'utilisation des services techniques est automatisé, les aspects d'apprentissage, de compréhension du service technique doivent être faits par l'intégrateur des services techniques avant l'exécution du système et non par le système pendant l'adaptation. Ce facteur n'entre donc pas en jeu dans le choix d'une personnalité. C'est la même chose pour le facteur de facilité de maintenance, qui n'est pas effectuée par l'utilisateur ni par le système lui même. Notons que le language CQML proposé dans la thèse de Jan Øyvind Aagedal [AAG 01], dédié à la description de la qualité de service des logiciels est très complet. Cependant, il ne semble pas exister de logiciel pour l'analyser et l'intégrer à notre plate-forme. N'ayant pas besoin de tous les concepts développés, nous avons préféré une version simplifiée de la description de la qualité de service telle que nous l'avons décrite plus avant.

Notons par ailleurs qu'il est nécessaire que les développeurs d'applications, de services techniques et du système gérant l'adaptation se mettent d'accord sur une ontologie commune. Cette ontologie regroupe un vocabulaire commun pour les types des services techniques, des modèles, les unités de mesures, etc. Cette ontologie doit être construite grâce à l'apport des différents spécialistes du domaine.

1.1.4 Génération de la P1S

On identifie deux sources de renseignements pour remplir cette description : les connaissances théoriques du développeur du composant et les outils empiriques de mesure des

performances du service technique. Le développeur de service technique fournit les informations qui ne sont pas quantitatives (nom, modèle, version). Il peut aussi fournir des informations quantitatives en se basant sur une connaissance algorithmique du service qui peuvent être démontrées ou non. Il peut par exemple connaître la compléxité temporelle ou spatiale (la taille des données utilisées) du service.
Les méthodes empiriques consistent à tester le service technique dans les différents environnements d'exécution sur des bancs de tests (benchmark en anglais) spécifiés par des organismes indépendants. Il existe plusieurs difficultés à ces tests : représentativité de l'environnement de test, complexité des données à collecter.
Ces tests sont effectués par le développeur de services techniques a priori c'est à dire dans l'environnement d'exécution décrit dans la section "environnement d'exécution" de la P1S. On pourrait aussi envisager des tests in situ, cependant il faudrait fournir au système les tests de performance adéquats et les exécuter lors de chaque changement d'environnement. Mais, de cette façon on surchargerait artificiellement le travail de la plate-forme.
D'autre part, dans la mesure où les valeurs de ces facteurs font partie d'ensembles continus, il est pratiquement impossible de tester toutes ces valeurs. De plus, il est très difficile d'empêcher totalement ces valeurs de varier pendant le test. On procède donc à une discrétisation des ensembles. Selon le comportement du service et la finesse de l'adaptation à laquelle on veut arriver, on peut simplifier ces fonctions en ne gardant que quelques couples : intervalle sur l'environnement d'exécution, valeur de la qualité de service. Par exemple, nous avons testé un service de transaction qui est capable d'exécuter 311500 transactions par minutes lorsque le CPU est initialement utilisé entre 0% et 15%, 258000 entre 15% et 30%, etc. Ou encore, si un service technique a de bonnes performances au dessus d'un certain seuil, puis que ses performances s'écroulent, on pourra ne garder comme information que le fait que le service s'exécute convenablement au dessus de ce seuil.

1.1.5 Lot de Services

La notion de P1S nous permet de représenter les différentes implantations d'un service technique qui cohabitent dans notre système. De plus, elle nous permet de les décrire afin d'effectuer un choix en fonction de l'environnement et des besoins de l'application.
Cependant, la description faite d'un service technique grâce à la notion de P1S suppose que l'on peut voir un service technique comme une entité autonome. Or, lorsqu'on développe des services techniques, on se rend compte que certains sont étroitement liés et qu'il est parfois difficile de totalement les dissocier. Par exemple, un annuaire LDAP fait appel au service de sécurité afin de chiffrer les échanges entre le client du service et lui-même ou pour authentifier le client.
De plus, considérer chaque P1S comme entité à part entière nous expose à devoir choisir l'ordre d'exécution de ces P1S. Par exemple, on doit déterminer l'ordre d'exécution entre un service de sécurité qui chiffre les données et un service de compression des données.
Enfin, de par leur expérience, les concepteurs de services techniques peuvent préconiser des associations de P1S qui fonctionnent bien ensemble. Par exemple, on sait que deux services, un service de transactions avancées et un service de reprise sur panne prenant en

compte les déconnexions fréquentes sont tous les deux adaptés à un environnement ayant de nombreuses ruptures de communication.

Nous définissons donc un second niveau de granularité pour les personnalités : un lot de

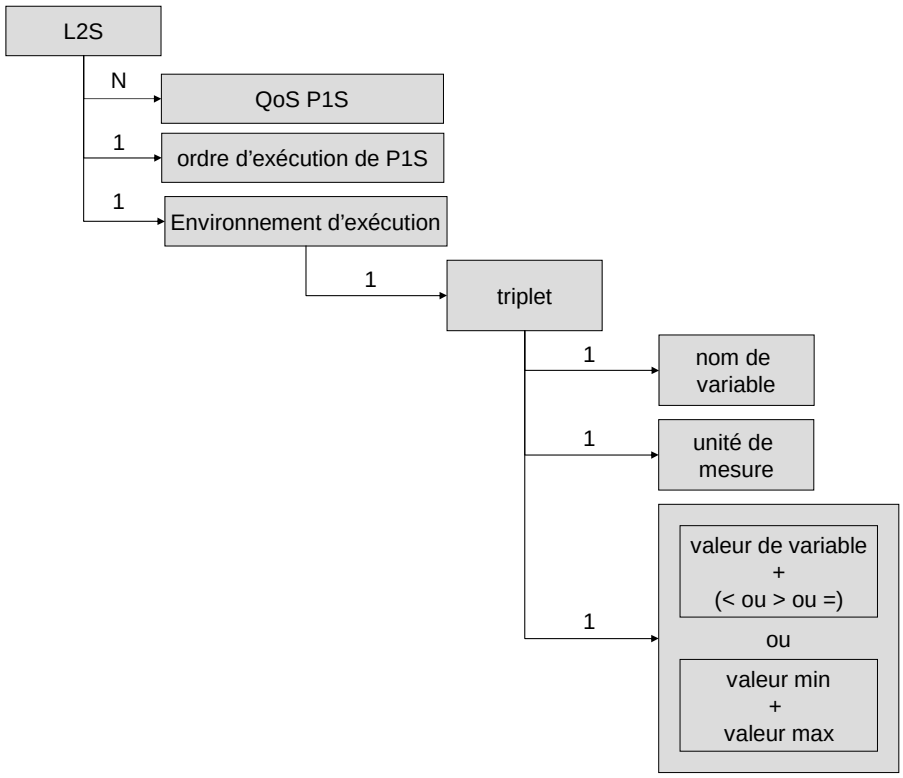

FIG. 1.3 – *Description d'une L2S.*

services ou L2S, le premier étant la P1S ([HéR 03b], [HéR 03a]) . Cette entité représente un ensemble cohérent de services techniques. Elle est donc constituée d'un ensemble de P1S ainsi que d'une description du contexte applicatif et de l'environnement d'exécution auquel ce L2S est le mieux adapté comme celle de la P1S(voir figure 1.3). Représenter un ensemble de P1S comme une P1S simplifie la gestion des services techniques en la résumant, du côté composant, au choix d'un L2S adapté à l'environnement d'exécution. Le travail de description de la L2S est fait par le développeur de services techniques car, soit il a la connaissance empirique ou théorique du bon fonctionnement de P1S, soit il a conçu ces services techniques ensemble.

1.2 Modèle de conception et de développement : choix du modèle à composants

Dans cette section, nous allons décrire les couches 2 et 3 de la figure 1.1. La couche 2 correspond à une phase de raffinement de la conception et la phase 3 au développe-

ment. L'approche choisie pour ces couches est celle de la programmation par composants [HéR 04b]. Ce modèle initialement destiné à la conception et au développement d'applications peut en effet s'appliquer aux services techniques de façon bénéfique. Cette approche a d'ailleurs tendance a se généraliser, elle est rendue possible grâce aux nouveaux modèles à composants hiérarchiques. C'est l'un de ces nouveaux modèles, le modèle Fractal, que nous avons choisi pour mener nos travaux. En effet, il nous fournit non seulement le modèle abstrait nécessaire mais aussi un modèle concret sur lequel nous avons basé nos expérimentations.

1.2.1 Des services techniques développés à base de composants

Jusqu'à présent, dans les modèles industriels, les services techniques n'ont pas été conçus sous forme de composants mais sous forme d'objets [HéR 03a]. On peut identifier plusieurs raisons à cela : tout d'abord, la programmation orientée objet était suffisante pour développer les services techniques. En effet, ils étaient principalement destinés à fonctionner sur des serveurs conventionnels, l'adaptation de ces services n'était donc pas une priorité. De plus, lors du développement des plates-formes à composants, on a préconisé la réutilisation du code technique pour les bus logiciels déjà existants sous forme d'objets, tel que les services techniques de CORBA pour CCM [CCM99]. En effet, leur utilisation était bien connu des développeurs d'ORB et avait fait la preuve de leur efficacité en terme de sûreté et de rapidité d'exécution, et ont été le résultats de nombreuses discussions au sein de l'OMG. Cela permettait aussi un gain de temps dans le développement de la plate-forme.
Enfin, l'utilisation des composants pose des problèmes en terme de performance. En effet, le temps d'exécution d'une application à base de composants est parfois plus long que celle d'une application à base d'objets. Cela est notamment du à la gestion des services par les conteneurs. Cependant concernant les performances, les réticences concernant le code technique sont les mêmes que pour le code applicatif pour lequel il a déjà été démontré que le modèle à composants possède des avantages, en terme de réutilisabilité de code, qui contrebalancent cet inconvénient.
Cependant, la programmation orientée objet n'est pas très bien adaptée à la conception de services techniques facilement adaptables. En effet, elle ne fournit par les outils efficaces pour la conception des services techniques : elle ne représente pas explicitement la notion de tâche (mise à part la notion d'interface) et d'assemblage de tâches. Si l'on veut reconfigurer les interactions entre les objets, on doit recompiler le code ou faire appel à des mécanismes complexes d'interception. Lors de l'exécution, elle ne fournit pas non plus les outils pour retrouver facilement les services et les réassembler avec les applications techniques.

Nous proposons donc l'utilisation de la programmation par composants pour les services techniques. Les services techniques sous forme de composants alors développés possèdent les mêmes caractéristiques bénéfiques qu'un composant applicatif.

- La modularité des composants permet de modéliser la notion de tâche. Elle facilite la conception et accroît la réutilisabilité du code. Ceci permet d'une part un gain de temps et de l'autre une meilleure cohérence du service grâce au partage de mé-

canismes ou de données. Ainsi, on peut définir un service technique comme étant un assemblage de plusieurs composants effectuant chacun une fonction élémentaire. Or, deux P1S d'un même service ont souvent des fonctions élémentaires en commun. Ils peuvent partager une structure de données ou un mécanisme de base. Ces composants sont donc potentiellement réutilisables entre plusieurs personnalités d'un même service.

- De plus, la programmation par composant promeut l'expression des différents points de vue d'une application à travers ses différentes interfaces. On applique ce principe aux services techniques : les méthodes fournies pour rendre le service de différentes façons sont regroupées dans différentes interfaces, ces interfaces étant associés aux différentes P1S fournies par le service. Ainsi, on facilite la phase de conception : pour concevoir des services techniques facilement adaptables, le développeur de services techniques se contente de développer les composants élémentaires et décrit les différents assemblages qui peuvent en résulter ainsi que les interfaces fournies associées.
- Grâce aux liaisons, il est possible de définir des assemblages de composants qui permettent de (re)configurer rapidement les interactions entre les composants. On peut ainsi décrire simplement les adaptations des services techniques sous la forme d'une reconfiguration de leurs composants élémentaires.
- Il existe des mécanismes plus évolués et faciles d'utilisation pour instancier ou retrouver des composants existants comme les annuaires de composants. Ils bénéficient notamment de la séparation des méta-données, qui nous permet d'ajouter la partie description de la P1S au code technique;
- Grâce à l'encapsulation du composant dans un conteneur, un composant bénéficie, de façon transparente au développeur du composant, de services techniques. Un service technique écrit sous forme de composant bénéficie donc des autres services techniques tel que le service de distribution. Ce service largement utilisé permet aux composants d'interagir malgré leur distribution sur le réseau. On peut aussi imaginer un service de persistance rendant les données persistantes dans une base de données étant lui-même exécuté dans une transaction.

Le modèle à composants semble donc être le plus apte à répondre à nos attentes : il définit une programmation modulaire qui permet à plusieurs P1S de cohabiter en capitalisant du code. Il permet l'expression des interfaces, facilitant l'association entre la description d'une P1S et son code. Ce modèle facilite l'adaptation grâce à l'assemblage ou le réassemblage des modules logiciels élémentaires formant les différentes versions d'un même service technique. Le modèle à composants fournit des mécanismes pour facilement déployer, exécuter, retrouver les services techniques sous forme de composants. Enfin un composant étant encapsulé dans un conteneur, il bénéficie lui-même de services tels que le service de communication.

De plus, il s'avère, d'après les expérience menées [HéR 04c] comme celles décrites dans la partie "état de l'art" et celle que nous allons décrire autour du service transactionnel dans la partie suivante, que le modèle à composants est adapté à la conception, au développement et à l'exécution des composants techniques. Nous avons déjà défini plusieurs concepts pour une conception intelligente de ces services techniques : les personnalités

P1S et L2S.

1.2.2 Projection sur le modèle Fractal

Le modèle à composants que nous avons retenu est le modèle Fractal [BRU 04]. Ce modèle est défini comme étant un modèle à composants modulaires et extensibles, couvrant l'ensemble du cycle de vie d'un composant (ce qui correspond aux trois dernières couches de la figure 1.1).
Tout d'abord, Fractal définit une hiérarchie de concepts qui vont des concepts fondamentaux les plus abstraits (seconde couche) tels que le concept de composant, contrôleur, d'introspection, d'intercession aux concepts les plus concrets (troisième et quatrième couche) tels que le concept de liaison et un système de typage. Chacun de ces concepts est optionnel : pour définir son propre modèle, on choisit uniquement les concepts nécessaires à partir desquels on peut en définir de nouveaux. Grâce à cette flexibilité, on obtient des composants dont les caractéristiques sont variables : des composants extensibles.
Les concepts de la programmation par composants que nous avons détaillée dans la partie précédente font partie des bases du modèle Fractal. D'autres comme la réflexivité, les composants composites et partagés, l'introspection et l'intercession sont spécifiques à Fractal. Ils sont utiles à la conception des services techniques car ils facilitent la conception et la gestion de services adaptables. Enfin, il est possible d'étendre ces concepts, c'est ce que nous avons fait en définissant l'assemblage entre les composants applicatifs et les services techniques.

Concepts de la programmation par composants

Etant basé sur le principe de séparation des préoccupations, ce modèle intègre les concepts de *composant*, d'*interface* et de *contrôleur* qui sont nécessaires à la conception des services techniques tels que nous l'avons vu précédemment. En effet, ils permettent de modéliser la séparation de préoccupations qui intervient à trois niveaux :

- la séparation des différentes entités d'exécution ou de code est représentée par la notion de composant. Cela facilite la conception des services techniques sous forme modulaire et facilite l'assemblage en réutilisant des composants élémentaires;
- la notion d'interface permet de modéliser la séparation entre le service fourni et son implantation (sous forme de "contenu" dans la terminologie Fractal). Dans notre modélisation, une P1S prend la forme d'une interface;
- la séparation entre la logique applicative et les services techniques et de façon plus générale le fonctionnement de la plate-forme est permise par l'inversion de contrôle qui se traduit dans le modèle Fractal par le concept de contrôleur. Le terme contrôleur (ou membrane), utilisé dans Fractal est à rapprocher du terme conteneur des EJB et de CCM. Cette séparation permet un contrôle accru sur l'exécution des services techniques. Nous verrons dans la partie 2.3 comment nous exploitons cette caractéristique pour définir l'assemblage entre les composants applicatifs et composants techniques.

Concepts spécifiques à Fractal

Outre les concepts génériques de la programmation par composant, Fractal a des spécificités qui correspondent à nos besoins.
Les différents niveaux d'abstraction du modèle et son extensibilité permettent de ne pas définir tous les concepts. C'est pourquoi, concernant les services techniques, il existe peu de spécification : ce modèle ne définit pas exactement comment modéliser les services techniques, ce qui nous laisse toute latitude quant aux propositions.
De plus, contrairement aux modèles EJB ou Fractal avec le service de persistance ou de transaction, un composant Fractal n'a pas par défaut de service technique qui lui est associé. Si cela avait été le cas, il aurait été ardu de construire des services techniques à base de composants Fractal puisqu'un service technique aurait lui-même bénéficié de son service. Ce qui aurait créé un cycle dans la conception.
D'autre part, ce modèle intègre plusieurs concepts qui permettent de rendre les applications développées grâce à ce modèle plus adaptables.

- Le modèle Fractal définit la notion de *composant composite*. Ainsi une composition de composants est vue comme étant elle même un composant et ceci à n'importe quel niveau, on dit que c'est un modèle récursif. L'implantation d'un service technique peut donc être vu comme un unique composant. Plusieurs implantations d'un service peuvent être regroupées dans un composant composite offrant plusieurs interfaces. On peut donc modéliser un service technique comme un composant composite associant les différents composants remplissant les tâches élémentaires. Ce composant composite peut "abriter" plusieurs P1S.
- Fractal définit le *partage de composants* : un composant peut faire partie de la composition de plusieurs composants simultanément. Ainsi, si une ressource (donnée ou traitement) est partagée par plusieurs services, elle peut être représentée par un composant qui est partagé par ces services. Deux P1S peuvent donc partager un même composant ce qui facilite la gestion de la cohabitation de deux P1S différentes.
- Le modèle Fractal repose aussi sur la *réflexivité*, qui est définie comme étant " la capacité d'un composant à manipuler, comme il manipulerait une donnée, les entités qui représentent ses états d'exécution durant sa propre exécution. Cette manipulation peut prendre deux formes : l'introspection et l'intercession ".
  - Les composants Fractal possède donc des capacités d'*introspection*[2] : ils sont capables de décrire leurs interfaces, leurs liaisons, leur contenu (en terme de sous-composants). Cela permet d'avoir une image du système en vue de le modifier. Ceci nous aidera à mettre en œuvre l'adaptation dynamique des services techniques.
  - Cette capacité d'introspection permet notamment aux composants de fournir une information sur leur composition. Additionné à la capacité d'*intercession*[3], elle permet alors de dynamiquement reconfigurer l'application. Les services techniques réalisés en composants Fractal sont ainsi plus malléables. En effet,

2. introspection : la capacité d'un composant à observer et raisonner sur son propre état d'exécution [BRU 04].

3. intercession : capacité d'un composant à modifier son propre état d'exécution ou d'altérer sa propre interprétation ou sémantique [BRU 04]

cette méthode de réalisation de services techniques permet de facilement ajouter, retirer ou remplacer un élément élémentaire du service par un autre qui dispose des mêmes interfaces. De plus, ces manipulations peuvent se faire de façon dynamique. Il devient donc plus facile de générer rapidement des services techniques adaptés à leur environnement d'exécution.

Extension de Fractal

Enfin, ce modèle définit séparément les concepts de leur implantation ce qui permet un plus grand niveau d'abstraction : la solution que nous allons fournir, bien que basée sur ce modèle spécifique, sera basée sur des concepts fondamentaux qui sont à la base d'une programmation par composants élégante [RIV 00].
Notons de plus, que ce modèle fournit aussi une implantation de référence qui reprend la plupart des concepts définis et qui nous a servi à prototyper nos propositions. Sur la figure 1.1, on pourrait placer la couche correspondant à cette implantation au niveau de la quatrième couche.
La section suivante détaille les extensions que nous fournissons à Fractal. Cette extension se place par définition sous le modèle Fractal, elle repose sur l'implantation concrète de la notion de conteneur dans Julia. Après avoir défini la forme prise par le service technique : plusieurs P1S implantées grâce à des composants composés, partagés, dotés de capacités d'introspection et d'intercession, décrites selon nos spécifications, nous allons définir l'assemblage que ces services vont former avec les composants applicatifs.

1.3 Modèle d'assemblage : sous-contrôleurs

En ce qui concerne l'assemblage des composants techniques entre eux, elle ne diffère pas de celle des composants applicatifs. La phase d'assemblage que nous allons décrire ici est celle des composants techniques avec les composants applicatifs.
En effet, l'assemblage des composants applicatifs et des services techniques des modèles à composants industriels n'est pas adaptée à la solution de conception des services techniques que nous proposons : c'est un assemblage fixe et que l'on retrouve de façon diffuse dans le code de l'ORB et du conteneur et qui n'est pas explicitement exprimé. Ceci empêche l'adaptation de cet assemblage.
De plus, le modèle Fractal, même s'il nous fournit beaucoup des concepts nécessaires, ne définit pas cette phase. En effet, le modèle Fractal n'est pas initialement un modèle à composants métiers, il n'intègre pas les services techniques comme les modèles EJB ou CCM. Cependant, le modèle à composants métiers peut être vu comme un raffinement du modèle général Fractal qui définit le tronc commun des concepts de base pour une bonne programmation par composants quelque soit le domaine d'application.
La solution que nous proposons est donc un raffinement de Fractal, orienté métier dans la mesure où elle intègre les services techniques. Elle permet, de façon simple l'ajout/retrait ou changement de personnalités de service technique associé à chaque composant à travers leur assemblage, permettant ainsi l'adaptation des services techniques.

1.3.1 Utilisation des services techniques par les composants applicatifs

Tout d'abord, la solution d'assemblage dont nous avons besoin doit maintenir la séparation entre le code applicatif et le code technique, permettre l'ajout/le retrait ou le changement facile et dynamique de service technique de l'assemblage. Il doit aussi respecter l'aspect récursif des composants composites.
C'est grâce à la propriété de flexibilité du conteneur ouvert que nous définissons cet as-

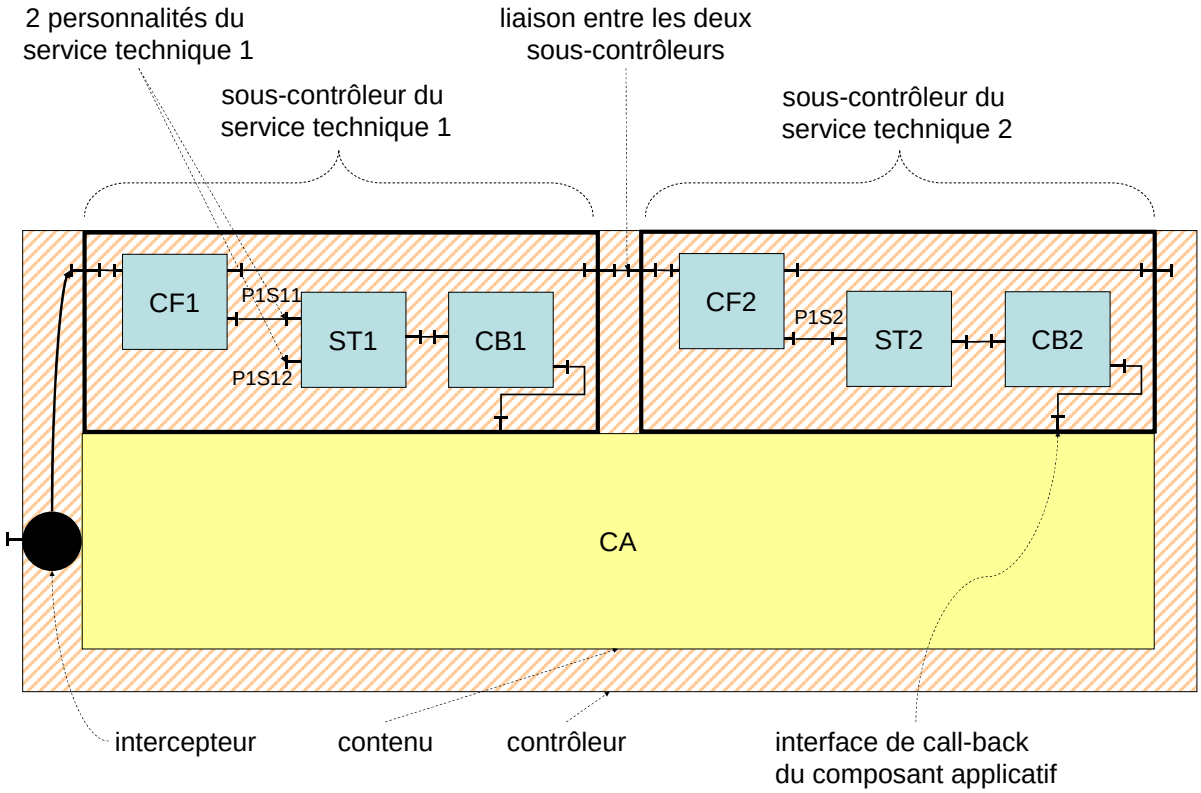

FIG. 1.4 – *Exemple d'assemblage d'un composant applicatif et de deux services techniques.*

semblage. L'exécution de services techniques associées à une interface d'un composant applicatif se fait donc par l'ajout d'un intercepteur sur cette interface et de sous-contrôleurs associé à cet intercepteur. Ainsi à chaque exécution d'une méthode du composant applicatif, le service technique est exécuté. L'ajout ou le retrait d'un contrôleur correspond à l'ajout ou le retrait d'un service technique au composant applicatif de ce contrôleur.
Sur la figure 1.4, un exemple de composant applicatif avec deux services techniques est représenté : on distingue le contenu (partie centrale non rayée) du conteneur (partie périphérique rayée). Ce composant possède une interface requise à laquelle est associé un intercepteur (cercle noir). Cet intercepteur est associé à deux sous-contrôleurs. Détaillons le rôle de chaque élément de cette figure.

Rôle de l'intercepteur

Un intercepteur est placé sur une interface du composant applicatif. Fractal en définit deux sortes : interfaces fonctionnelles et interfaces de contrôle. Ce sont les interfaces fonctionnelles qui nous intéressent ici dans la mesure où c'est l'exécution du composant applicatif qui déclenche l'exécution des services techniques. Les interfaces de contrôle personnifient les capacités d'introspection du composant (connaissance et modification des attributs, des liaisons reliant les composants d'une composition, etc). On pourrait aussi ajouter des intercepteurs sur ces interfaces, mais cela ne rentre pas dans le cadre que nous avons défini.

Réification par les sous-contrôleurs

Un appel de méthode, est intercepté puis réifié par les sous-contrôleurs. Cette réification consiste à faire appel à une ou plusieurs méthodes du service technique avant ou après l'exécution du composant applicatif (nous les appelerons méthode "pré" et "post"). Par exemple pour le service transactionnel, la réification consiste à commencer la transaction (begin) puis exécuter l'appel de méthode fait sur le composant et terminer la transaction (commit ou abort). On peut introduire quelques subtilités en passant les paramètres de l'appel de méthode au service technique, par exemple pour le service de transaction, on pourra passer les valeurs des données modifiées pour les sauvegarder dans la table des compensations. On peut aussi empêcher l'exécution du service technique, par exemple si le service de sécurité n'identifie pas le client de l'appel de méthode comme autorisé.

Structure des sous-contrôleurs

Le modèle Fractal ne spécifie pas la conception des sous-contrôleurs, juste le fait que l'on peut, selon les besoins, en ajouter ou en retirer du contrôleur. Nous définissons donc un sous-contrôleur comme un composant composite.
Ainsi les liaisons du contrôleur sont explicitement et automatiquement gérées : entre les différents sous-composants du sous-contrôleur ou entre les différents sous-contrôleurs.
Un sous-contrôleur de service technique se compose de :

- un composant de façade (sur la figure CF1 et CF2) qui est sollicité par l'intercepteur. Il effectue les appels sur le service technique comme défini ci-dessus, selon les spécificités du service technique. L'interface générique de l'intercepteur n'étant pas forcément la même que celle du service technique, ce composant de façade joue le rôle d'adaptateur (ou wrapper). Il possède donc une interface serveur compatible avec celle de l'intercepteur et une interface client compatible avec la P1S choisie;
- un service technique (sur la figure ST1 et ST2) sous forme d'un composant exposant plusieurs interfaces, dont certaines sont des P1S. D'autres peuvent être des interfaces d'administration. L'une de ces P1S est liée avec le composant de façade. Le choix de la P1S ne relève pas de l'assemblage mais de la phase de déploiement qui sera décrite dans le chapitre 4;
- un composant de "call-back" (sur la figure CB1 et CB2) dont nous détaillons l'utilité dans la section suivante (1.3.2.

Dans Fractal, un composant pouvant être partagé, le service technique peut être partagé par plusieurs sous-contrôleurs de différents composants applicatifs. Il est aussi possible facilement d'interchanger un service technique par un autre comme on échange un composant par un autre.
De plus, dans cet assemblage, le fait qu'un composant applicatif puisse bénéficier de l'une ou l'autre des P1S d'un service technique, se traduit par la possibilité de modifier la liaison entre le composant de façade et le service technique. Sur la figure, le composant de façade CF1 peut être lié à P1S11 ou P1S12.
Enfin, la séparation du code technique et du code applicatif est garantie dans la mesure où les services techniques sont inclus dans les sous-contrôleurs. Or, ces sous-contrôleurs font partie du contrôleur qui est à différencier du code applicatif.
D'autre part, pour associer au composant applicatif plusieurs services techniques, on ajoute au contrôleur autant de nouveaux sous-contrôleurs que de services techniques (sur la figure le composant applicatif est associé à deux services techniques ST1 et ST2). Ces sous-contrôleurs s'exécutent à la suite les uns des autres grâce à la liaison qui existe entre le composant de façade d'un sous-contrôleur et le sous-contrôleur suivant qui en fait de même (sur la figure, le composant de façade CF1 est lié au sous contrôleur suivant). Tout d'abord tous les sous-contrôleurs exécutent la méthode "pré" du service technique à leur tour puis le composant applicatif s'exécute et la méthode "post" de chacun des services techniques dans l'ordre inverse.
Concernant la récursivité des composants applicatifs, la cohérence est maintenue. Les différents composants d'une composition composite (le composant père et ses enfants sur tous les niveaux) peuvent tous bénéficier de services techniques différents à travers des P1S différentes. En effet, chaque composant applicatif est associé a un contrôleur (ainsi que ses sous-contrôleurs) qui lui est propre, les services techniques étant partagés par les sous-contrôleurs.
Par conséquent, l'assemblage que nous avons défini respecte les trois contraintes posées : séparation du code applicatif du code technique, facilité d'ajout/retrait de services techniques et de changement de P1S, respect de la récurivité. L'assemblage proposé maintient la séparation du code fonctionnel et technique en associant les services techniques au contrôleur, le contrôleur étant à opposer au contenu " applicatif " du composant. Il permet l'ajout/retrait facile d'un service technique qui se traduit par l'ajout/retrait d'un sous-contrôleur. Pour changer de P1S, on peut soit interchanger le service technique soit modifier la liaison entre le composant de façade et le service technique. Cet assemblage respecte l'aspect récursif des composants Fractal en fournissant les services techniques adaptables à tous les niveaux de récursivité des composants applicatifs.

1.3.2 Composants de "call-back"

Comme nous l'avons vu précédemment, l'un des sous-composants de nos sous-contrôleurs est le composant de "call-back". Nous l'avons introduit afin de tenir compte de l'existence de code dit de retour ou "call-back" ([HéR 03a], [HéR 04c]).
On peut donner comme définition du code de "call-back" : "*code utilisé par certains services techniques afin de connaître des informations spécifiques ou d'effectuer un traitement spécifique à un composant applicatif particulier*". Ce code est développé par le développeur

qui a connaissance à la fois de la logique applicative du composant et des services dont a besoin le service technique. Le code obtenu est spécifique à la fois au composant applicatif et au service technique.
Il apparaît, dans les modèles à composants industriels, sous la forme de méthodes de "call-back" au sein du composant applicatif. Il est appelé par le service technique par l'intermédiaire du conteneur du composant applicatif ou par le code applicatif lui-même. Par exemple, les méthodes de l'interface javax.ejb.SessionBean ou dans javax.ejb.EJBContext sont des méthodes de "call-back" utilisés par les services techniques de gestion de cycle de vie, transactionnel et de sécurité : la méthode ejbCreate permet d'effectuer un traitement en début de cycle de vie d'un composant.
Jusqu'à présent, ce code avait une représentation faible : il apparaissait, sans distinction particulière, sous la forme de méthodes au sein des objets implantant le code applicatif. Il existe deux raisons à cela : la première est que dans les modèles industriels, il n'a pas été fait un grand effort d'abstraction pour modéliser les services techniques dans la mesure où ils étaient considérés comme liés à l'implantation du conteneur. La seconde raison est plus technique : dans ces modèles industriels, un composant prend la forme d'un ensemble prédéfini d'objets. La notion de sous-composant n'existe pas, il n'y avait donc pas d'intérêt particulier à le modéliser de façon indépendante.
Cependant, cette solution n'est pas très élégante et elle ne facilite pas l'adaptation des services techniques : il n'est pas possible d'ajouter ou de retirer de méthode de "call-back" du composant applicatif.
Pour résoudre ce problème, nous proposons de modéliser les méthodes de "call-back" d'un service technique sous forme de composants indépendants (sur la figure CB1 et CB2). Si nécessaire (tous les services techniques n'en ont pas besoin), ce composant est ajouté au sous-contrôleur de son service technique. L'un des avantages du fait que tous les éléments sont modélisés sous forme de composants est que le composant de "call-back" (sur la figure CB) est directement utilisé par le service technique (sur la figure ST) par l'intermédiaire d'une liaison. Le composant de "call-back" utilise le composant applicatif à travers l'interface requise notée " interface de call-back du composant applicatif " sur la figure.
Une telle solution nous apporte la modularité nécessaire à l'adaptabilité : si l'on ajoute ou retire un service technique au composant applicatif, on peut alors ajouter ou retirer le composant de "call-back" associé. De plus, il est possible de représenter différentes implantations des méthodes de "call-back" par plusieurs composants différents et de les interchanger.
Notons que la notion de L2S, définie au niveau de la conception, se projette dans le modèle à composants sous la forme de plusieurs P1S. Par exemple, la figure 1.4 peut représenter la projection d'un L2S contenant la personnalité P1S1.1 et la personnalité P1S2. Ainsi ajouter les personnalités d'un L2S revient à ajouter plusieurs fois les personnalités d'une P1S.

1.4 Conclusion

Dans la partie problématique, nous avions souligné le besoin d'intergiciels plus souples, permettant aux services techniques de s'adapter aux besoins applicatifs et aux contraintes

de l'environnement d'exécution. Afin de résoudre ce problème, nous avons remis en cause le cycle de vie des applications et des services techniques. Dans cette section, les trois premières phases ainsi que les projections entre ses phases sont détaillées : conception, développement et assemblage.

1.4.1 Abstraction à travers la notion de personnalité

Afin d'avoir le niveau d'abstraction nécessaire à l'adaptabilité des services techniques, la phase de conception permet de modéliser une implantation (modèle ou version) d'un service technique par une P1S et un ensemble cohérent de ces P1S par un L2S. La représentation fournie et les informations sur la qualité de service serviront à faire un choix lors de l'adaptation.
La difficulté à mettre en œuvre une telle solution réside dans la complexité de l'expression de la qualité de service. Deux problèmes apparaissent : récolter les informations de qualité de service (des services techniques et de l'environnement) et les exprimer dans un langage compris par toutes les parties (développeur du service technique, du composant applicatif et de la plate-forme).
Nous avons déjà brièvement abordé l'obtention des informations de qualité de service des services techniques dans la partie 2.1.4. L'obtention d'informations de qualité de service de l'environnement sera traitée dans les parties 4.2.2 "Moniteur" du chapitre 4.
Concernant l'expression de la qualité de service, nous avons déjà fourni un formalisme dans la partie 2.1.3 "contenu de la P1S", cependant il reste à définir une ontologie commune. Ce travail pourrait faire partie d'un effort de normalisation entre les différents métiers de la programmation par composants.

1.4.2 Flexibilité grâce au modèle à composants

Pour développer les services techniques ainsi décrits, nous avons choisi le modèle à composants. Il facilite le développement des services en fournissant une vue modulaire et hétérogène des sous tâches effectuées par le service. Il permet aussi une reconfiguration facile des assemblages au sein du service technique et entre les composants applicatifs et les services techniques. Le modèle à composants plus précisément choisi est le modèle Fractal, un modèle initialement conçu pour les composants intergiciels. Ce modèle nous fournit à la fois les concepts fondamentaux de la programmation par composant et un modèle concret avec son implantation de référence. Il prend en compte la possibilité d'ajout de nouveaux concepts et nous fournit les outils de développement nécessaires.
Cependant, il n'est pas toujours évident de séparer le code applicatif du code technique [KIN 02]. Il faut donc nuancer le fait que les interactions qui existent entre code technique et code applicatif sont totalement modélisables grâce à une représentation modulaire des services techniques. Toutefois, les composants de call-back amènent un premier élément de réponse : il permettent de modéliser les interactions du service technique avec le composant applicatif. On a par exemple la possibilité de déclencher un traitement spécial s'il survient une erreur lors de l'exécution d'un composant.

1.4.3 Cohérence et expressivité grâce aux composants sous-contrôleurs

La phase d'assemblage s'effectue entre les composants applicatifs et les services techniques grâce au contrôleur. Celui-ci est notamment constitué de sous-contrôleurs composés du service technique, d'un composant faisant la passerelle avec le contrôleur et d'un composant de "call-back. Ainsi il est facile d'ajouter/retirer ou de changer un service technique par un autre ou encore de changer de P1S. De plus, chaque composant (qu'il soit englobé dans un composant composite ou pas) peut bénéficier de la P1S qu'il lui est approprié.
Si la récursivité est maintenue, on peut se demander s'il en est de même pour le partage des composants. En effet, dans la programmation récursive, on identifie une application à un composant composite. Donc, lorsqu'un composant est partagé entre plusieurs composants composites, il s'exécute potentiellement dans des contextes applicatifs différents. Est ce que dans chacun des contextes d'exécution, le sous-composant applicatif devrait être exécuté avec les mêmes services techniques ? Cela ne semble pas évident car les services techniques sont choisis aussi en fonction des besoins de l'application. Notre représentation ne permet pas de modéliser les différents contextes applicatifs dans lesquels le sous-composant applicatif pourrait s'exécuter.
Enfin, nous tenons compte d'une réalité des services à composants : le code de call-back qui se trouve entre certains services techniques et les composants applicatifs et proposons une technique adéquate à son implantation.

Il est maintenant nécessaire de fournir un service pour gérer dynamiquement les adaptations des services techniques, en fonction des besoins de l'applications et des changements de l'environnement. Ce service doit :

- gérer les différentes personnalités d'un même service;
- détecter les modifications de l'environnement d'exécution;
- être capable de fournir au composant applicatif les services techniques adéquats.

2

Gestion dynamique de l'adaptation

Dans le chapitre précédent, une première proposition a été faite concernant la conception des services techniques. Nous avons ajusté les modèles de conception, de développement et d'assemblage des services techniques pour une plus grande flexibilité. De plus, nous avons motivé l'expression des différentes implantations d'un service technique à travers les notions de P1S et de L2S, assemblages documentés et prédéfinis de composants. Les services techniques tels que nous les avons définis, composés avec l'application constituent le système de base de notre système pour l'adaptation des services techniques.
Cette seconde partie des propositions présente un service de gestion de l'adaptation dynamique des services. La solution proposée doit permettre, le plus souvent possible, à chacune des applications du système étudié à base de composants de bénéficier des services techniques les mieux adaptés (ou tout au moins adaptés) au contexte d'exécution. Cela signifie que chaque application, selon ses besoins, aura la possibilité d'utiliser un service technique différent de ceux utilisés par une autre application. Ainsi on a un gain en terme de qualité de service.
Cela signifie aussi qu'il sera possible d'ajouter, de retirer au système des services techniques ou encore d'en changer la version ou le modèle. De cette façon, on espère aussi faire des gains en terme de temps de déploiement et de place mémoire. Pour cela, on n'embarquera sur la machine et l'on ne déploiera que le code nécessaire.
Nous venons de définir brièvement le but de l'adaptation préconisée. Dans la section 2.1, nous allons déterminer les éléments du système à adapter ainsi que les éléments qui effectuent l'adaptation. Nous définirons aussi les événements qui déclenchent l'adaptation ainsi que les facteurs pris en compte pour décider des adaptations à mener. Cette description a pour but de donner une idée générale des principes de fonctionnement du système.
La section 2.2 sera consacré au service de gestion de l'adaptation qui met en œuvre ces principes de fonctionnement. Ce service est constitué de différents éléments (contrat, moniteur, annuaire et coordinateur) dont nous détaillerons le fonctionnement et les interactions.

2.1 Gestion de l'adaptation

Nous avons vu, lors de la présentation de l'adaptabilité, qu'il existait deux possibilités quant à la gestion de l'adaptation : soit elle est décidée par un intervenant extérieur (administrateur ou utilisateur), soit par le système lui-même (voir figure 2.1). Il semble peu raisonnable et peu compatible avec la notion de programmation par composant et de service technique de demander à un utilisateur d'intervenir, par exemple en choisissant quel modèle de service technique, quelle implantation, quelle version il veut utiliser. On pourrait aussi solliciter l'administrateur qui a une meilleure connaissance du système, mais ces interventions doivent être limitées à la période d'installation du système. Il semble donc plus indiqué de gérer les adaptations au niveau du conteneur.

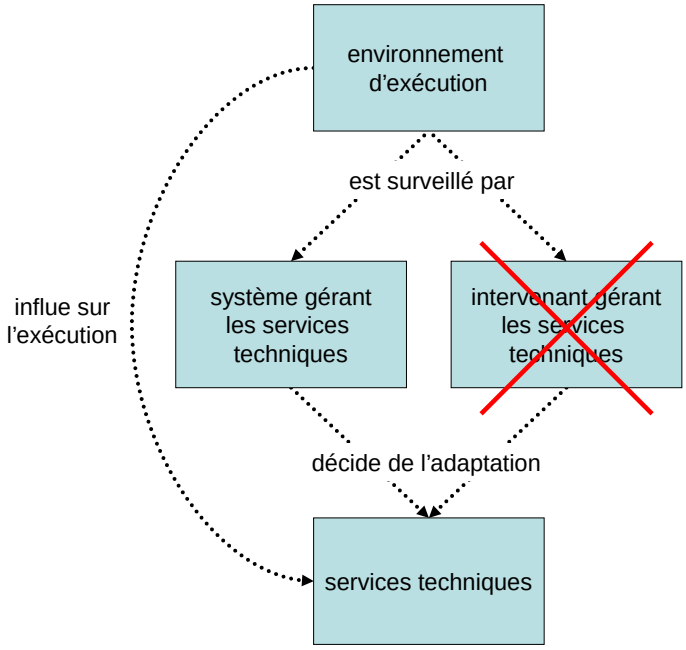

FIG. 2.1 – *Schéma des différents éléments intervenants dans l'adaptation.*

Les éléments du système que l'on veut adapter et que l'on a besoin de connaître pour l'adaptation sont : les services techniques, les composants applicatifs (i.e. le conteneur et le contenu) et l'environnement d'exécution.

- Si le composant applicatif devait gérer l'adaptation, cela surchargerait notablement le travail des développeurs. De plus, on ne maintiendrait pas la séparation du code applicatif et du code technique.
- Il ne semble pas non plus du ressort d'un composant gérant l'environnement d'exécution de gérer les adaptations si l'on veut maintenir la séparation des tâches préconisée dans le modèle à composants.
- On peut envisager un système où les services techniques s'adaptent eux-mêmes. Le fait que chaque service technique s'adapte lui-même pose plusieurs problèmes.

Le développeur de services techniques voit sa tâche s'alourdir car il doit gérer lui-même l'adaptation. De plus, il n'existe pas d'intermédiaire qui mette en relation les composants applicatifs et les services techniques dont ils ont besoin. Enfin on ne capitalise pas les solutions trouvées pour l'adaptation d'un service technique, ce qui ne permet pas de les réutiliser pour d'autres services techniques.

– Enfin, on peut envisager la gestion de l'adaptation par le conteneur. Cependant le conteneur est un intermédiaire entre le contenu applicatif et les services techniques. Il va faire partie lui-même des éléments à adapter. Or, nous avons vu qu'il était préférable de dissocier les éléments à adapter de ceux qui les adaptent.

La solution semble donc de créer un nouvel élément qui va gérer l'adaptation de l'ensemble de services techniques et de leur liaisons avec les conteneurs. Cette solution ne surcharge pas le travail des développeurs et permet une gestion globale. On veillera à ce que cet élément joue le rôle d'intermédiaire de la gestion des relations entre les composants applicatifs et les services techniques.
Le même genre de réflexion que celle que nous venons de mener sur l'utilité d'un élément externe permettant au système de s'adapter a déjà été menée. Elle fait partie de la réflexion menée autour de la notion de réflexivité.

2.1.1 Application de la notion de réflexivité à notre problématique

Au cours de la recherche en informatique, la réflexivité s'est montré être un concept efficace pour la conception de systèmes auto-adaptables [MCK 04], c'est-à-dire des systèmes gérant eux même leur adaptation. En effet, un système réflexif est un système qui a une représentation de lui même et peut en fonction de cette représentation modifier son propre comportement. Ainsi, un système réflexif peut adapter son comportement en fonction des connaissances qu'il a de lui même [KIC 91]. Nous appliquons donc le principe de réflexivité aux services techniques afin de les rendre adaptables.
L'application du concept de réflexivité au domaine plus particulier de la programmation orientée objet s'est traduit par l'adjonction à un objet dit "de base" d'un objet dit "meta" qui donne des informations sur le premier et permet d'en modifier le comportement. Par mimétisme, en programmation par composant, à un composant de base, on adjoint un méta-composant.

2.1.2 Définition du niveau de base et du niveau méta

Le niveau de base est constitué des éléments que l'on veut adapter (intercession) ainsi que les éléments que l'on doit connaître pour faire cette adaptation (introspection). Le niveau de base de notre système est donc constitué :

– des composants applicatifs. Il sont développés sous forme de composants Fractal sans se préoccuper des futurs services techniques qui leur seront adjoints ainsi qu'une description de leurs besoins en terme de services techniques;
– des services techniques tels qu'ils ont été décrits dans les sections 1.1 et 1.2 du chapitre 1;

– les liaisons existantes entre les services techniques et les composants applicatifs telles qu'elles ont été décrites dans la section 1.3 du chapitre 1;
– et de l'environnement d'exécution. Les éléments qui le constituent ont été décrits dans la section 1.1.1 du chapitre 1 partie I.

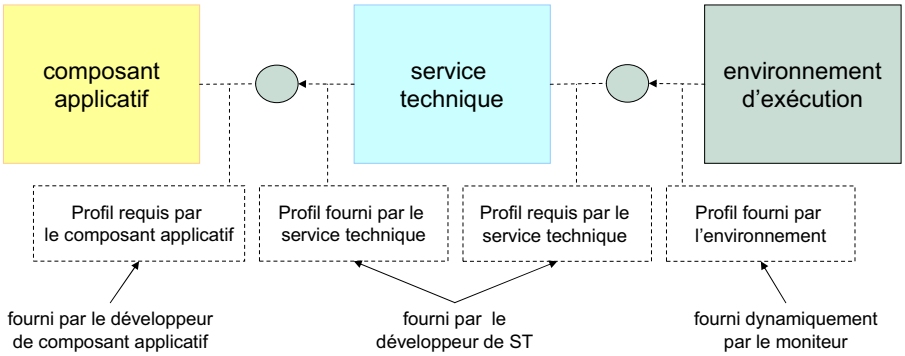

FIG. 2.2 – *Relations entre application, service technique et environnement d'exécution*

Les services techniques fournissent une prestation aux composants applicatifs et l'environnement d'exécution fournit aux services techniques un cadre d'exécution avec une qualité de service plus ou moins bonne (cf figure 2.2).
En effet, on voudrait pouvoir agir sur les services techniques et leur assemblage avec les composants applicatifs. Les paramètres "environnement d'exécution" et "besoin de l'application" interviennent de deux façons : ce sont leurs modifications qui déclenchent une adaptation et l'on tient aussi compte de ces paramètres pour décider de l'adaptation à mener. Pour mener l'adaptation on a donc besoin d'une représentation des services techniques, de leur assemblage avec les composants applicatifs et de l'environnement d'exécution. On retrouve ces éléments dans la partie basse, niveau de base, de la figure 2.3.

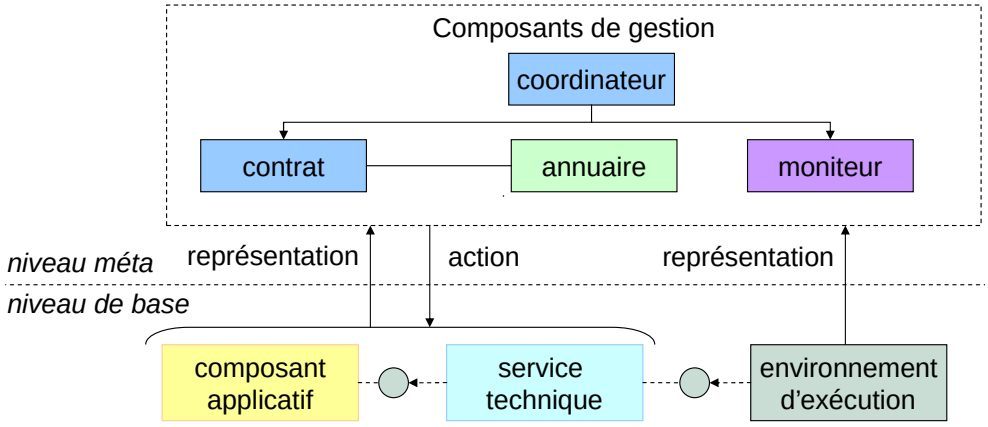

FIG. 2.3 – *Niveau de base et méta du système.*

Le niveau méta regroupe un ensemble de méta-composants qui vont correspondre aux

éléments du niveau de base :

- des méta-composants appelés "contrats" et représentant chacun un assemblage :
  - d'un composant applicatif;
  - avec ses services techniques;
  - par l'intermédiaire des liaisons symboliques qui les lient;
- un méta-composant appelé "moniteur" représentant l'environnement d'exécution;
- un méta-composant appelé "annuaire de services techniques" dont chaque entrée correspond à une personnalité de service technique

Un dernier composant est adjoint pour la cohérence de l'ensemble. Nous appellerons les composants du niveau méta " composants de gestion " car ils permettent de gérer l'adaptation du niveau de base. On les retrouve dans la partie haute, niveau méta, de la figure 2.3.

Les interactions entre niveau de base et niveau méta sont de deux types : introspection et intercession. Il existe une relation d'introspection :

- entre le contrat et les éléments de base "composant applicatif / services techniques / liaison entre les deux"
  - Le composant applicatif peut préciser la liste des services techniques qu'il utilise. Le contrat contient une référence sur le composant applicatif ainsi que cette liste des services techniques requis.
  - Le service technique peut fournir une description méta du service qu'il fournit. Le contrat contient une référence sur le service technique et cette description méta.
- entre l'annuaire et les services techniques : l'annuaire contient les descriptions méta du service fourni par les services techniques;
- entre le moniteur et l'environnement d'exécution : le moniteur contient une description méta de l'environnement d'exécution.

Il existe une relation d'intercession entre un contrat et les liaisons symboliques qui existent entre le composant applicatif et ses services techniques : le contrat peut modifier ces liaisons pour que le composant applicatif utilise les personnalités de services techniques les mieux adaptées.

2.1.3 Boucle d'adaptation

La boucle d'adaptation est un cycle dont les noeuds sont les différents états du système : état dans lequel les services techniques ne sont pas adaptés, état où les services techniques sont adaptés. En plus de ces deux états principaux, on doit ajouter les états initiaux et terminaux qui correspondent au début et à la fin du cycle de vie du système (voir figure 2.4).
Les transitions entre états sont déclenchées par deux événements : le changement de l'environnement et l'adaptation des services techniques.
Le changement d'environnement a été identifié à deux étapes du cycle de vie d'une

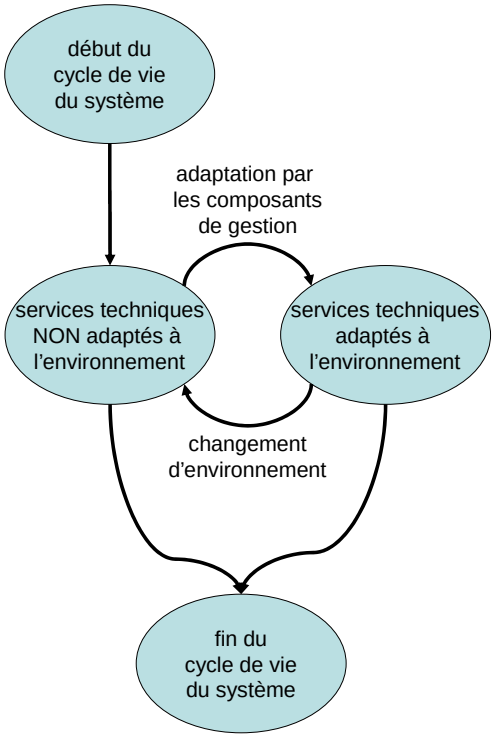

FIG. 2.4 – *Boucle d'adaptation.*

application :

- lorsque l'application est exécutée dans deux environnements différents, par exemple lorsqu'une application cliente peut s'exécuter sur deux terminaux différents (ex : un PDA et un PC).
- lorsque l'application change de contexte au cours de son exécution. On peut prendre pour exemple une application fonctionnant sur un PDA tout d'abord en mode déconnecté. Puis le PDA vient se connecter à un PC, l'application fonctionne alors dans un mode connecté. Ce cas est de plus en plus fréquent avec le développement des appareils mobiles tels que les PDA ou les téléphones mobiles comme nous le montrent [WAN 01] et [BLA 00].

Dans le premier cas, une adaptation statique au moment du déploiement suffira puisque l'environnement n'évolue pas après le déploiement de l'application. Cette adaptation correspond à l'exécution d'une seule boucle d'adaptation : on passe de l'état initial à l'état non adapté, puis on effectue une adaptation. Il n'est pas possible de revenir à l'état non adapté.

Le second cas nécessite l'adaptation dynamique dans la mesure où elle peut intervenir à n'importe quel moment. Cela correspond à l'exécution à plusieurs reprises de l'adaptation : le passage successif de l'état initial à l'état non-adapté, puis une boucle entre ces deux derniers. Cela dit, notre solution devra limiter le nombre de ces adaptations car elles risquent de se révéler coûteuses.

Pour un système adaptable statiquement, chaque phase du cycle de vie (conception, déve-

loppement, assemblage, déploiement et exécution) est exécutée une seule fois. Pour un système adaptable dynamiquement, les trois premières phases (conception et développement) s'exécutent une fois puis les trois dernières phases (assemblage, déploiement et exécution) se répètent en boucle, l'adaptation étant vue comme un réassemblage/redéploiement d'une partie du système.

2.1.4 Conclusion

Dans cette section 2.1, nous avons détaillé le fonctionnement du mécanisme de la boucle d'adaptation. Son but étant d'améliorer la qualité de service fournie par la plate-forme aux composants par l'intermédiaire de services techniques bien adaptés à leur environnement ainsi que de limiter la taille du code embarqué et déployé et le temps de déploiement. Cette adaptation est déclenchée par le changement d'environnement d'exécution et tient compte de deux facteurs : le nouvel environnement et les besoins de l'application. Elle est gérée de façon automatique par le système dont nous allons détailler la composition et le comportement dans la section suivante.

2.2 Système de gestion de l'adaptation

Cette partie décrit le fonctionnement des méta-composants du service de gestion de l'adaptation dynamique des services techniques. Situé au dessus du niveau de base, c'est ce service qui permet met en œvre de la boucle d'adaptation qui vient d'être décrite. Ce niveau est constitué d'un ensemble de composants de gestion dont nous détaillons d'abord le fonctionnement individuel, puis les interactions [HéR 04a].

2.2.1 Définition des composants de gestion du niveau méta

Les composants de gestion constituent le niveau méta du système d'adaptation de la relation entre l'application et les services techniques. Cette adaptation consiste à localiser, choisir, configurer et utiliser les P1S et L2S. Pour effectuer ces tâches, nous définissons des composants de gestion du système : un coordinateur du système, des contrats, un annuaire et les moniteurs (cf. figure 2.5). Le coordinateur du système est le point d'entrée du système pour l'administrateur et en maintient la cohérence. Le contrat est le meta-composant représentant la liaison entre le composant applicatif et le service technique. L'annuaire permet de localiser et choisir la personnalité de service adéquate aux besoins de l'application et de son environnement d'exécution. Enfin le moniteur est le méta-composant représentant l'environnement d'exécution. Dans les sections suivantes, nous détaillerons le rôle et fonctionnement de chacun de ces composants.

Contrat

Pour modéliser les relations entre les éléments du niveau de base, on utilise la notion de contrat (voir figure 2.2). Dans un contrat, on représente l'offre du fournisseur et la de-

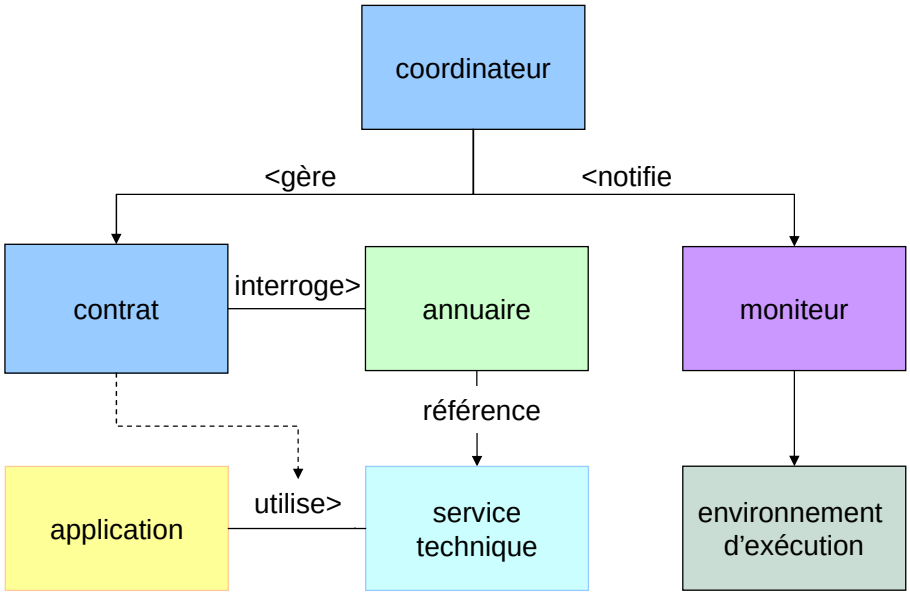

FIG. 2.5 – *Composants de gestion du système*

mande du client, puis on les négocie et fixe à valeur commune. Ici, on pourrait définir deux contrats : un premier contrat entre le service technique et l'environnement d'exécution, un second entre le composant applicatif et le service technique.

- Le premier se traduit par l'idée que si la qualité de service fournie par l'environnement d'exécution (profil fourni par l'environnement) correspond aux besoins du service technique (profil requis par le service technique) alors ce dernier garantit une certaine qualité de service (profil fourni par le service technique). On peut imaginer qu'un même service technique décrive plusieurs qualités de service en fonction des différents environnements qui pourraient l'accueillir. Le profil fourni par le service technique n'est rien d'autre que la description de la P1S définie dans la partie précédente.
- Le second contrat traduit l'idée que l'application attend du service technique (profil requis par l'application) une certaine qualité de service qu'il s'engage à rendre (profil fourni par le service technique).

Dans l'adaptation des services techniques telle que nous la définissons, la négociation est simplifiée : l'environnement et l'application sont regroupés en une seule partie qui passe un contrat avec un service technique. De plus, on fait une sélection des personnalités de service technique afin qu'elle réponde aux besoins. Pour chercher la personnalité de service technique adéquate, la requête est constituée du profil requis par l'application et du profil fourni par l'environnement. On compare cette requête à la description du service technique constituée des profils requis et fournis par le service technique. Lorsque la bonne personnalité est trouvé, le contrat est accepté.
Pour chaque composant applicatif qui nécessite une ou plusieurs personnalités de services techniques, le système crée un contrat. Il existe donc non pas un contrat mais autant de contrats que d'applications. Un contrat passé entre des personnalités de services tech-

niques et une application représente le fait que cette application bénéficie des services de ces personnalités (i.e. que cette application est composée avec les services techniques comme cela est défini dans la partie 1.3.1).
Or, un composant méta a deux attributions, celle d'offrir une abstraction au niveau de base et celle de pouvoir modifier ce niveau de base. La première attribution se traduit par le fait que le contrat regroupe le profil requis de l'application, une référence sur le composant applicatif et les différentes références sur les instances de services techniques utilisés par le composant applicatif ainsi que leur profil fourni. Comme le profil fourni par le service technique, le profil requis de l'application est une description faite par un opérateur humain, ici le développeur de l'application. Il peut prendre par exemple la forme d'un fichier xml qui suit la DTD décrite dans l'appendix A. Les besoins de l'application se décomposent en une liste de descriptions de services techniques. Pour chacun des services, on a des propriétés que celui-ci doit obligatoirement remplir et les préférences en personnalité de l'application qui sont optionnelles. De cette façon, le développeur de l'application peut améliorer sa description tout en n'étant pas trop restrictif. Dans les modèles tels que EJB ou CCM, on a la même approche, cependant elle est limitée à un cadre strict pour un ensemble de services techniques prédéfini .
La seconde attribution d'un composant méta étant de modifier le niveau de base, il peut corriger l'association entre application et services techniques pour la tenir à jour avec les personnalités les plus adaptées aux ressources fournies par l'environnement d'exécution et aux besoins de l'application. Pour cela, il est notifié des changements d'environnement, qui ont pour conséquence la mise à jour des associations application - services techniques. Pour effectuer cette mise à jour, le contrat consulte l'annuaire et lui demande s'il existe des services techniques mieux adaptés que ceux déjà utilisés. Si c'est le cas, il modifie les associations qu'il représente avec les nouveaux services. La requête faite sur l'annuaire est constituée du profil requis par l'application et du profil fourni par l'environnement d'exécution, ceci est détaillé dans la partie consacrée à l'annuaire.

Moniteur

Le moniteur fournit des informations sur l'environnement d'exécution et les communique au système afin qu'il choisisse de façon adéquat les services techniques. Ces informations sont des données statiques ou dynamiques, logicielles ou matérielles; elles sont décrites dans le chapitre 2. Par exemple, on peut connaître la puissance du CPU, donnée statique, ou encore son taux d'utilisation, donnée dynamique. Ces données matérielles sont complétées par des données logicielles telles que la version de la machine virtuelle Java, etc.
Pour fournir ces informations, le moniteur possède deux sources d'informations : un ensemble de capteurs qui fournissent des informations de façon automatique que l'administrateur peut compléter de façon ponctuelle par des données statiques. Ces informations sont assemblées dans un fichier descriptif, le profil fourni par l'environnement d'exécution puis communiquées au contrat pour choisir les services techniques.
On peut faire deux améliorations à ce moniteur : amélioration de la synthèse des données et filtrage de la diffusion des données.

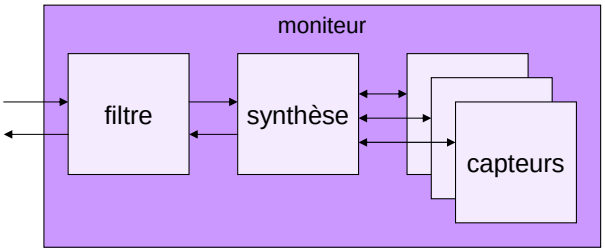

FIG. 2.6 – *Moniteur de l'environnement d'exécution*

Concernant la synthèse des données, si l'on possède un nombre de capteurs important, il peut être utile de faire une synthèse des valeurs relevées. En effet, pour pouvoir les comparer, il est nécessaire d'avoir une homogénéité entre les informations sur l'environnement d'exécution que l'on retrouve dans le profil fourni par l'environnement et le profil requis par le service technique. Si au contraire, on communiquait les données telles quelles, c'est à dire comme une compilation brute de données, le développeur de service technique devrait exprimer le profil requis par le service technique de la même façon; son travail serait exhaustif et donc difficile à mener. Par exemple, il est plus facile pour le développeur de service technique de préciser que son service est adapté à l'utilisation sur "un PDA connecté au réseau par une connexion sans fil" que de lui demander quelle est la puissance du CPU, la taille du disque dur, l'alimentation, la norme du réseau, etc dont son service a besoin.
Un second aspect de la synthèse consiste à "discrétiser" la fonction représentant les évolutions d'une variable. En effet, pour certaines données, on préférera exprimer leur valeur par son appartenance à un intervalle plutôt que son égalité à une valeur précise. Par exemple, la description "ce service technique est adapté à un environnement où le CPU est utilisé à 70%" n'est pas vraiment exploitable. Par contre la description "ce service technique est adapté lorsque le CPU est utilisé à moins de 70" ou encore "lorsque le CPU n'est pas trop utilise" l'est. Cette synthèse de la donnée relève du capteur, qui a des connaissances sur la donnée particulière.
La seconde amélioration consiste à filtrer les informations fournies à l'environnement d'exécution. En effet, la mesure des données sur l'environnement se fait de façon régulière, il n'est cependant pas intéressant de mettre à jour les services techniques à chaque mesure. On peut même aller plus loin dans la démarche et dire qu'il n'est pas nécessaire de faire une mise à jour à chaque modification de l'environnement mais plutôt à chaque modification significative de l'environnement. Une première opération de filtre a été exécutée au niveau du capteur, par exemple, lorsqu'il a analysé et arrondi les résultats. Une seconde opération de filtre peut être effectuée en ne diffusant l'information que si l'environnement a été modifié (ou modifié de façon sensible), en comparant la nouvelle valeur de l'environnement d'instant t a celle mesurée à l'instant t-1. Ainsi on peut éviter de faire des mises à jour des contrats pour rien.
On a donc un moniteur tel que sur la figure 2.6, constitué d'un module de filtre et d'un module de synthèse, chacune des informations étant collectée auprès d'un capteur qui fournit comme données le nom de l'information mesurée et sa valeur.

Notre moniteur nous fournit des informations synthétisées sur l'environnement d'exécution qui seront utilisées pour l'adaptation des services techniques. La notion de capteur permet une plus grande modularité. Le filtrage des données évite la surcharge du système et la synthèse de données permet de simplifier le travail du développeur de service technique.

Coordinateur

Le rôle du coordinateur est de maintenir la cohérence entre les différents composants de gestion. Son rôle principal est donc de gérer les contrats et de les lier au moniteur.
La gestion des contrats consiste tout d'abord en leur création. Le coordinateur est donc une usine à contrats, il garde une référence sur chacun des contrats crées afin de les modifier lors des changements d'environnement. Il est donc le point d'entrée du système pour l'administrateur du système qui veut enregistrer une application ou une personnalité de service technique (la création d'un contrat est détaillée dans la section 2.2.2).
Ensuite lorsque le moniteur perçoit une modification de l'environnement d'exécution, il prévient le coordinateur qui a pour rôle de diffuser l'information (le détail de fonctionnement du moniteur est décrit dans la partie suivante). Trois principales manières d'échanger les informations entre moniteur et contrats sont alors envisageables selon que l'on choisisse le modèle pull, le push ou une hybridation des deux.

Modèle pull Selon le modèle *pull* (ou tirer en français), c'est le contrat, ayant besoin d'une mise à jour du profil requis par l'environnement, qui en fait la demande (voir figure 2.7). On a alors deux cas de figure possibles :

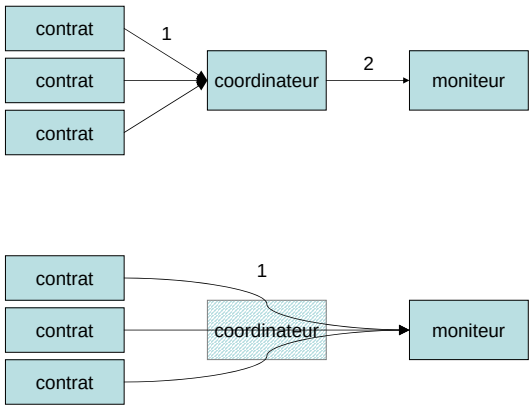

FIG. 2.7 – *Diffusion du profil fourni par l'environnement d'exécution : modèle pull*

- le contrat s'adresse au moniteur par l'intermédiaire du coordinateur, en effet c'est le moniteur qui maintient la cohérence du système et en connaît l'état. Si l'on veut modifier l'emplacement du moniteur, seul le coordinateur a besoin d'en être averti ce qui facilite la gestion du système, cependant on surcharge le coordinateur d'appels.
- le contrat demande directement au moniteur dans la mesure où ce dernier n'a besoin d'effectuer aucun traitement particulier. Ainsi on a un gain de temps pour la dif-

fusion du profil fourni par l'environnement. Par contre, le maintien de la cohérence du système est plus onéreux.

Le problème posé par ce modèle est que les contrats sont obligés de prendre de façon régulière l'initiative de demander la mise à jour, ce qui surcharge leur travail même si l'environnement d'exécution n'a pas été modifié depuis leur dernière demande.

Modèle push Selon le modèle *push* (ou pousser en français), l'initiative de la diffusion revient au moniteur (voire figure 2.8). Plusieurs variantes de ce modèle sont possibles dans notre système.

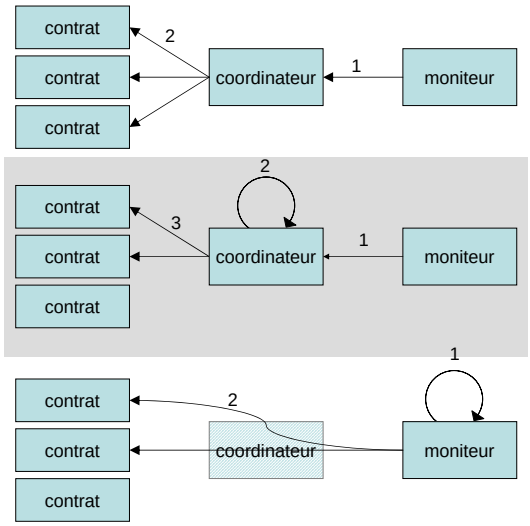

FIG. 2.8 – *Diffusion du profil fourni par l'environnement d'exécution : modèle push*

- le moniteur envoie l'information au coordinateur qui la diffuse à tous les contrats.
- si l'on considère que le moniteur regroupe les informations concernant plusieurs capteurs, une politique évoluée de diffusion des informations entre moniteur et contrats consiste à trier les informations récoltées et ne les diffuser qu'aux contrats qui seraient intéressés par ces informations. Alors on peut effectuer ce tri au niveau du coordinateur car il a une connaissance de tout le système.
- une variante consiste à ce que le tri soit directement effectué par le moniteur, ainsi le coordinateur n'a pas besoin d'appartenir à la chaîne de diffusion. Cependant, le coordinateur doit alors connaître les informations dont les contrats ont besoin et le moniteur perd de son indépendance.

Ce modèle semble tout à fait adapté au problème posé. En effet, il permet de ne pas surcharger les contrats et de multiplier les demandes et les réponses inutiles. On retiendra la seconde variante. En effet, le travail de tri y est effectué par le coordinateur. Or le modèle composant préconise la séparation des tâches pour faciliter la réutilisation des modules logiciels tels que le moniteur.

Modèle hybride Dans la mesure où l'on a deux communications qui s'effectuent : l'une entre le coordinateur et les contrats et l'autre entre le coordinateur et le moniteur, on peut utiliser une solution *hybride* où l'une des communications se fait selon le modèle push et l'autre selon le modèle pull (voir figure 2.9).

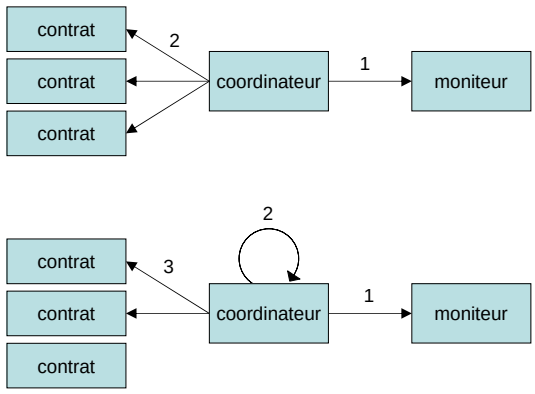

FIG. 2.9 – *Diffusion du profil fourni par l'environnement d'exécution : modèle hybride*

- dans cette configuration c'est le coordinateur qui demande les informations sur l'environnement. Il ne sait pas si l'environnement a été modifié depuis sa dernière demande, il doit donc le faire régulièrement. Ensuite il diffuse les informations aux contrats.
- une amélioration de cette solution consiste à effectuer un tri au niveau du coordinateur et à ne diffuser l'information qu'aux contrats intéressés grâce à un système de souscription.

Ce dernier modèle semble moins adapté que le modèle push pur, dans la mesure où il nécessite que le coordinateur génère parfois des requêtes inutiles (lorsque l'environnement n'a pas été modifié depuis la dernière requête). Cependant, il est mieux adapté que le modèle pull dans la mesure où il y a moins de requêtes générées.

Le coordinateur a donc un double rôle, celui d'usine à contrats et celui de diffuser et de sélectionner les informations à diffuser entre le moniteur et les contrats. En effet, dans la mesure où l'on se rapproche d'un modèle à événements, le modèle d'échange d'informations le plus intéressant est le modèle push avec le tri effectué au niveau du coordinateur.

Annuaire de services techniques

Notre annuaire de service technique permet à la plate-forme la mise à disposition et le retrait dynamique de services techniques ainsi que la gestion de multiples personnalités d'un même service ([GRI 05],[GRI 04]). En effet, il permet de localiser et choisir les personnalités de services techniques adaptées en tenant compte de l'environnement d'exécution et des besoins de l'application. Afin de localiser les personnalités de services techniques, il fournit un service de courtage basé sur un annuaire générique de composants. La fonctionnalité de décision de la meilleure personnalité est basée sur le concept

de courtage sémantique [TER 00].
Il ne faut pas le confondre avec l'annuaire de composants applicatifs, fréquemment utilisé dans les plate-forme à composants avec l'annuaire de services techniques. L'annuaire de composants applicatif est à considérer comme n'importe quel autre service technique, il fait partie des services technique enregistré dans l'annuaire des services techniques.

Annuaire générique de composants Afin de localiser les personnalités de services techniques, nous nous sommes inspirés des services d'annuaire proposés pour CORBA [OMG 00]. Ceux-ci permettent de localiser de façon transparente les objets CORBA (ou les composants dans CCM) grâce à un nom symbolique qui se substitue à l'IOR (pour Interoperable Object Reference) de l'objet, pas évident à manipuler, ou grâce à une description. Ces services peuvent être comparés aux pages jaunes et pages blanches de l'annuaire téléphonique, on les nomme respectivement service de nommage et de courtage.
Notons que dans CCM, l'annuaire n'est pas utilisé pour retrouver les services techniques, dans la mesure où ceux-ci ne sont pas des composants mais des objets notoires, c'est-à-dire des objets dont on a fixé l'emplacement au moment du déploiement. Dans CCM, l'emplacement d'un service technique ne peut donc pas être modifié pendant l'exécution et le bus logiciel n'est pas conçu pour permettre d'avoir plusieurs services techniques implantant la même fonctionnalité (persistance, sécurité, etc).
Dans notre annuaire de personnalités de services techniques, c'est plus précisément le service de courtage qui est utilisé pour stocker les références sur les services techniques sous forme de composants. Ainsi le fournisseur d'un composant l'enregistre dans l'annuaire en le caractérisant par une liste de propriétés et leur valeur associée. C'est l'opération d'export. Ensuite, le client indique ses besoins, il récupère les références des objets répondant à sa requête. C'est l'opération d'import. Il peut par la suite utiliser directement les services d'un de ces objets.
Afin de compléter les concepts de l'annuaire d'objet CORBA, on intègre les notions de type de composants et de patron (template) de composants. En effet, dans l'annuaire CORBA, on ne retrouve que les instances de composants. En enregistrant aussi les types, on a une connaissance sur les interfaces du composants et l'on évite la redondance en ne stockant cette information qu'un fois pour tous les composants fils de ce type. Enregistrer les patrons permet au système de n'instancier les composants que lorsqu'on en a besoin, ce qui permet un gain en terme de place mémoire vive. La structure de stockage de l'annuaire prend la forme d'un arbre où les instances sont les fils des patrons et les patrons les fils des types.
Grâce à l'écriture des services techniques sous forme de composants, on peut utiliser le service de courtage ainsi mis en œuvre. Cependant, cet annuaire ne permet pas de prendre en compte l'environnement d'exécution des services techniques et les préférences des développeurs d'applications.

Courtage sémantique Afin de prendre en compte les facteurs d'environnement d'exécution et de préférence de l'application, le service de courtage générique de composants, décrit ci-dessus, est lui même intégré dans un service de courtage sémantique pour créer notre annuaire de personnalités de services techniques. En effet, basé sur la notion de

courtage classique, le courtage sémantique permet non seulement de fournir un service qui correspond exactement aux contraintes exprimées par l'utilisateur, mais il permet à l'émetteur de la requête et au système d'exprimer aussi des préférences [TER 00]. A la différence d'une requête de courtage classique, les préférences ne sont pas forcément satisfaites. Elles sont utilisées pour départager les solutions répondant à la requête classique. Une requête satisfaisant plus de préférences ayant une plus grande chance d'être choisie. On peut ajouter à ce premier critère (nombre de préférences satisfaites), un poids à chaque préférence afin d'exprimer le fait que certaines préférences sont plus importantes que d'autres. Ainsi la solution choisie est celle dont la somme des poids des contraintes satisfaites est la plus grande.
Pour regrouper ces préférences, le courtage sémantique se repose sur une base de connaissances. La base de connaissance de notre annuaire est constitué de deux types de préférences : celles exprimées par le développeur de l'application, elles sont directement exprimées dans sa requête et celles exprimées par l'administrateur du système, qui sont persistantes.
En permettant l'expression des préférences de l'administrateur et du développeur d'applications, on évite qu'une requête trop précise ne soit pas du tout satisfaite. En effet, on permet au développeur de l'application de faire une requête minimale, ce qui lui assurera une réponse positive à sa demande. Puis la réponse fournie est affinée grâce aux préférences.

Fonctionnement de l'annuaire de services techniques Notre annuaire de services techniques est donc un annuaire de courtage sémantique qui intègre un annuaire générique de composants décrit dans la section 2.2.1. Il est constitué de trois principaux modules :

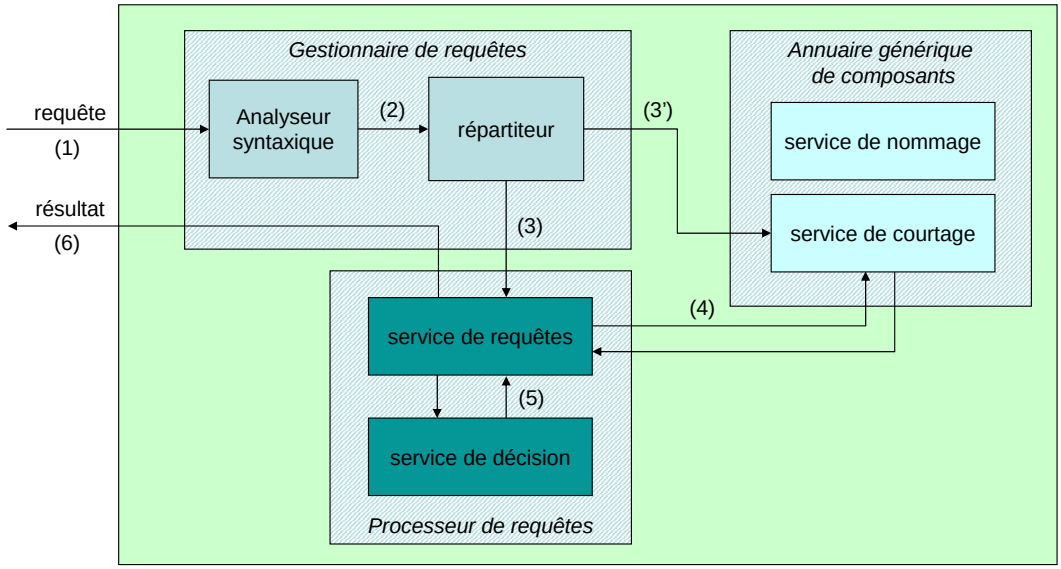

FIG. 2.10 – *Annuaire de personnalités de services techniques*

– un *gestionnaire de requête* qui a pour but de faire une première analyse qui permet de déterminer le type de requête : requête d'export, faite par l'administrateur lors

de l'ajout d'un service technique au système, ou une requête d'import, faite par le contrat pour localiser une personnalité. Ce gestionnaire de requête se décompose lui même en un analyseur syntaxique et un répartiteur.

- un *processeur de requête* qui traite uniquement les requêtes d'import et se décompose en un service de requête et un service de décision qui a pour rôle d'améliorer la qualité de réponse grâce à l'analyse sémantique.
- un *annuaire générique de composants* qui contient le service de nommage et de courtage.

L'administrateur peut à tout moment (avant le déploiement des applications et même après) exporter (ou enregistrer) une personnalité de service technique dans l'annuaire en fournissant des renseignements sur l'environnement idéal pour lequel elle est adaptée ainsi que le type d'application à laquelle elle est destinée. Par exemple, la personnalité "plate" du service transactionnel est mieux adaptée à une application de durée courte. La personnalité "asynchrone" d'un service de communication est mieux adaptée à un contexte d'exécution où le réseau est sujet aux pannes.

La requête d'export d'une personnalité de service technique reprend la description de la P1S, elle reprend le profil requis et le profil fourni par le service technique. Le traitement d'une requête d'export (1) est pris en charge par l'analyseur syntaxique qui détermine s'il a affaire à une requête d'import ou d'export, puis il détermine s'il s'agit d'une requête d'export d'un type de composant, d'un patron ou d'une instance et la confie au répartiteur (2). Celui-ci délègue la requête au service de courtage de l'annuaire générique de composants (3'). Une référence sur le service technique avec sa description est alors stockée dans l'annuaire et peut être utilisée par les composants applicatifs.

La requête d'import (ou de localisation) permet à un contrat de trouver un service technique adéquat. Elle est constituée du profil requis par l'application en terme de service technique et du profil fourni par l'environnement d'exécution. Afin de permettre au développeur d'applications d'exprimer ses besoins mais aussi ses préférences s'il en a, le profil requis par l'application s'exprime sous forme de contraintes et de préférences. Les contraintes correspondent aux besoins minimaux du composant applicatif pour s'exécuter correctement. Les préférences permettent au développeur du composant applicatif d'exprimer des recommandations.

Comme pour l'export, l'import est pris en charge par le gestionnaire de requête (1) et (2), celui-ci redirige la requête vers le gestionnaire de requête (3). Le service de requête transforme la requête pour qu'elle puisse correspondre à une requête de type courtage classique et fait les demandes nécessaires auprès du service de courtage (4). Ce dernier lui renvoie une liste de résultats qui est confiée au service de décision. Ce service classe les résultats selon les préférences exprimées par le développeur de l'application (5). Enfin cette liste des références est renvoyée au contrat qui va pouvoir mettre à jour la liaison entre le composant applicatif et ses services techniques (6).

En conclusion, un service de nommage tel que celui proposé dans CORBA n'aurait pas été suffisant pour la gestion de multiples personnalités fournies par plusieurs développeurs de services techniques. En effet, il aurait fallu que l'on connaisse à l'avance les noms symboliques des services techniques et qu'ils aient été uniques. Grâce à l'utilisation de l'annuaire générique de composants, adaptation du concept d'annuaire d'objet CORBA

aux composants, notre annuaire permet de retrouver la personnalité de service adaptée à la fois aux besoins de l'application et à son environnement d'exécution, sans connaissance a priori sur les services et sur les besoins en termes de services techniques. L'ajout d'un aspect sémantique à notre annuaire lui confère une plus grande souplesse. En effet, il ne va pas simplement répondre à une requête stricte, il va pouvoir répondre à cette requête et améliorer cette réponse en répondant à des préférences exprimées par l'administrateur et le développeur d'applications. Cela permet d'éviter de ne trouver aucune réponse à la requête.

2.2.2 Interactions des composants du niveau méta

Dans la partie précédente, le fonctionnement de chacun des composants de gestion a été détaillé. Cette section résume les interactions entre ces composants tout au long de la boucle d'adaptation que nous avions présenté de façon générale dans l'introduction de cette partie. Pour l'adaptabilité statique, la boucle d'adaptation s'effectue une seule fois, au moment du déploiement des composants applicatifs. Pour l'adaptabilité dynamique, elle s'effectue au déploiement puis tout au long du cycle de vie des composants applicatifs.

Déploiement

Le déploiement d'un composant applicatif, du point de vue du système d'adaptation, consiste en la création d'un contrat lui étant associé. Avant le déploiement de l'appli-

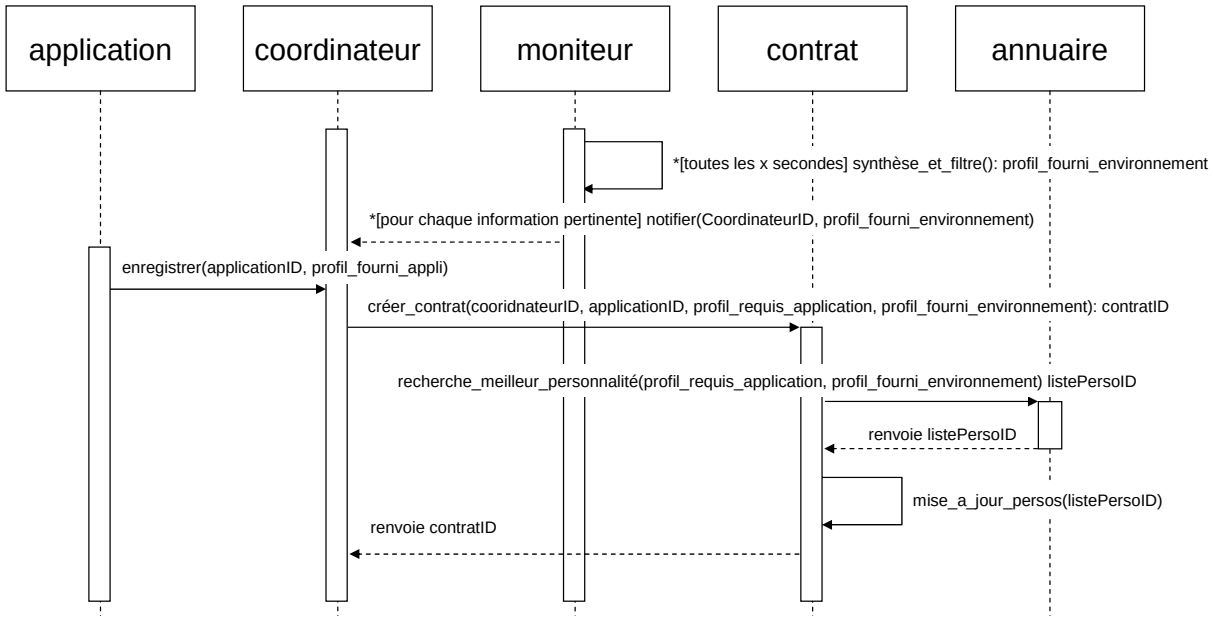

FIG. 2.11 – *Création d'un contrat*

cation, les personnalités sont enregistrées dans l'annuaire par l'administrateur avec leur description constitué des profils requis et fournis de la personnalité. Le moniteur fournit

des informations filtrés au coordinateur.
La création du contrat commence par l'enregistrement du composant applicatif auprès du coordinateur en fournissant ses préférences dans le profil requis de l'application (voir figure 2.11). Le coordinateur crée alors un contrat, c'est-à-dire qu'il va chercher les personnalités adaptées à la fois à l'environnement d'exécution et aux besoins et préférences de l'application. Le contrat, grâce aux profils requis de l'application et au profil fourni de l'environnement d'execution demande à l'annuaire de lui fournir des services techniques adaptés. Puis il crée les liaisons décrites dans la partie 1.4.2 de "un nouveau modèle de réalisation des services techniques".

Execution de la boucle d'adaptation

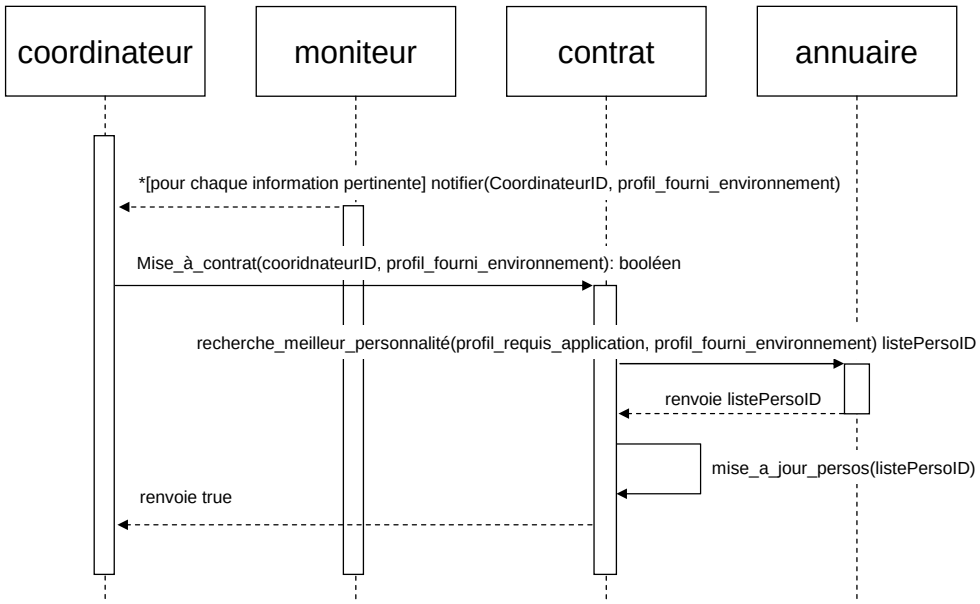

FIG. 2.12 – *Adaptation du système*

Pendant l'exécution, l'environnement peut évoluer : changement de bande passante, de l'utilisation du CPU, etc. Ainsi les contrats doivent être renégociés. Le système initie cette adaptation grâce aux informations sur l'environnement qui lui sont données par les moniteurs. Le coordinateur prévient les contrats qui sont concernés qu'il doivent évaluer la nécessité de leur propre mise à jour et chercher à nouveau les personnalités les plus adaptées grâce à l'annuaire.
Lorsque le composant applicatif termine son cycle de vie, son contrat expire. Les personnalités et les moniteurs ont normalement la même durée de vie que le coordinateur. Néanmoins, le système doit gérer la fin de vie accidentelle d'une personnalité afin d'éviter toute panne. Cela signifie que les contrats dans lesquels la personnalité était engagée doivent être mis à jour.

2.2.3 Conclusion sur l'adaptabilité dynamique

Dans cette section, nous vous avons présenté des mécanismes pour l'adaptation dynamique des services techniques. Ils reposent sur un ensemble de composants de gestion. Un composant méta "contrat" donne une représentation du niveau de base et agit dessus afin de maintenir la cohérence entre le niveau de base et sa représentation au niveau méta. Les autres composants agissent sur ce contrat afin d'associer aux applications les services techniques adéquats et offrir une qualité de service globale du système meilleure.

Meilleure vision du système grâce à la réflexivité

Nous avons défini chaque contrat comme représentant les interactions entre un composant applicatif unique et des services techniques. Cette association peut se faire à tous les niveaux de granularité. Le choix de la granularité est laissé au développeur qui devra trouver un juste milieu entre associer les services techniques avec l'application dans son entier ou avec chacun des composants applicatifs élémentaires.
Le contrat donne une abstraction des interactions entre composant applicatif et services techniques. Il permet de représenter et modifier ces interactions sans pour autant que les développeurs de composants applicatifs et techniques n'aient à gérer ces adaptations.

Automatisation de l'adaptation grâce aux composants de gestion pour l'amélioration de la qualité de service

D'autres composants complètent ce niveau : le coordinateur, l'annuaire et le moniteur. Ces composants sont conçus pour faciliter l'adaptation automatique du contrat. Ils interagissent de façon claire et simple grâce à leur conception par composants. Leur fonctionnement est indépendant de l'exécution des composants et des services techniques, mis à part le point d'accroche entre niveau de base et niveau méta au niveau du contrat.

- Le coordinateur, usine à contrats, donne une cohérence au système en connaissant les "emplacements" de tous les autres composants. Ainsi il est le point d'entrée du système pour l'administrateur.
  Dans notre solution, on suppose une certaine unité d'emplacement des composants : on ne représente qu'un seul environnement d'exécution qui est celui de tous les services techniques dont les applications peuvent bénéficier. Ceci implique que tous les composants constituant un service technique doivent être sur la même machine et que le service technique doit être sur la même machine que le moniteur. Cependant on n'empêche pas que le coordinateur, l'annuaire et l'application soient sur d'autres machines.
  D'autre part, des composants applicatifs étant distribués peuvent coopérer, bénéficiant des services du coordinateur auprès duquel elles sont enregistrées. Cependant nous n'avons pas défini de coopération entre les différents coordinateurs. On pourrait imaginer qu'un système de composants de gestion ne trouvant pas de service technique adapté émette une requête auprès des autres systèmes de composants de gestion qui l'entourent.

– L'annuaire de service a été enrichi par le concept de courtage sémantique. Ainsi, il apporte une solution flexible et efficace pour la localisation et le choix des P1S en permettant l'expression non seulement de contraintes sur le choix des services techniques mais aussi des préférences.
  La notion de L2S manque dans l'annuaire. Afin de la représenter, il faudrait modifier la représentation sous forme d'arbre qui ne permet de représenter que les types, les patrons et les instances. Un L2S est un ensemble de P1S, les P1Sétant représenté au niveau des instances, un L2S devrait être représenté comme un ensemble d'instances.
  D'autre part, l'annuaire permet grâce à une base de connaissances de choisir les services techniques. On aimerait pouvoir enrichir automatiquement cette base de connaissances grâce à un module d'apprentissage de règles. Par exemple, si un certain environnement est très souvent associé aux mêmes personnalités de services, alors on pourrait l'ajouter comme une règle de décision ce qui faciliterait le travail de décision. Par exemple, Jim Dowling dans [DOW 04] présente un système de composants adaptés grâce à des règles de type "événement,condition,action". Ce système est capable d'apprendre grâce à l'experience des systèmes voisins ("collaborative feedback" en anglais).
– Enfin le moniteur, nécessaire au choix de personnalités adaptées à leur environnement, donne des informations synthétisées sur l'environnement d'exécution.
  L'implantation du moniteur pose deux types de problèmes : comment récupérer les données de qualité de service à travers des intergiciels très différents, comment synthétiser ces informations. L'aspect récupération des données sera abordé dans la partie prototype où nous montrerons deux mises en œvre de ce moniteur. Concernant la synthétisation des données, on pourrait utiliser des règles du type " si on a telles ou telles caractéristiques d'environnements alors on se trouve dans un environnement de tel type ". Cela signifie qu'on a besoin d'une classification et d'une hiérarchisation des descriptions d'environnement, et que l'on a besoin d'établir une base de connaissances regroupant ces règles. Ceci passera sans doute par un effort de standardisation.

Grâce aux composants de gestion, les composants applicatifs peuvent bénéficier de la personnalité d'un service technique la plus adaptée à leur environnement d'exécution dont on a une idée concrète grâce au moniteur et à leurs besoins exprimés de façon complète grâce aux contraintes et aux préférences. Ainsi la qualité de service globale, fournie par le système, est améliorée.

3

Mise en oeuvre et validation

Afin de montrer la faisabilité des propositions faites dans la partie "propositions", nous avons développé un prototype dont la majeure partie est conçue à base de composants Julia 1.0.6. Julia est l'implantation standard du modèle à composants Fractal. En effet, si le modèle répondait à nos attentes du point de vue théorique, sa mise en oeuvre, Julia, satisfait à nos contraintes techniques : elle facilite l'intégration des différentes parties (code ancien ou nouveau, modèle hiérarchique, etc), les composants sont écrits en Java, langage largement utilisé dans le domaine des applications nouvelles, les conteneurs sont ouverts.
Comme pour les propositions, on peut distinguer plusieurs parties de ce prototype : la conception et la mise en œuvre des services techniques, leur assemblage avec les composants applicatifs à travers l'utilisation d'un conteneur ouvert et les composants de gestion. Notons que la validation de la conception et de la mise en œuvre des services techniques est faite au travers de l'exemple d'un service transactionnel multi-personnalités.

3.1 Validation du modèle de réalisation à travers un service transactionnel multi-personnalités

Cette section est consacré à la mise en pratique des propositions faites dans la section "nouveau modèle pour la réalisation des services techniques", à travers la conception et le développement d'un service technique de transaction ainsi que la configuration d'un conteneur ouvert pour l'assemblage du composant applicatif et des services techniques.

Ce travail a été mené en collaboration avec Sergiy Nemchenko ([NEM 04], [HéR 04c], [BEN 02]) dont la thèse est consacrée à la définition de mécanismes de gestion des transactions ouvertes imbriquées dans les plate-formes à composants. Ses propositions ont été validées grâce à un prototype dont la mise en œuvre a fait l'objet de travaux communs. Le service transactionnel implante deux modèles transactionnels : les transactions plates et les transactions ouvertes imbriquées.
Dans un premier temps, nous reviendrons sur la définition du service transactionnel puis nous nous intéresserons à l'utilité des modèles décrits dans la littérature. Ensuite nous détaillerons la mise en œuvre à proprement parler. Elle suit la projection proposée dans

la figure 1.1 de la partie I chapitre 1 : (1) la description avant implantation des deux personnalités du service, (2) l'identification des composants nécessaires correspondants aux mécanismes de la transaction notamment celui de gestion de la compensation, (3) l'assemblage de ces composants, (4) la mise en œuvre de l'assemblage entre composants applicatifs et services techniques. Une cinquième partie reviendra sur les personnalités décrites en (1) pour les compléter avec les valeurs obtenues lors des tests faits sur le service prototypé. Enfin nous ferons un bilan de l'expérience menée.

3.1.1 Trois modèles de transaction

Le service transactionnel est l'un des services techniques les plus importants. Il a pour but d'améliorer la fiabilité d'exécution des applications en maintenant la cohérence des bases de données. Ainsi il améliore la tolérance aux pannes du système. Il est d'ailleurs présent dans les différentes implantations des modèles EJB [MON 01] et CCM [CCM02] et l'un des premiers services techniques implantés pour le modèle Fractal [NEM 04].
On définit une transaction comme étant un ensemble indivisible d'opérations qui débute par une opération démarrage et se termine soit par une opération de validation en cas de réussite, soit par une opération d'annulation de la transaction en cas d'échec d'une des opérations. De plus la transaction respecte les propriétés ACID ([ORF 99], [BES 97]) :

- Atomicité : Une transaction est indivisible, elle est soit exécutée complètement soit pas du tout.
- Cohérence : des données qui sont cohérentes avant l'exécution d'une transaction, le seront aussi après. On notera qu'il est possible qu'au cours de cette transaction, les données passent par des états incohérents.
- Isolation : les modifications qui sont faites par une transaction sont invisibles aux transactions concurrentes.
- Durabilité : si une transaction est validée, le résultat de ses actions ne sera pas remis en cause, même en cas de panne.

Le service transactionnel gère la création et la validation des transactions. Il garantit la validité d'exécution des transactions et le respect des propriétés ACID.
L'utilisation d'une transaction est nécessaire lorsque deux parties de code doivent accéder aux mêmes données. Sinon, on risque de perdre la cohérence de la base de données. Prenons comme exemple une application de virement bancaire. Elle consiste en deux opérations : débit d'une somme S d'un compte A, crédit de S sur le compte B. Si jamais la première opération était effectuée et pas la seconde, la base de données serait dans un état incohérent : la somme S n'apparaîtrait plus. Par contre, si ces opérations sont effectuées dans une transaction, la première opération ne peut pas être effectuée sans la seconde. S'il y a échec de la seconde opération la première est défaite : le débit est annulé.
Il existe de nombreux modèles permettant d'implanter les transactions : transaction plate, CNT (Closed Nested Transaction [MOS 81]), ONT (Open Nested Transaction [GRA 81]), workflow [BES 97], etc. Dans les modèles à composants usuels, seul le modèle des transactions plates est implanté. Cependant, ce modèle de transaction ne s'avère pas toujours être le plus approprié. Par exemple, les modèles de transaction CNT et ONT sont mieux adaptés aux nouvelles applications distribuées où les transactions concernent plusieurs

base de données distribuées sur une longue période de temps.

Transactions plates

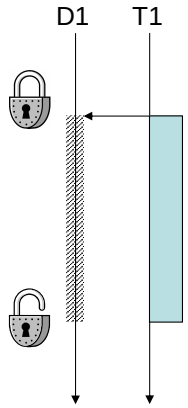

FIG. 3.1 – *Exemple de transaction plate.*

Les transactions plates sont les transactions les plus communément utilisées et les plus simples (voir figure 3.1). Elles respectent chacunes des propriétés ACID et reposent sur un mécanisme de validation comme le mécanisme de la validation à deux phases (2PC) [GRA 81]. Les transactions plates ont pour inconvénient de bloquer l'accès aux données participant à la transaction pendant tout le temps de la transaction. Sur la figure 3.1, on voit que la donnée D1 est bloquée pendant tout le temps de la transaction T1. Ce modèle empêche le parallélisme entre les transactions. Il est donc bien adapté aux applications courtes, mais lorsqu'une application courte veut accéder à des données qui sont déjà utilisées par une application longue, alors l'application courte est pénalisée car elle ne peut accéder à ces données. De plus, la validation d'une transaction plate impliquant plusieurs bases de données distantes peut être très coûteuse. Elle a aussi une grande probabilité d'aboutir à un abandon total de la transaction.
Afin de résoudre ces problèmes, d'autres modèles de transactions on été proposés. Ils permettent d'optimiser l'exécution des applications en relâchant certaines des contraintes ACID.

Transactions emboîtées fermées

Le modèle des transactions emboîtées fermées a pour but de pallier les inconvénients des transactions plates en introduisant la notion de sous-transaction. [MOS 81] qui est un sous-ensemble de la transaction (voir figure 3.2). On les appelle respectivement transaction mère et transactions filles. Un transaction fille peut elle-même être la mère d'une ou plusieurs sous-transactions. On voit sur la figure que les transactions T1 et T2 ont deux filles respectivelement T1.1, T1.2 et T2.1, T2.2.
Les règles de démarrage et d'achèvement des transactions CNT sont :

- la validation d'une sous-transaction devient permanente une fois que celle-ci est validée localement et que tous ses ancêtres sont validés.

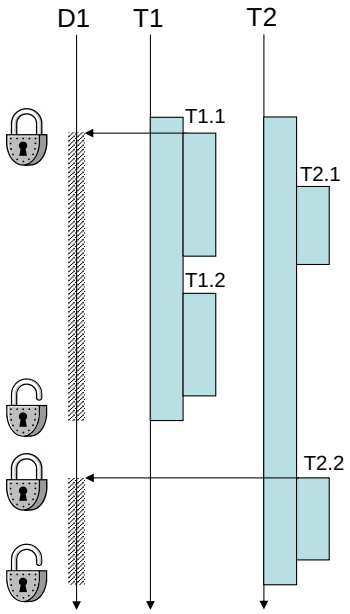

FIG. 3.2 – *Exemple de transactions CNT.*

– toutes les sous-transactions sont abandonnées en cas d'abandon de la transaction-mère.

Sur la figure, on voit que la sous-transaction T2.2 ne peut accéder à la donnée D1 qu'après que la mère transaction de T1.1 (e.g T1) soit totalement validée.
Ce modèle de transaction autorise un parallélisme limité. Les ressources utilisées par les transactions ne sont verrouillées qu'à partir du moment où elles sont utilisées pour la première fois et ceci jusqu'à la validation de la transaction mère. Ainsi lorsque deux transactions doivent être exécutées parallèlement, elles peuvent l'être tant que deux de leurs sous-transactions ne veulent pas accéder à la même donnée simultanément. De plus, ce modèle permet une plus grande souplesse dans la reprise sur panne car si une sous-transaction est abandonnée, il n'est pas nécessaire de relancer la transaction mère.
Bien qu'amélioré par rapport au modèle des transactions plates, le parallélisme des transactions CNT reste limité. Ce modèle n'est donc pas tout a fait adapté aux transactions longues entre des bases de données distantes.

Transactions emboîtées ouvertes

Proposé par Jim Gray en 1981 [GRA 81], ce modèle de transactions imbriquées relâche la contrainte d'Isolation. Lors de la validation d'une sous-transaction, les verrous mis sur les données utilisées par cette sous-transactions sont relâchés et d'autres applications peuvent alors accéder aux nouvelles valeurs de ces données sans attendre la validation de la transaction mère. Dans la figure 3.3, la sous-transaction T2.2 peut accéder à la donnée D1 dès la validation de la sous-transaction T.1.1 sans attendre la validation de sa transaction mère T1.
Le modèle des ONT est donc particulièrement avantageux dans des applications de type

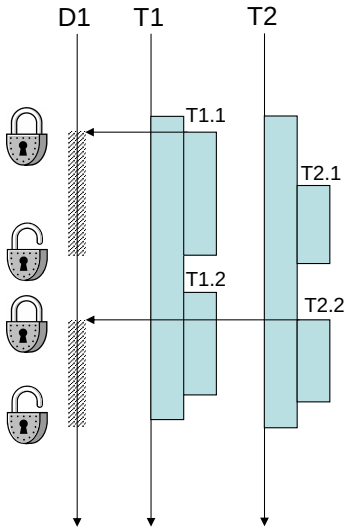

FIG. 3.3 – *Exemple de transactions ONT.*

B2B où les transactions peuvent être particulièrement longues car il augmente le parallélisme.
Toutefois relâcher la contrainte d'Isolation amène le besoin d'un mécanisme de compensation. En effet, si une transaction mère est abandonnée toutes ses transactions filles doivent être abandonnées. Ainsi il est nécessaire de compenser l'action de chacun de ses sous-transactions (voir figure 3.4). Si l'on reprend l'exemple d'opération bancaire, on peut l'exécuter dans une transaction T1. Cette transaction T1 se décompose en la transaction T1.1 qui exécute le débit et la transaction T1.2 qui exécute le crédit. Si la transaction T1 est annulée après la validation de la sous-transaction T1.1, la base de donnée est dans un état incohérent. Il faut alors exécuter une opération de compensation. Notons qu'entre la validation de T1.1 et l'annulation de T1, d'autres applications ont pu accéder à la base de données. C'est pourquoi l'opération de compensation n'est pas forcement triviale à définir. Ici une compensation simple consiste à recréditer le compte bancaire de la même somme. Une compensation plus élaborée consiste à recréditer la somme initiale ainsi que les frais annexes qui pourraient avoir été facturés si le compte bancaire avait été débiteur.
Malgré le fait que ce modèle ait été spécifié en 1981, jusqu'à présent il ne semble pas avoir été implanté entièrement dans les plates-formes à composants. En effet, ce modèle demande un travail supplémentaire au développeur de l'application car celui-ci doit fournir des mécanismes de compensation pour défaire, en cas d'échec de la transaction, les opérations effectuées par la transaction.

3.1.2 Conception du service transactionnel multi-personnalités

Le service transactionnel proposé implante les modèles des transactions plates et des ONT sous la forme d'un assemblage de composants.

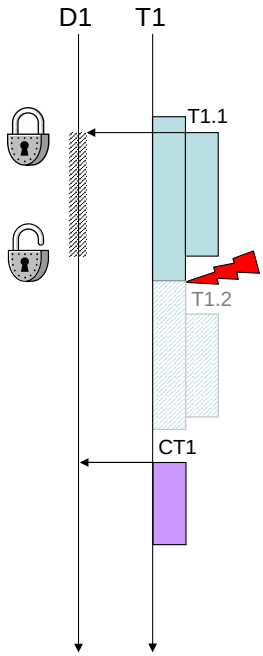

FIG. 3.4 – *Exemple d'une transaction ONT compensée.*

(1) Description des personnalités

Afin de pouvoir utiliser le service transactionnel dans notre système, des descriptions de ses personnalités sont élaborées selon les spécifications faites dans la partie II chapitre 1 section 1.1.3. La première étape de l'élaboration de ces déscriptions consiste en une identification des facteurs permettant de décrire la personnalité. Ensuite, on complète cette description avec les valeurs associées à ces facteurs. Certaines des valeurs sont déjà identifiables grâce aux connaissances du développeur de services techniques car elles ne dépendent pas de l'implantation de la personnalité (ex : environnement pour lequel la personnalité va être conçu). D'autres valeurs seront complétées après implantation (voir section 3.1.2) grâce aux résultats de tests empiriques.
Après cette première phase, nous avons obtenu les P1S présentées dans les figures 3.5 et 3.6. La P1S de la figure 3.5 signifie que l'implantation de service technique qui lui est associée fournit un service transactionnel dont le modèle est celui des transactions plates. L'environnement d'exécution pour lequel il a été conçu sera caractérisé par le type d'application exécutées, la fréquence de son processeur et son pourcentage d'utilisation. Les performances de la personnalité seront caractérisées par le nombre de transactions par minute. Ces caractéristiques sont représentatives des caractéristiques utilisées pour faire le choix entre différentes personnalités d'un service transactionnel (voir exemple de choix dans la partie 3.1.2). La P1S de la figure 3.6 décrit un service transactionnel ONT qui est plus particulièrement adapté aux Webservices.

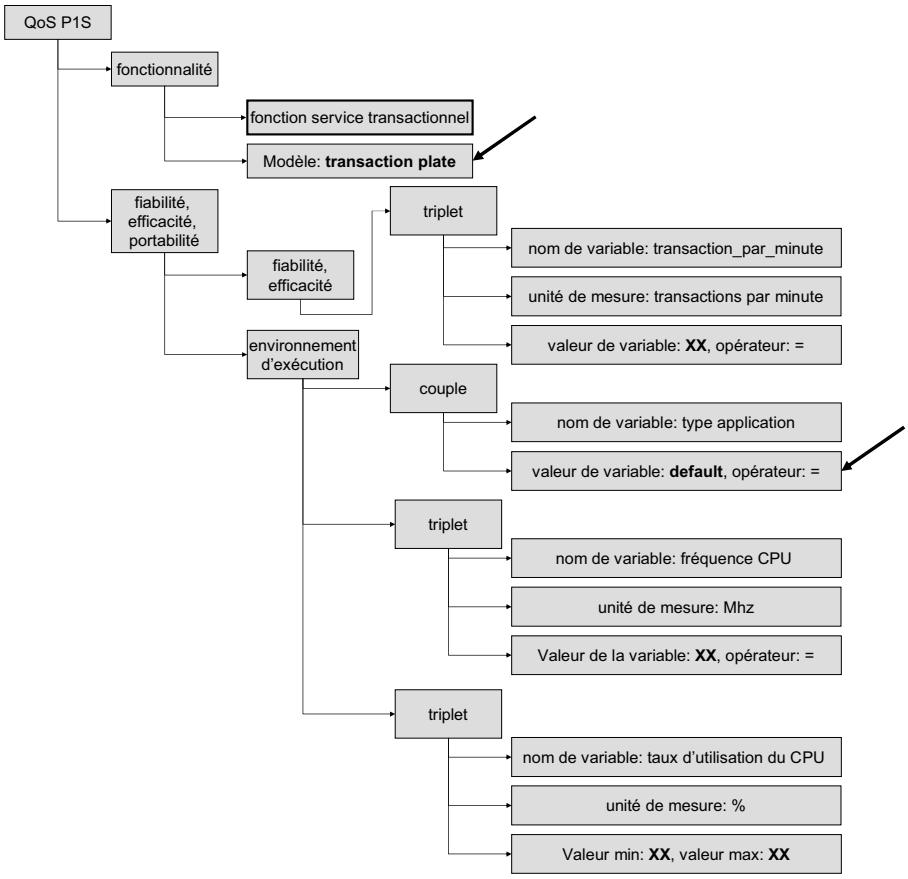

FIG. 3.5 – *Extrait de la P1S transaction plate partiellement complétée.*

(2) Identification des composants

Pendant cette phase de travail, nous distinguons les différents mécanismes transactionnels, puis nous identifions les mécanismes redondants entre les différentes personnalités. Chacun des mécanismes est assimilé à un composant (qui peut lui même être composé de sous composants) et les mécanismes utiles aux différentes personnalités apparaissent sous la forme de composants partagés.

Implantation du modèle des transactions plates Afin d'implanter les transactions plates, les travaux de S. Nemchenko [NEM 04] se basent sur le gestionnaire de transaction JOTM [JOT] respectant la norme OTS [OTS03]. Largement diffusée, cette norme a été proposée par l'OMG pour définir le service transactionnel de CORBA. Elle définit les transactions plates ainsi que les transactions emboîtées fermées.
Ce moniteur transactionnel JOTM es simplement encapsulé dans un composant primitif, c'est à dire que le code du moniteur, sous forme d'objet, n'est pas modifié, il est simplement accédé à travers ses interfaces. La réutilisation de ce code permet un gain de temps en terme de développement.
Dans ce contexte, il ne nous était pas nécessaire de redécouper le moniteur de transaction

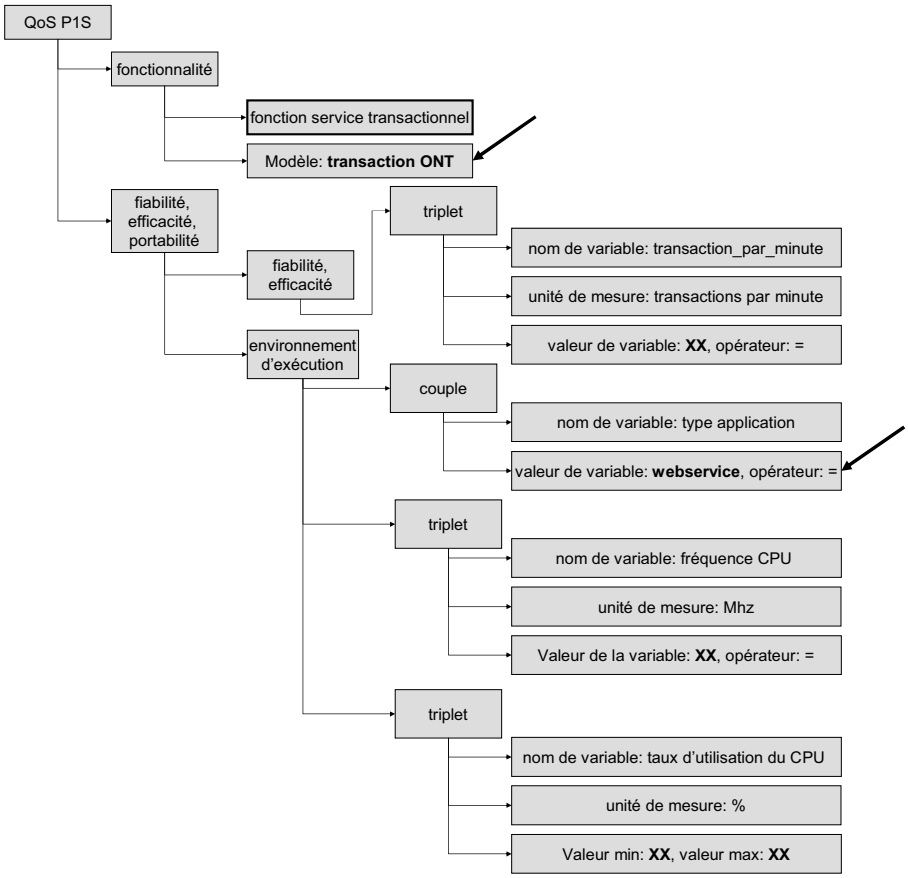

FIG. 3.6 – *Extrait de la P1S transaction ONT partiellement complétée.*

pour l'utiliser, cependant une autre technique consisterait à décomposer le moniteur lui même en sous-composants, on gagnerait certainement en réutilisabilité mais on perdrait sans doute en temps d'exécution. Il serait possible pour cela de s'inspirer de la décomposition proposée par GoTM.

Implantation du modèle des ONT D'après le modèle de transaction ONT, il est nécessaire d'associer une opération de compensation à chaque transaction. Or, chaque appel de méthode est exécuté dans une transaction. Il est donc nécessaire d'associer une méthode de compensation à chaque méthode d'un composant.
Afin de gérer ces méthodes de compensation, S. Nemchenko étend la spécification OTS en ajoutant un mécanisme de compensation. Ce mécanisme est implanté du côté service technique d'un gestionnaire de transaction et du côté applicatif par les compensateurs (un compensateur par composant utilisant des ONT). S. Nemchenko définit un *compensateur* comme l'ensemble des méthodes de compensation d'un composant. Un compensateur étant spécifique au composant auquel il est associé, on peut le voir comme un composant de call-back (voir (4)).
Une transaction ONT impliquant plusieurs sous-transaction et plusieurs composants, lors d'un abandon de transaction, le gestionnaire de transaction ONT doit gérer la compensa-

tion de plusieurs méthodes de plusieurs composants. Pour cela S. Nemchenko propose de décrire le processus de compensation grâce à un descripteur CDO (Compensation Data Object) qui comporte l'identifiant de la transaction, l'identifiant de l'interface et de la méthode ainsi que la liste des valeurs passées en paramètre. Ces CDO sont rangés dans une table "CDOtable" qui permet à partir du CDO d'une transaction mère de retrouver les CDO des transactions filles. Un gestionnaire de compensation consulte la table des CDO, identifie les composants concernés puis exécute les compensations nécessaires.
On identifie donc deux modules logiciels spécifiques à l'exécution des ONT : la table des CDO et le gestionnaire de compensation.

(3) Composition du service transactionnel multi-personnalités

Nous avons vu qu'il existait différents types de transactions. D'après le type d'application et son environnement d'exécution, un modèle sera plus adéquat que l'autre. Pour les applications courtes où les bases de données sont locales, avec peu de pannes, on privilégiera les transactions plates. Pour les applications de type WebService fortement distribuées et sujettes aux pannes, on préférera les transactions imbriquées ouvertes.
L'implantation du service transactionnel tel que spécifié par S. Nemchenko et développé

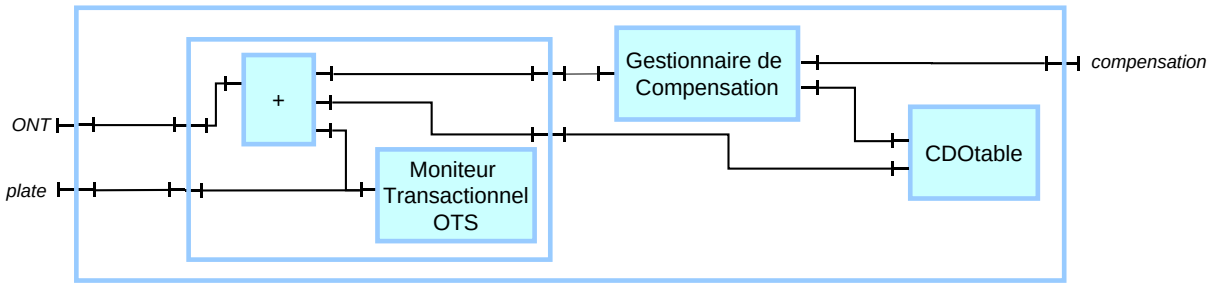

FIG. 3.7 – *Composition du service transactionnel.*

selon nos recommandations fournit les transactions plates et les ONT. Il prend la forme d'un composant composite dont chacune des interfaces correspond à un modèle transactionnel (voir figure 3.7). Nous avons déjà identifié deux composants : la table des CDO et le gestionnaire de compensation. De plus, nous avions mentionné que la base des spécifications repose sur un gestionnaire de transaction OTS (voir 3.1.2). Ce gestionnaire est associé, dans un composant composite, au composant "+" permettant la gestion de fonctionnalités annexes des transactions imbriquées tel que le verrouillage des données.

(4) Assemblage du service transactionnel et du composant applicatif

Les compensateurs de call-back, sont intégrés dans le sous-contrôleur du composant applicatif (voir figure 3.8), simplifiant la gestion de la compensation en maintenant une unité physique. Même intégrés aux composant applicatif, ils restent indépendants dans la mesure où l'on peut les interchanger facilement, cette opération nécessitant tout de même un redéploiement du composant.

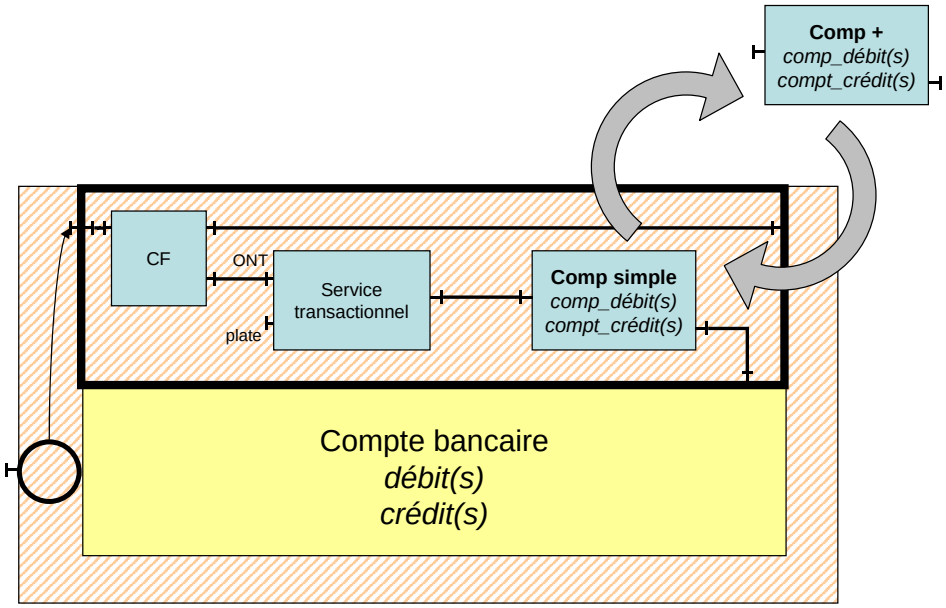

FIG. 3.8 – *Exemple de compensateur.*

Le concept de composant de call-back est bien adapté à la mise en œuvre de la compensation des transactions ONT. Il en simplifie la gestion et permet en intégrant le composant de call-back au composant applicatif de garder une cohérence de l'application. De plus, le concept facilite le changement de politique indépendamment de l'implantation du composant applicatif.

(5) Complément des descriptions des personnalités

Après tests, nous avons complété les P1S présentées dans les figures 3.9 et 3.10. Dans la figure 3.5, outre le fait que l'implantation de service technique associée fournit un service de transactions plates, cette P1S signifie que l'environnement d'exécution pour lequel elle a été conçu se caractérise par un processeur dont la fréquence est supérieure à 200 Mhz. Lorsque le processeur est utilisé entre 30% et 60% de ses capacités, le service transactionnel peut exécuter jusqu'à 19 transactions par minute pour un processeur à 200 Mhz (respectivement 23 transactions par minute pour un processeur à 300 Mhz). Notons que cette figure ne représente qu'un extrait de la description que l'on peut retrouver en entier en annexe de ce document.
Contrairement à la P1S "transaction plate", la P1S ONT est adaptée aux Webservices (cf figure 3.6). Son environnement d'exécution doit se caractériser par un processeur dont la fréquence est supérieure à 300 Mhz. Lorsqu'elle est exécutée dans les même conditions que celles décrites pour la première P1S, elle peut exécuter 21 transactions par minute.

Pour choisir entre l'une ou l'autre de ces P1S, on utilise les deux critères : environnement d'exécution et besoin de l'application. Prenons comme premier exemple une

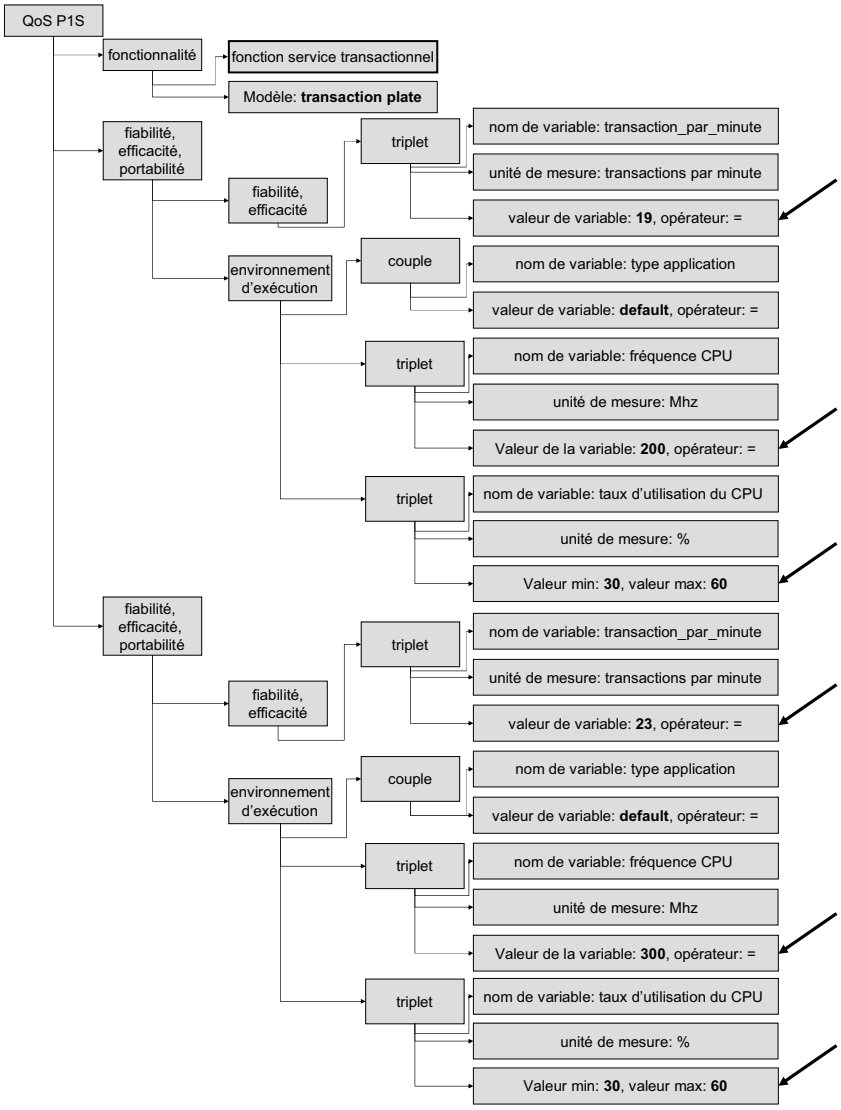

FIG. 3.9 – *Extrait de la P1S transaction plate.*

application

- dont l'environnement d'exécution est une machine dont la fréquence du CPU est égale 200 Mhz et l'utilisation du CPU se trouve entre 30 et 60%
- dont les besoins sont un service transactionnel, implantant les modèles des transactions plates ou des ONT et les préférences sont qu'on prenne en compte le fait que c'est une application de webservice (voir figure 3.11 pour une représentation graphique des besoins de l'application, ou les annexes pour une représentation XML).

Les P1S sélectionnées doivent répondre aux critères :

- s'exécute dans l'environnement décrit;
- fournit un service transactionnel dont le modèle est celui des transactions plates ou des ONT

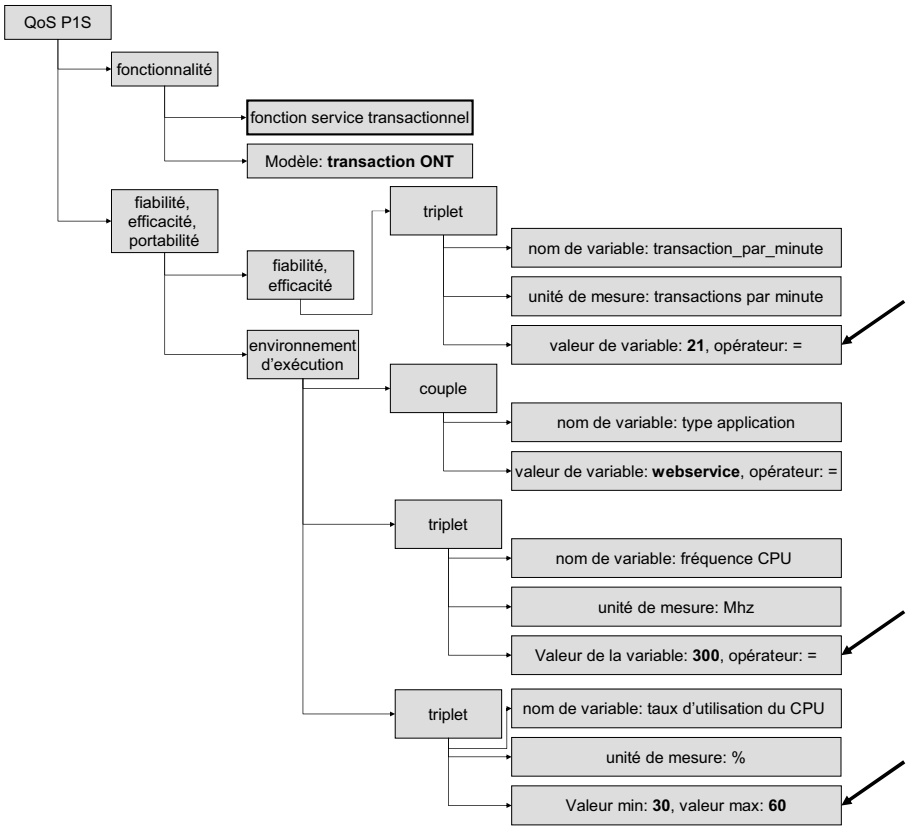

FIG. 3.10 – *Extrait de la P1S transaction ONT.*

La seule P1S que peut nous fournir la plate-forme est donc la P1S transactions plates car la P1S transaction ONT n'a pas été conçue pour fonctionner dans ce genre d'environnement.

Dans le second exemple, on change d'environnement : l'environnement d'exécution est une machine dont la fréquence du CPU est supérieur 300 Mhz et l'utilisation du CPU se trouve entre 30 et 60%. Dans la liste des résultats, on retrouve donc nos deux P1S. Dans un second temps on affine cette solution avec une préférence pour les P1S adaptée aux applications de WebService. La P1S renvoyée est donc la P1S offrant les transactions ONT.
La forme adoptée pour les descriptions des P1S et des besoins de l'application est celle d'un fichier XML. Leurs DTD sont définies en annexe 1 et 2. Les descriptions complètes des deux personnalités sont en annexe 3 et 4, celle des besoins de l'application en 5.

Analyse des résultats

Comme nous l'avions vu dans la partie 1.4.3, notre ambition est de permettre l'adaptation des services techniques à quatre niveaux. Au premier de ces niveaux, un service technique devait pouvoir bénéficier des services dont il avait besoin, ni plus ni moins. Ceci est permis grâce à l'assemblage entre composant applicatif et services techniques et la notion de sous-contrôleur de service technique. Le second niveau doit permettre au

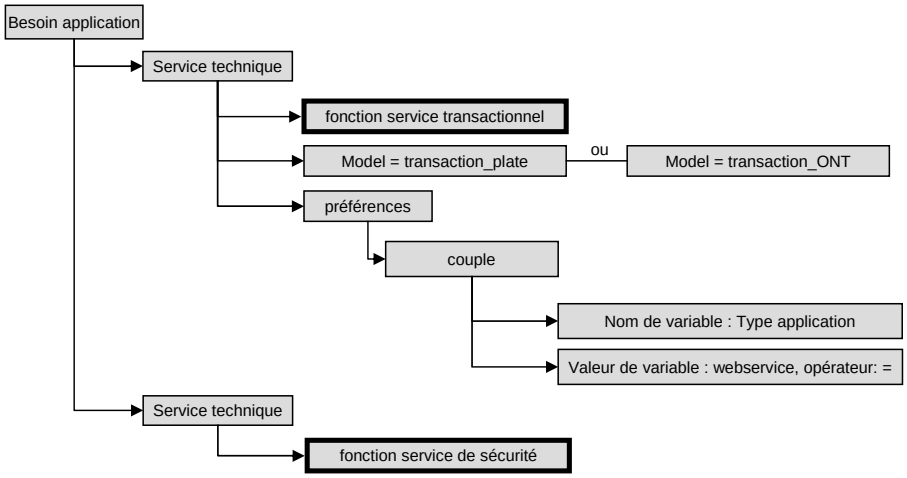

FIG. 3.11 – *Exemple de description d'une application.*

composant applicatif de n'utiliser qu'une partie des fonctionnalités d'un service technique. Nos travaux préconise l'utilisation des composants et plus particulièrement des composants composite. Leur modularité facilite l'identification des fonctionnalités et la définition de plusieurs personnalités de service utilisant l'une ou l'autre des fonctionnalités. De même au troisième niveau, plusieurs versions d'un même service sont rendues disponibles à travers les différentes personnalités qui regroupent une définition du service offert et requis ainsi qu'une composition de composants. Ici nous avons pu utiliser deux personnalités différentes (transactions plates et ONT). Le quatrième niveau définit l'adaptation à travers la personnalisation d'un service technique. Cela se fait par la paramétrisation des composants. Chaque paramétrisation particulière d'un service technique correspondant à une personnalité. Nous vérifions donc que les quatres niveaux d'adaptabilité sont permis grâce aux solutions proposées.

Le service transactionnel testé intègre un gestionnaire de transaction JOTM1.4.1. Les tests effectués consistent en 100 accès consécutifs transactionnels à une base de données Mckoi SQL Database v1.00. Le code est compilé avec le jdk 1.4.1, exécutés sur une machine P3 à 700 MHz avec 340 Mo de mémoire vive dans un environnement Windows 2000.

On pouvait s'attendre à une forte augmentation du temps d'exécution du service liée à l'utilisation du modèle à composants pour l'implantation du service transactionnel. Cependant, lorsqu'on compare les temps d'exécution du service transactionnel à bases d'objet et à base de composants (voir figure 3.12 et tableau 3.1), on constate une augmentation du temps d'exécution très faible. On passe de 258 289 msec pour le service à base d'objets à 258 653 msec pour le service à base de composants, soit une augmentation de moins de 1% (voir tableau 3.1).
On constate que Julia est bien adaptée à l'utilisation que nous en faisons, dans la mesure où le temps d'exécution des composants Julia est assez rapide. Cette rapidité d'exécution s'explique principalement par le fait que cette plate-forme fournit des composants dont

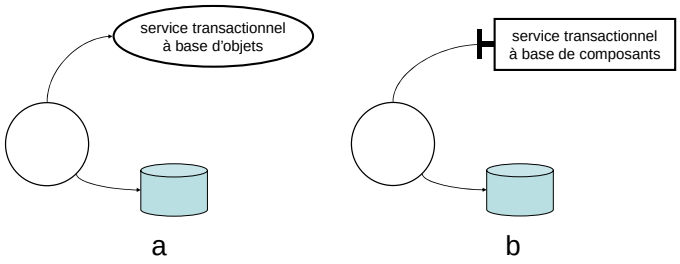

FIG. 3.12 – *a) service technique à base d'objets, b) service technique à base de composants*

service transactionnel	à base d'objets	à base de composants
temps (msec)	258 289	258 653
augmentation (%)	0%	0,14%

TAB. 3.1 – *Vitesse d'exécution du service transactionnel multi-personnalités*

le conteneur est minimal [DEM 04]. Cependant, le découpage en composants a été limité grâce à la réutilisation du gestionnaire de transaction JoTM tel quel. On voit que le découpage par composants est une solution pour la conception des services techniques si à condition que l'on ne multiplie pas les redécoupages en sous-composants.

Améliorations possibles

Le travail de conception comprend une phase de description des différentes personnalités du service technique. Cette phase est totalement laissée à la charge du développeur de services techniques car c'est lui qui en a la meilleure connaissance. Il doit décrire les fonctionnalités, le modèle implanté, qui est simple pour lui. Par contre, il doit aussi décrire l'environnement d'exécution pour lequel le service est adapté et peut ajouter des informations qualitatives sur les performances de son service. Ces informations sont relativement complexes à établir. Il serait vraiment intéressant de concevoir des outils d'aide au développeur de services techniques, notamment des bancs d'essai.

Implantation de la liaison entre composants applicatifs et services techniques

Réalisation L'implantation de la liaison entre les composants applicatifs et les services techniques repose sur le conteneur ouvert de Julia. Dans Julia, les différentes configurations du conteneur sont décrites dans un fichier de configuration qui donne les classes implantant les intercepteurs et les contrôleurs ainsi que leurs associations (i.e. quels contrôleurs sont associés à quels intercepteurs). Ce fichier de configuration est utilisé au moment de la création du composant, il n'est donc pas possible d'ajouter dynamiquement de contrôleur comme nous l'avons spécifié dans la partie "propositions". Nous avons donc opté pour l'association de plusieurs services techniques à un seul contrôleur. Ce contrôleur offre une interface spécifique au contrat afin qu'il puisse gérer l'ajout et le retrait de services techniques. Il gère la liste des références sur les interfaces des services techniques et

lorsque l'intercepteur fait appel à lui, il exécute les méthodes "pré" des services techniques puis les méthode "post".

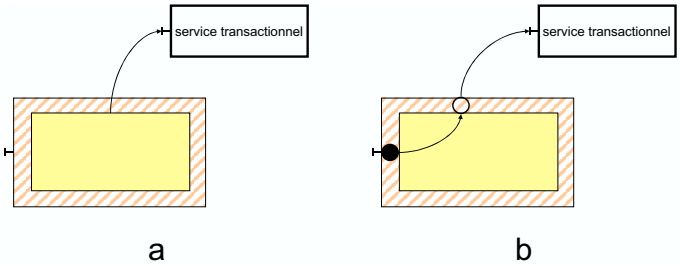

FIG. 3.13 – *a) service transactionnel appelé dans le code applicatif, b) service technique appelé par le conteneur*

Analyse des résultats Comme nous l'avons souligné précédemment, le temps d'exécution induit par l'utilisation du conteneur pour la gestion des services techniques est assez faible. Nous le vérifions grâce au test suivant[4] (voir figure 3.13 et tableau 3.2) : a) le service transactionnel est appelé directement dans le code du composant applicatif, b) le service transactionnel est appelé automatiquement lors d'un appel au composant grâce au contrôleur. On constate que les deux types de gestion ont des temps tout à fait équivalents. On constate que l'utilisation des contrôleurs de Julia est viable.

appel au service technique	dans le code applicatif	grâce au conteneur
temps (msec)	238 8161	238 870
augmentation (%)	0%	0.02%

TAB. 3.2 – *Impact de l'appel au service technique à travers le contrôleur*

Améliorations possibles L'idée de faire des contrôleurs eux-mêmes basés sur des composants n'a pas été mise en oeuvre par manque de temps. Elle induirait certainement une augmentation du temps d'exécution due à l'utilisation du modèle à composants.

3.1.3 Synthèse

Pour ce qui est de la partie conception, nous avons présenté un service transactionnel multi-personnalité implantant les transactions plates et les ONT. En appliquant de façon concrête nos recommandations au service transactionnel, sa conception et son développement ont été facilités. L'utilisation du modèle à composants pour la conception et l'implantation des services techniques s'est révélée efficace. En effet, on a pu diviser les tâches en sous-composants, ce qui a permis d'implanter de façon beaucoup plus claire

4. Les tests on était effectués dans des conditions similaires au précédent test, si ce n'est que la machine utilisée est équipée d'un processeur P4 à 1.5 GHz avec 512 Mo de mémoire vive.

et rapide la compensation. La réutilisabilité du code a été accrue de trois façons. Pour développer le code du gestionnaire de transactions plates, on a facilement encapsulé du code objet, ce qui a permis un gain de temps. De plus, ce composant est partagé par les deux personnalités du service. Si l'on veut changer la politique de compensation, il suffit de changer de composant de compensation ce qui peut être fait dynamiquement. Grâce à cette conception, on a développé un service transaction qui garantit, grâce à un mécanisme de verrouillage adéquat, la cohabitation de deux composants applicatifs utilisant différents types de transaction.
Ce service nous a permis de démontrer l'intérêt d'une telle démarche de conception. Grâce à la description des personnalités, nous pouvons intégrer ce service à notre plate-forme de services techniques adaptables et permettons au système de choisir le modèle transactionnel le plus adapté. Elle a permis un gain de temps de conception qui est difficilement quantifiable, néanmoins, on peut remarquer qu'une fois les spécifications du service transactionnel obtenues, l'intégration du code existant et le développement des autres mécanismes (verrouillage, call-back, etc) n'ont pris que quelques jours.

3.2 Validation des composants de gestion des services techniques

Cette partie est consacrée à la validation du concept de composant de gestion au travers de son implantation et de tests qui nous ont permis de vérifier la faisabilité d'une telle solution. Elle nous donne aussi des pistes pour l'amélioration de ce prototype.

3.2.1 Implantation

L'ensemble des composants de gestion a été mis en oeuvre sous forme de composants Julia, excepté le moniteur (voir figure 3.14). L'ensemble du code regroupe :

- 83 classes Java;
- 44 interfaces Java;
- 10 fichier FractalADL;
- 7 fichiers cpp.

Ces composants sont associés dans un composant composite qui possède deux interfaces : une interface pour administrer les applications (i.e. leur créer un contrat) et une interface pour administrer les services techniques (i.e. les référencer dans l'annuaire).
Ce composant composite comporte les composants de gestion : le coordinateur, l'annuaire et les contrats. Outre ces composants, dont nous détaillerons l'implantation dans les parties suivantes, on trouve trois autres composants : un "super contrat", un répartiteur et un assistant de requête. Ces composants effectuent des tâches annexes :

- le super contrat regroupe tous les contrats. Il est nécessaire car lors d'un ajout de composant dans un composant composite, ce composant doit être arrêté. C'est le cas lorsqu'on veut ajouter un nouveau contrat. En évitant l'arrêt du composant global, le super contrat permet donc de limiter le nombre de composants à stopper;

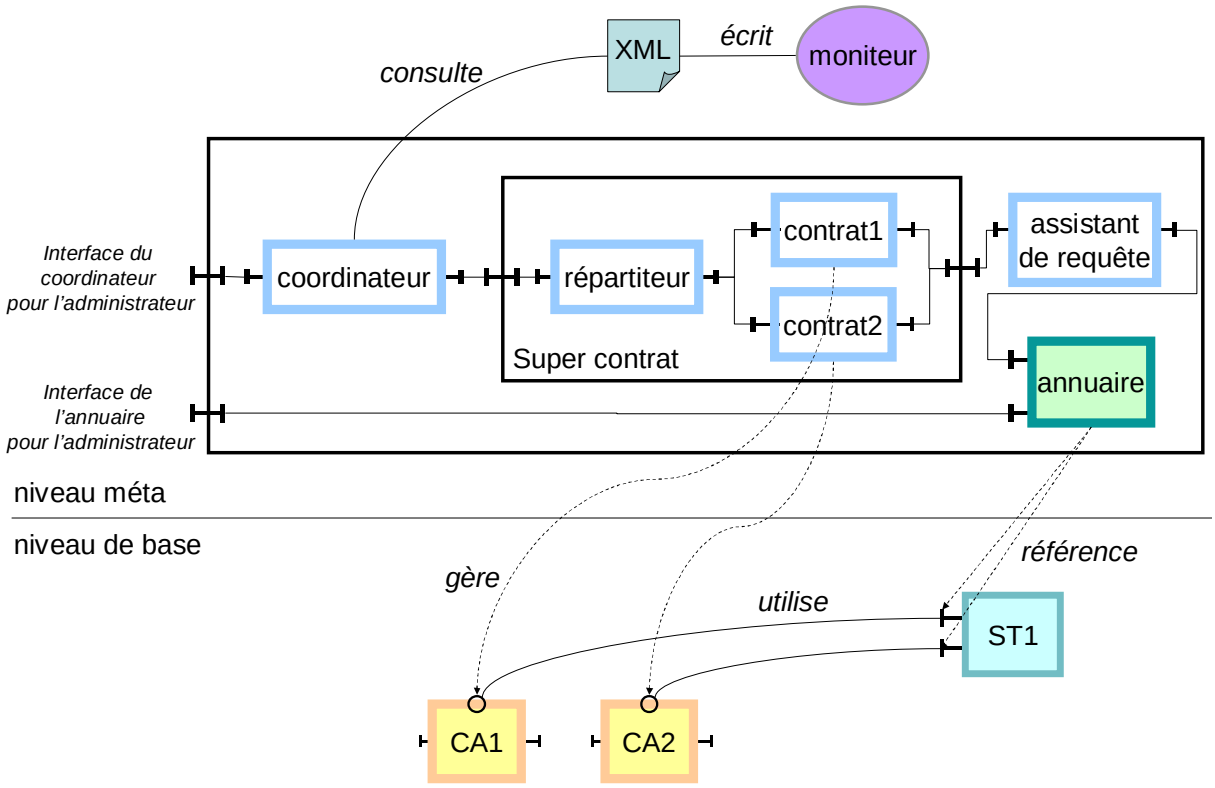

FIG. 3.14 – *Implantation des composants de gestion*

- le composant répartiteur permet de gérer la liaison multiple qui existe entre le coordinateur et les composants contrat;
- le composant assistant de requête permet d'assurer la correspondance entre le format des requêtes des contrats et celui des requêtes de l'annuaire.

Il existe :

- 17 composants composites;
- 54 composants primitifs;

Ces composants sont liées à l'aide de :

- 97 + (2* le nombre de contrats) liaisons.

Le moniteur, écrit en C, est à part. Le détail des composants se trouve en annexe (dans la partie IV, section 8 pour la représentation graphique des composants et sous-composants, section 9 pour le nombre de liaisons et de sous-composants).

Coordinateur

Le coordinateur gère la cohérence du système : il garde une référence sur le fichier grâce auquel il échange des informations sur l'environnement avec le moniteur. Il crée aussi un composant contrat pour chaque application qui s'enregistre auprès de lui. Puis il établit les liaisons du contrat avec le répartiteur et le super contrat. Il gère les contrats, avec

l'aide du répartiteur en leur fournissant les informations sur l'environnement d'exécution et en déclenchant leur mise à jour.

Contrats

A l'ajout d'une application au système, le coordinateur crée dynamiquement un composant primitif contrat. Tous les contrats dérivent du même patron et sont paramétrés grâce à une référence sur le contrôleur du composant applicatif et les besoins de l'application en terme de services techniques. Le contrat nouvellement créé interroge l'annuaire de services techniques en lui fournissant les informations sur l'environnement d'exécution et les besoins de l'application qu'il représente (voir figure 3.15). Les références sur les

```
<?xml version="1.0" encoding="UTF-8"?>
<qs:Query xmlns:qs="http://objectweb.org/query">
   <qs:Function>Transaction Service</qs:Function>
   <qs:propertyMatching>
      <!-- Constraints -->
      <qs:Constraints>
          <!--Instance query-->
          <qs:queryInstance>
             <!--Technical properties query-->
             <qs:queryTechnical>
                <![CDATA[ (version = FLAT)]]>
             </qs:queryTechnical>
             <!--Environment properties query -->
             <qs:queryEnvironment>
                <![CDATA[ (memory = 100 & cpu = 100) ]]>
             </qs:queryEnvironment>
          </qs:queryInstance>
      </qs:Constraints>
      <!--Preferences-->
      <qs:preferenceEnvironment priority="8">
         <![CDATA[ (memory = 100 )]]>
      </qs:preferenceEnvironment>
      </qs:Preferences>
   </qs:propertyMatching>
</qs:Query>
```

FIG. 3.15 – *Exemple de requête d'exportation d'un service de transactions imbriquées*

services techniques ainsi récupérées sont enregistrées dans le contrat. Enfin, le contrat crée la liaison entre le composant applicatif et les services techniques enregistrés par l'intermédiaire du contrôleur du composant applicatif. Par la suite, à chaque modification de l'environnement d'exécution ou des besoins de l'application, le coordinateur demande aux contrats de se mettre à jour (e.g. chercher de nouveaux services techniques plus adéquats) par l'intermédiaire du répartiteur.

Annuaire de services techniques

L'annuaire est lui même un composant composite, conçu suivant les spécifications que nous avions indiquées dans la partie "propositions" (voir figure 3.16, en annexe D se trouve une décomposition complète de l'annuaire.) Cet annuaire permet de stocker une référence sur un service technique puis de la retrouver en fonction de leur environnement d'exécution et des besoins de l'application.

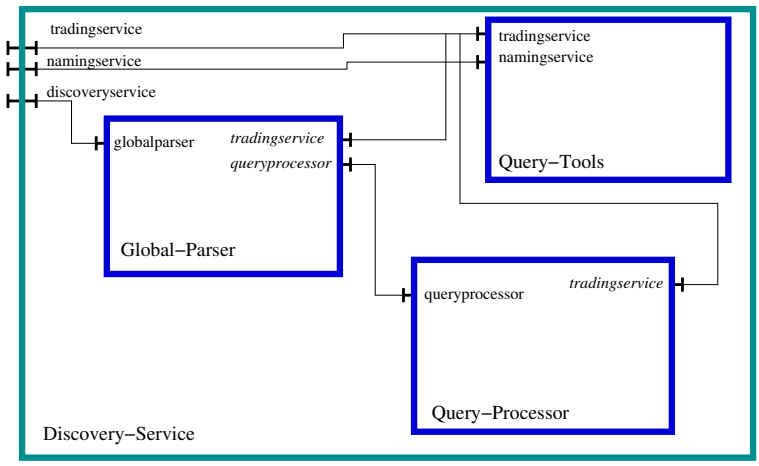

FIG. 3.16 – *Structure du composant Discovery Service*

L'annuaire se compose de trois grand sous-composants : *global parser*, *query processor*, *query tools*.

- *global parser* qui correspond au gestionnaire de requêtes. Il se décompose en :
  - un analyseur syntaxique (*Basic Parser*) qui analyse la requête, valide le XML et détermine le type de la requête (import ou export).
  - un *trading handler* et un *query handler* qui extraient les données pour les déléguer respectivement à l'annuaire de services techniques ou au *query processor*
- *query processor* qui correspond au processeur de requêtes. Il se décompose en :
  - le *query service* qui correspond au service de requête.
  - le *decision service* qui correspond au service de décision.
- *query tools* qui correspond à l'annuaire générique de composants. Il se décompose en :
  - l'annuaire de courtage générique de composants, annoté *trading service*, tels que nous l'avions défini les propositions.
  - un service de nommage, annoté *namming service*.

Les annuaires de courtage et de nommage partagent une même structure de données qui prend la forme d'une base de données relationnelle. Cette structure de données permet de stocker (voir figure 3.17) :

- les types décrits grâce à leurs interfaces qui sont caractérisées par leur nom, leur rôle (client ou serveur) et leur signature;

– les patrons qui dérivent d'un type qui peuvent être primitifs ou composites. Si un patron est composite alors son contenu est vide, sinon, on précise la classe qui l'implante. Il est caractérisé par des attributs sans valeur ainsi que l'identifiant de son père s'il en a un. On précise aussi les liaisons qui existent au sein d'un patron entre les différentes interfaces;
– les instances dont on précise la localisation et que l'on caractérise grâce à leurs attributs et leurs valeurs.

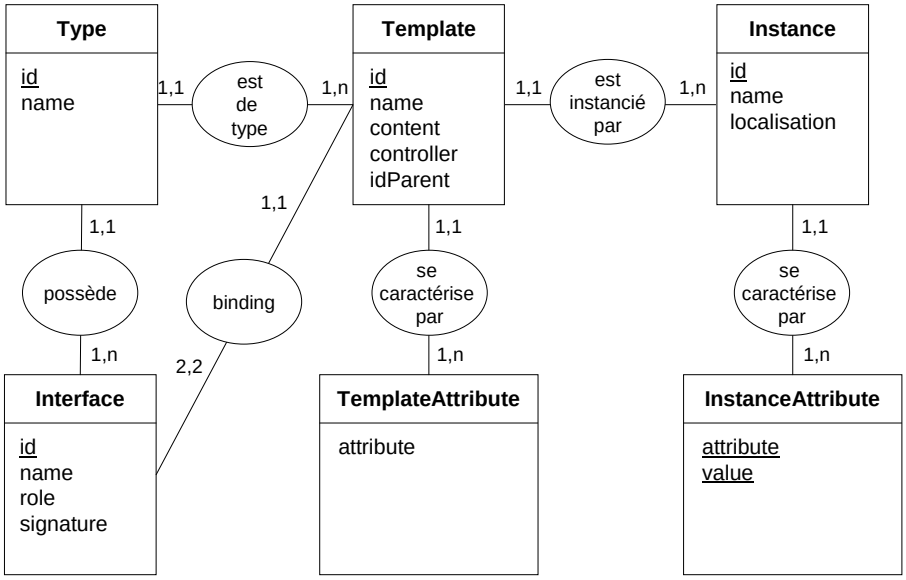

FIG. 3.17 – *Schéma entité-association de la base de stockage de l'annuaire générique*

Cet annuaire nous permet donc de stocker de façon persistante les informations sur les services techniques nécessaires à leur localisation. Néanmoins, il n'implante qu'une version basique de la partie décisionnelle qui renvoie simplement la première solution trouvée. De plus, dans la partie "propositions", nous avons vu que la description de la P1S regroupait plusieurs couples (description de la fiabilité et de l'efficacité, description de l'environnement d'exécution. Or notre annuaire ne permet de stocker qu'un couple à la fois. Il est donc nécessaire de référencer la P1S autant de fois qu'il y a de couples. Enfin, la solution de stockage des données est amenée à évoluer vers une base de données XML pour prendre en compte le fait que les requêtes sont toutes écrites dans ce langage.

Moniteur

Les échanges d'informations sur l'environnement d'exécution des moniteurs avec le coordinateur se font à travers un fichier XML dont on trouve des exemples dans les figures 3.18, 3.19 et dont la DTD se trouve en annexe.
Ces moniteurs ne sont pas écrits en composant Julia. Il n'est pas possible de récupérer des informations bas niveau sur l'environnement d'exécution avec le langage Java dans la mesure où ce langage n'est pas destiné à cela. Les moniteurs sont implantés en C ou en

C++ et compilés pour chaque machine.
Trois moniteurs ont été développés pour les systèmes d'exploitation Windows, Windows CE et Palm OS. Un tableau qui récapitule les détails d'implantation des moniteurs se trouve en annexe.

Windows Un moniteur destiné au système d'exploitation Windows a été développé. Il fonctionne sur les versions supérieures à Windows NT 4.0 SP7 c'est-à-dire Windows 2000 Pro et Windows XP. Il est écrit en C++ à l'aide des bibliothèque pdh.dll (e.g. Performance Data Helper) en utilisant le logiciel de développement Visual C++ 6.0.
Il fournit des informations sur le système d'exploitation, le processeur et la mémoire. La figure 3.18 est un exemple de fichier XML généré grâce à ce moniteur.

```
- <execution_environment_description>
  - <hardware>
    - <hardware_set name="MEMORY">
      - <hardware_element>
          <hardware_property name="use" statORdyn="dyn" unit="%" value="FAIL"/>
          <hardware_property name="available" statORdyn="dyn" unit="MB" value="FAIL"/>
          <hardware_property name="size" statORdyn="stat" unit="MB" value="511"/>
          <hardware_property name="type" statORdyn="stat" value="RAM"/>
        </hardware_element>
      </hardware_set>
    - <hardware_set name="CPU">
      - <hardware_element>
          <hardware_property name="use" statORdyn="dyn" unit="%" value="50"/>
          <hardware_property name="name string" statORdyn="stat" value=" Intel(R) Pentium(R) 4 CPU 1.50GHz"/>
          <hardware_property name="identifier" statORdyn="stat" value="x86 Family 15 Model 1 Stepping 2"/>
          <hardware_property name="frequency" statORdyn="stat" unit="MHz" value="1495"/>
        </hardware_element>
      </hardware_set>
    - <hardware_set name="NIC">
      - <hardware_element>
          <hardware_property name="Max bandwidth" statORdyn="stat" unit="bits/s" value="FAIL"/>
          <hardware_property name="Used bandwidth" statORdyn="dyn" unit="%" value="FAIL"/>
        </hardware_element>
      </hardware_set>
    </hardware>
  - <software>
    - <software_set name="OS">
        <software_element/>
        <software_property name="Version" value="Professional"/>
        <software_property name="SP" value=""/>
        <software_property name="Build" value="2195"/>
      </software_set>
    </software>
</execution_environment_description>
```

FIG. 3.18 – *Exemple de fichier XML produit par le moniteur sur un PC sous Windows*

Windows CE Ce moniteur pour Pocket PC sous Windows CE a été développé à l'aide de Embedded Visual C++, testé grâce à l'émulateur *PocketPC 2003 Emumator*. Il utilise les librairies msxml.h pour le XLM et winbase.h pour le traitement des tâches (e.g. en anglais thread).
Ce moniteur fournit des informations sur le système d'exploitation, le processeur, la mémoire et la batterie. La figure 3.19 donne un exemple de fichier XML généré grâce à ce moniteur.

FIG. 3.19 – *Exemple de fichier XML produit par le moniteur sur un Pocket PC sous Windows CE*

Palm OS Ce moniteur a été développé grâce à l'IDE *CodeWarrior* [Cod]. Il a été testé avec l'émulateur *PalmOS Emulator* avant d'être testé en conditions réelles sur une machine *Tungsten T3*. Il n'utilise pas l'API XLM pour Palm OS car elle est payante et n'est pas multi-threadée dans la mesure où la librairie implantant les tâches n'est pas disponible avant la future version 6 de PalmOS.
Ce moniteur fournit des informations sur le processeur, la mémoire, la batterie, la carte mémoire, les connexions réseaux infrarouge et bluetooth.

Les trois moniteurs n'utilisent pas les mêmes librairies, il a donc été nécessaire de développer trois moniteurs totalement distincts. De plus, il n'a pas été possible en l'état

actuel des choses de développer un moniteur pour Symbian. Enfin, ces moniteurs ne récupèrent pas toutes les données dont on aimerait disposer et ils n'implantent pas la synthèse des données. Il n'a pas été possible de développer un moniteur pour Symbian OS, car il n'existe pas de librairie disponible pour accéder aux données sur l'environnent d'exécution.

3.2.2 Analyse des résultats

Nous avons mesuré le temps que met le système pour changer de services technique lorsque l'environnement d'exécution change. Pour cela, nous avons inséré dans l'annuaire : 4 types de composants, 2 patrons héritant des types et 20 personnalités de services techniques. Nous avons mesuré le temps entre le moment où le système détecte un changement d'environnement et le moment où il finit de mettre à jour la liaison avec le nouveau service technique. Ce temps est en moyenne de 96 msec.
Etant donné que la detection de changement d'environnement se fait toutes les x msec, il faut ajouter aux 96 msec entre 0 et x msec pour obtenir le temps qui s'écoule entre un réel changement d'environnement et la mise à jour du service technique. Ce laps de temps, x msec est à paramétrer en fonction du type d'environnement (très changeant ou pas).

3.2.3 Améliorations possibles

Notre système ne gère pas l'instanciation des services techniques. On part du principe que les services techniques référencés par l'annuaire sont déjà instanciés. Une amélioration de l'annuaire est donc de référencer les patrons des services techniques sans qu'ils soient instanciés et de les instancier si nécessaire. Cela pourrait augmenter le temps de changement d'un service à l'autre mais éviter de surcharger la machine. Le choix entre les deux options pourrait être paramétré.
D'autre part, notre prototype ne gère pas non plus les contradictions qu'il pourrait y avoir entre les besoins de l'application et les contraintes de l'environnement d'exécution. On pourrait résoudre ce genre de problème en mettant des priorités sur les éléments de la requête.
Notre prototype n'implante pas la notion de lot de service. Pour gérer les lots de service, il faudrait modifier l'annuaire de composant générique pour qu'il puisse référencer non pas un composant mais un lot de composants.
Enfin, nous n'avons pas géré les composants de call-back dans le système. Pour cela, il faudrait ajouter un annuaire de composants de call-back, afin de les localiser. Puis les gérer de la même façon que les services techniques grâce au contrat qui les ajouterait au composant et les lierait au sous-contrôleur.

3.3 Conclusion

Pour la partie conception de services techniques et liaison avec les composants applicatifs, la plate-forme Fractal s'est avérée être un bon choix. Elle nous fournit des conteneurs

simples mais ouverts qui sont adaptés à la conception des services techniques et des composants applicatifs. Ils permettent l'ajout de services techniques et ont un bon temps d'exécution. On notera cependant que deux limites de Julia nous sont apparues au cours du projet : les composants ne peuvent être développés qu'en Java ce qui n'était pas compatible avec le développement du moniteur, et les conteneurs sont construits à base de "mixins" dont la manipulation n'est pas aisée et dont l'ajout dynamique est impossible.
Pour la partie composants de gestion, on a réussi à mettre en place un système qui permet d'adapter les services techniques en fonction de l'environnement d'exécution. Le système étant lui même à base de composants, on pourra facilement le faire évoluer et réutiliser les éléments du prototype comme l'annuaire ou le moniteur. Il serait intéressant maintenant d'étendre les tests à plus de personnalités, plus de composants ainsi que l'ajout de plusieurs sous-contrôleurs pour vérifier la montée en charge du système.
la gestion de la distribution n'a pas été une priorité dans notre prototype . Toutefois, on pourrait utiliser FractalRMI, pour masquer la distribution des différents éléments sur des machines distribuées. Reposant sur des éléments de base de Jonathan, cette API fournit une implantation légère du protocole RMI entre les composants Fractal. Plusieurs éléments du système pourraient être distribués : les services techniques, les composants applicatifs, les composants de gestion.

- On peut tout à fait imaginer des services techniques distribués sur des machines distantes, cela pose néanmoins comme problème la dégradation de la qualité de service fournie par le service technique. Cette distribution doit donc être prise en charge dans l'évaluation de la qualité de service fournie par le service technique et celle fournie par l'environnement d'exécution. De plus, il faudrait aussi prendre en compte l'association entre une P1S et son environnement d'exécution, pour l'instant elle est implicite dans le prototype car les P1S s'exécutent sur une seule machine dont on connaît toujours l'environnement d'exécution.
- Les composants applicatifs peuvent tout a fait être distribués, bénéficiant plutôt des services techniques qui leur sont locaux.
- Enfin les composants de gestion sont potentiellement distribués, cependant, mise à part un annuaire distribué référençant des services techniques distants, il semble de peu d'interêt qu'ils se trouvent ailleurs qu'en local.

Troisième partie

Conclusion et perspectives

Conclusion et perspectives

1 Conclusion

Dans ce mémoire, nous proposons un cadre de conception et d'utilisation de services techniques adaptables aux besoins des applications et à leur environnement d'exécution. Jusqu'à présent conçus pour les applications client-serveur "classiques", les services techniques des plate-formes à composants apportent aux applications une grande simplicité d'implantation et une haute qualité de service à l'exécution. Cependant, nous avons mis en évidence les limites de leur conception et de leur utilisation pour les nouvelles applications distribuées. En effet, ce type d'applications s'exécute principalement dans des environnements hétérogènes qui sont souvent plus contraignants, du point de vue des ressources de fonctionnement, que les machines de type serveur.
La conception actuelle des services techniques, sous forme d'objets, ne semble plus appropriée. Elle ne permet pas d'adapter facilement le comportement des services techniques aux modifications de l'environnement d'exécution. De plus, les services techniques étant fournis de façon transparente mais figée aux composants applicatifs par l'intermédiaire d'un conteneur fermé, il n'est pas possible de rectifier l'utilisation des services au cours de l'exécution.

Nos études se sont portées sur la flexibilité des services techniques, que ce soit dans leur phase de conception ou dans leur phase d'exécution. Deux contraintes principales étaient posées : maintenir la simplicité d'implantation des composants applicatifs et des services techniques et assurer l'interopérabilité de deux composants applicatifs quelques que soient les services techniques qu'ils utilisent.

La première partie de notre travail a été consacrée à la définition d'un cadre de conception pour les services techniques adaptables. Nous définissons les notions de P1S et de L2S qui, projetées sur les différentes phases du développement du service technique, donnent une représentation aux différentes personnalités d'un service technique.
Nous avons montré les avantages du modèle à composants dans ce cadre et en particulier du modèle Fractal. Le modèle à composant, basé sur le principe de modularité, accroît la réutilisabilité et le partage des fonctionnalités entre plusieurs implantations d'un service technique. Le modèle Fractal fixe les bases d'un modèle à composants hiérarchique et extensible. Il nous permet de définir un service technique comme étant un ensemble d'assemblages de composants, ayant des composants en commun.
Nous utilisons la définition d'un conteneur ouvert, fournie par Fractal, pour redéfinir

l'assemblage entre les services techniques et les composants applicatifs. Celui-ci était jusqu'alors fixe et non explicite; le code correspondant à l'utilisation du service technique était enfoui dans le code du conteneur qui était fermé et utilisait des références sur les services techniques convenues au moment de la compilation. Nous complétons la définition faite par le modèle Fractal : un conteneur est un assemblage de composants regroupant les intercepteurs, les services techniques ainsi que les composants de call-back qui leur sont associés. Ce nouvel assemblage explicite et plus souple permet l'ajout, le retrait ou le changement de service technique ou de composant de call-back.
La conception et le développement d'un service transactionnel multi-personnalité au sein de notre équipe de recherche fut l'occasion de valider les propositions faites dans cette première partie. Le service transactionnel ainsi développé fournit les transactions plates ainsi que les transactions imbriquées ouvertes. Il permet à deux composants utilisant des personnalités différentes de cohabiter grâce au partage, par les deux personnalités, du moniteur transactionnel.

Si le premier volet de ce travail a pour but de rendre les services techniques adaptables, le second volet s'intéresse aux mécanismes de l'adaptation. Il définit un ensemble de composants de gestion qui permettent de localiser, de choisir et d'utiliser les services techniques en fonction de l'environnement d'exécution et des besoins de l'application.
Le contrat représente, au niveau des composants de gestion, l'assemblage entre les services techniques et le composant applicatif. C'est à travers lui que les composants de gestion agissent sur la liaison entre les personnalités de services techniques et le composant applicatif. Le moniteur fournit une représentation de l'environnement d'exécution utilisé pour choisir la personnalité de service adéquate. Le coordinateur maintient la cohérence entre les composants de gestion et initie l'adaptation. Enfin l'annuaire de services techniques permet la localisation efficace des services techniques.
Tous ensemble, ces composants de gestion adaptent dynamiquement l'utilisation des personnalités de services techniques en fonction des changements de l'environnement d'exécution.

Afin de valider les propositions faites dans ces deux parties, un prototype a été réalisé. Implanté grâce à Julia, il permet de vérifier la faisabilité de notre solution. A travers l'exemple du service transactionnel, nous avons pu testé la faisabilité de l'adaptation dynamique des services techniques. Cependant, il montre aussi les améliorations possibles de cette implantation du modèle notamment concernant le contrôleur.

2 Perspectives

2.1 Perspectives à court terme

Implantation d'un contrôleur à base de composants

Le modèle à composant Fractal spécifie la forme générale des contrôleurs (à base de sous-contrôleur et d'intercepteur) et laisse toute liberté quant à son implantation. L'implantation de ce contrôleur dans Julia n'est pas totalement satisfaisante dans le contexte

de nos recherche : on aimerait notamment pouvoir ajouter ou retirer dynamiquement les sous-contrôleurs. La solution proposée, c'est-à-dire implanter les contrôleurs à base de composants, n'a pas été mise en œuvre faute de temps. Elle permettrait une gestion explicite de la composition du contrôleur.

Continuité dans le changement de personnalités

Lorsque l'on change d'implantation de service technique, il faut assurer une continuité dans le rendu du service. Par exemple, il est nécessaire qu'une transaction qui a été débutée avec un service transactionnel ONT se termine avec le même gestionnaire. Notre solution ne spécifie pas de mécanisme précis pour s'assurer de cette continuité. Cette difficulté peut néanmoins être atténuée pour des services techniques n'intervenant que ponctuellement ou des personnalités partageant des composants grâce auxquels elles peuvent échanger des informations.

Assistance au choix du profil requis par le composant applicatif

Que ce soit dans un modèle tel que les EJB ou dans notre solution, la gestion des services techniques se fait composant par composant. Cela nécessite que le développeur de composant spécifie pour chaque composant les besoins en terme de services techniques. Dans un modèle à composant récursif, tel que Fractal, on se demande s'il ne serait pas intéressant de pouvoir définir des politiques de gestion des services techniques pour les sous-composants d'un composant composite ou encore pour des classes de services techniques tel que cela est fait dans les EJB. On aurait alors la facilité des EJB mais aussi la flexibilité d'un système où l'on peut ajouter ou retirer des services facilement.

2.2 Perspectives à long terme

Définition de la qualité de service d'une composition de composants

Notre solution repose sur la définition de la qualité de service de plusieurs éléments du système : qualité de service fournie par une P1S, un L2S ou requise par une application. Ces informations peuvent être extraites de tests ou de spécifications formelles. Néanmoins, définir la qualité de service nécessite de tester le composant dans sa globalité. Si ce composant est un composant composite, on ne pourra pas se reposer uniquement sur les définitions de qualité de service des éléments qui le composent. Or, la connaissance de la qualité de service d'une composition de composants intervient à deux niveaux dans nos travaux :

- pour le choix d'une P1S ou d'un L2S;
- pour déterminer la qualité de service fournie par composant global formé par le composant applicatif (qu'il soit composite ou non) et les services techniques.

Pouvoir prédire la qualité de service en fonction de la qualité de service des composants qui le composent reste un problème complexe. La résolution de ce problème passe tout d'abord par une définition complète de contrats de qualité de service.
Cette définition doit prendre en compte les interactions entre les composants, leur distribution ainsi que l'environnement d'exécution. Pour cela, il faut tout d'abord déterminer

la portée des contrats : opération, interface, port, composant [TRA 04]. Au niveau du composant, on peut même affiner cette portée en associant un contrat à la partie interne du composant ou à sa partie externe [COL 05].
De plus, ces contrats peuvent être définis à plusieurs niveaux de conception. Dans [BEU 99], A. Beugnard et ses collaborateurs définissent quatre niveaux de contrats, du moins négociable au plus négociable :

- *niveau syntaxique* correspondant notamment aux IDL ou le contrat syntaxique défini par O. Barais et ses collaborateurs dans [BAR 04], il est couramment utilisé dans les modèles à composants à travers le typage des composants;
- *niveau comportemental* correspondant aux pré et post conditions comme celles du langage Eiffel, les contraintes OCL, ou le contrat d'assertion de [BAR 04];
- *niveau de synchronisation* avec les objets de synchronisation. Ce niveau décrit le type de dépendances existant entre les services fournis par un composant (ex : en série, en parallèle). Cet aspect n'a pas été abordé dans cette thèse, il relève d'un niveau plus formel de spécifications.
- *niveau qualité de service* largement abordé dans cette thèse comme CQML.

Notre solution intègre le premier niveau, qui est fourni par la plate-forme à composants, et définit le quatrième pour les services techniques. On constate cependant que deux niveaux ne sont pas intégrés et qu'il existe une disparité dans le traitement des niveaux fournis. Ces niveaux pourraient être modélisés comme différentes vues d'un même composant et traités grâce à des outils formels [PET 03], semi-formels, basés sur le programmation par contrats [MEY 92], etc.

Représentation dynamique d'une application

Les approches actuelles pour spécifier les architectures dynamiques des applications reposent sur les ADL [MED 00] tels que Wright, UniCon, Rapide, ou Fractal-ADL. Dans un premier temps, ces langages permettent de fournir une vision abstraite de l'application, indépendante du modèle d'implantation, sur laquelle on peut raisonner. Cette vision permet de mieux appréhender l'architecture de l'application et les interactions entre des composants. Elle permet aussi d'exprimer les contraintes et les dépendances existantes entre les composants. Dans un second temps, selon l'approche MDA, cette représentation de l'application peut être projetée, selon des règles bien définies, sur un modèle d'implantation spécifique.
Au niveau de l'implantation, on ne retrouve pas les spécifications faites au niveau de l'ADL, ce qui pose un problème pour l'adaptation. En effet, on n'a plus les informations nécessaires sur la structure ou le comportement des composants.
Dans nos travaux, les développeurs spécifient, pendant la conception, le comportement de certains composants (description de la personnalité, des besoins du composant applicatif), puis cette spécification est projetée au niveau implantation et utilisée pour l'adaptation. De plus, dans le modèle Fractal un composant est capable de fournir des informations sur les interfaces qu'il fournit et sur sa composition.
Il serait certainement intéressant de compléter cette représentation du comportement et

de la structure des composants en conservant, au niveau implantation, les informations spécifiées au niveau de la conception. Cette représentation évoluerait au cours des différentes adaptations afin de toujours refléter les composants.

Adaptation au sein du composant technique

Dans nos travaux, ce n'est pas le service technique à proprement parler qui s'adapte en adoptant le comportement de telle ou telle personnalité. En réalité, le système adapte plutôt l'utilisation qui est faite d'un service. C'est une démarche globale d'intégration de différents services existants.
L'approche qui consiste à adapter le comportement d'un service technique est une voie prometteuse dans la mesure où les services obtenus sont optimisés. C'est l'approche adoptée notamment dans ArcticBeans [AND 01]. Cependant, les solutions, totalement spécifiques au domaine traité (par exemple un langage de description de la politique de sécurité dans ArcticBeans) ne sont pas réutilisables.
Il serait intéressant maintenant, en utilisant le retour d'expérience acquis grâce aux différentes approches, d'essayer de dégager une méthodologie globale pour la conception de ce genre de service et de l'outiller. Outre les concepts de moniteur et d'annuaire qui pourrait être réutilisés, un langage de description d'adaptation (par réassemblage ou par paramétrage de composants) ou des modules d'apprentissage permettant au système de tirer parti de son expérience seraient des outils intéressants pour l'adaptation non seulement du code technique mais aussi des applications.

Quatrième partie

Annexes

annexes

1 DTD de description d'une P1S

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT TechnicalService (property*, preferences*)>
<!ATTLIST TechnicalService function NMTOKEN #IMPLIED>

<!ELEMENT property ( (supinf | minmax)+ ) >
<!ATTLIST property name NMTOKEN #REQUIRED >
<!ATTLIST property value NMTOKEN #REQUIRED >
<!ATTLIST property unit NMTOKEN #IMPLIED>
<!ATTLIST property composition NMTOKEN #IMPLIED>

<!ELEMENT supinf () >
<!ATTLIST supinf operateur NMTOKEN #REQUIRED>
<!ATTLIST supinf value NMTOKEN #REQUIRED >

<!ELEMENT minmax () >
<!ATTLIST minmax valuemin NMTOKEN #REQUIRED>
<!ATTLIST minmax valuemax NMTOKEN #REQUIRED >

<!ELEMENT preferences ( preferenceTechnical*, preferenceEnvironment*) >

<!ELEMENT preferenceTechnical (property) >
<!ATTLIST preferenceTechnical priority NMTOKEN #REQUIRED >

<!ELEMENT preferenceEnvironment (property) >
<!ATTLIST preferenceEnvironment priority NMTOKEN #REQUIRED
```

2 DTD de description des besoins d'une application

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT NeedOfApplication (TechnicalService*)>

<!ELEMENT TechnicalService (property*, preferences*)>
<!ATTLIST TechnicalService function NMTOKEN #IMPLIED>

<!ELEMENT property ( (supinf | minmax)+ ) >
<!ATTLIST property name NMTOKEN #REQUIRED >
<!ATTLIST property value NMTOKEN #REQUIRED >
<!ATTLIST property unit NMTOKEN #IMPLIED>
<!ATTLIST property composition NMTOKEN #IMPLIED>

<!ELEMENT supinf () >
<!ATTLIST supinf operateur NMTOKEN #REQUIRED>
<!ATTLIST supinf value NMTOKEN #REQUIRED >

<!ELEMENT minmax () >
<!ATTLIST minmax valuemin NMTOKEN #REQUIRED>
<!ATTLIST minmax valuemax NMTOKEN #REQUIRED >

<!ELEMENT preferences ( preferenceTechnical*, preferenceEnvironment*) >

<!ELEMENT preferenceTechnical (property) >
<!ATTLIST preferenceTechnical priority NMTOKEN #REQUIRED >

<!ELEMENT preferenceEnvironment (property) >
<!ATTLIST preferenceEnvironment priority NMTOKEN #REQUIRED >
```

3 DTD de l'environnement

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT execution_environment_description (hardware, software)>

<!ELEMENT hardware (hardware_set* ) >

<!ELEMENT hardware_set ( hardware_element* ) >
<!ATTLIST hardware_set name NMTOKEN #REQUIRED>

<!ELEMENT hardware_element (hardware_property*)>

<!ELEMENT hardware_property (EMPTY)>
<!ATTLIST hardware_property name NMTOKEN #REQUIRED>
<!ATTLIST hardware_property value NMTOKEN #REQUIRED>
<!ATTLIST hardware_property statORdyn NMTOKEN #REQUIRED>
<!ATTLIST hardware_property unit NMTOKEN #IMPLIED>

<!ELEMENT software (software_set* )>

<!ELEMENT software_set ( element* ) >
<!ATTLIST software_set name NMTOKEN #REQUIRED>

<!ELEMENT software_element (hardware_property*)>

<!ELEMENT software_property (EMPTY)>
<!ATTLIST software_property name NMTOKEN #REQUIRED>
<!ATTLIST software_property value NMTOKEN #REQUIRED>
```

4 P1S de la personnalité "transaction plate" du service transactionnel

<TechnicalService function="transaction">
<property name="model" value="transaction_plate"/>
<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="24" unit="nb/min"/>
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="0" valuemax="30"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>

<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="23" unit="nb/min"/>
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="30" valuemax="60"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>

<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="15" unit="nb/min"/>

```
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="60" valuemax="100"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>
<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="20" unit="nb/min"/>
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="0" valuemax="30"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="200" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>

<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="19" unit="nb/min"/>
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="30" valuemax="60"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="200" unit="Mhz">
</property>
</preferenceEnvironment>
```

<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>

<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="13" unit="nb/min"/>
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="60" valuemax="100"/>
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="200" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="défaut"/>
</preferenceEnvironment>
</preference>
</TechnicalService>

5 P1S de la personnalité "transaction ONT" du service transactionnel

<TechnicalService function="transaction">
<property name="model" value="transaction_ONT"/>
<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="23.5" unit="nb/min">
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="0" valuemax="30">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="webservice">
</preferenceEnvironment>
</preference>
<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="21" unit="nb/s">
</preferenceTechnical>
<preferenceEnvironment>
<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="30" valuemax="60">
</property>
<preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="webservice">
</preferenceEnvironment>
</preference>
<preference>
<preferenceTechnical>
<property name="transaction_par_minute" value="13.5" unit="nb/s">
</preferenceTechnical>
<preferenceEnvironment>

<property name="utilisation_CPU" value="" unit="%">
<minmax valuemin="60" valuemax="100">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="frequence_CPU" value="300" unit="Mhz">
</property>
</preferenceEnvironment>
<preferenceEnvironment>
<property name="type application" value="webservice">
</preferenceEnvironment>
</preference>
</TechnicalService>

6 Exemple de description des besoins d'une application

<NeedOfApplication>
<TechnicalService function="transaction">
<property name="model" value="transaction_plate" composition="or"/>
<property name="model" value="transaction_ONT"/>
<preferences>
<preferenceTechnical>
<property name="type application" value="webservice"/>
</preferenceTechnical>
</preferences>
</TechnicalService>
<TechnicalService function="security"/>
</NeedOfApplication>

7 Détails de conception des composants de gestion

Cette annexe détaille la composition du composant "ManagementComponentBootstrap" qui contient l'ensemble des composants de gestion. Par rapport à la figure présentée dans la partie prototype, il existe un composant "Query Assistant" en plus, dont la fonction est simplement de transformer le format de la requête du contrat pour qu'il corresponde à celui de l'annuaire. Le détail du composant "Discovery Service" se trouve dans l'annexe suivante.

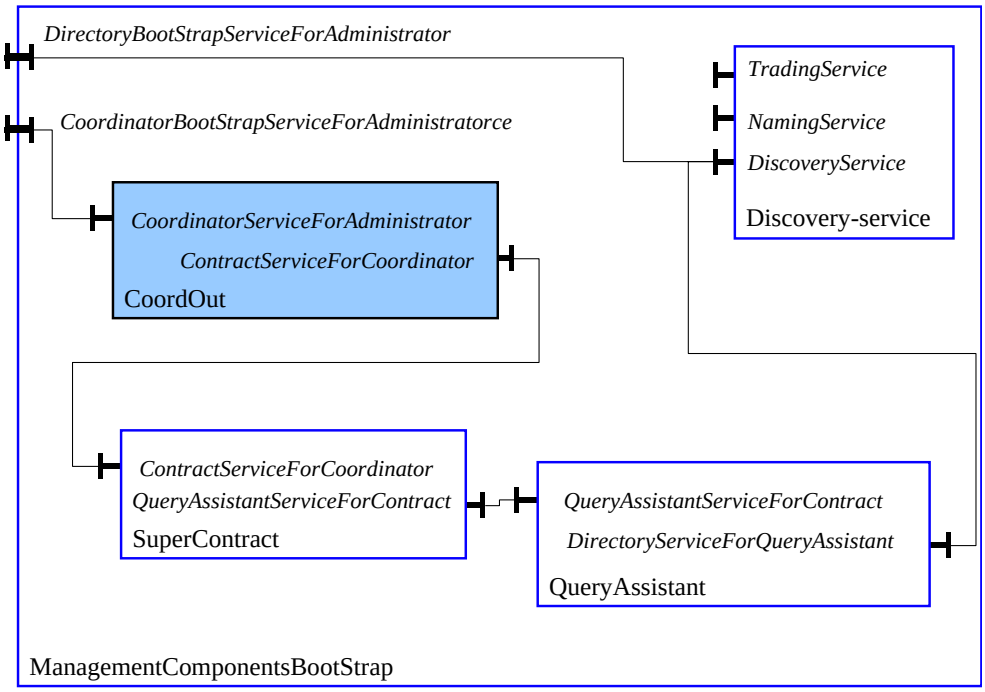

FIG. 1 – *Structure du composant ManagementComponentBootstrap*

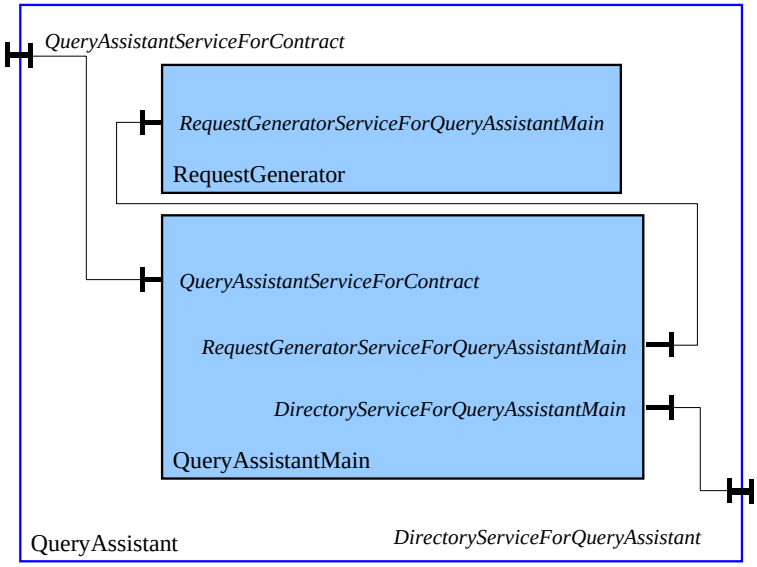

FIG. 2 – *Structure du composant Query Assistant*

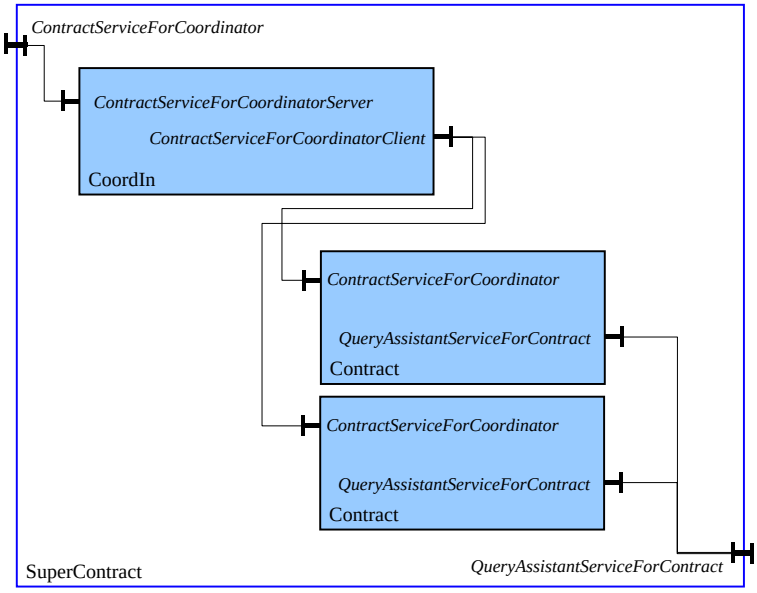

FIG. 3 – *Structure du composant SuperContract*

8 Détails de conception de l'annuaire

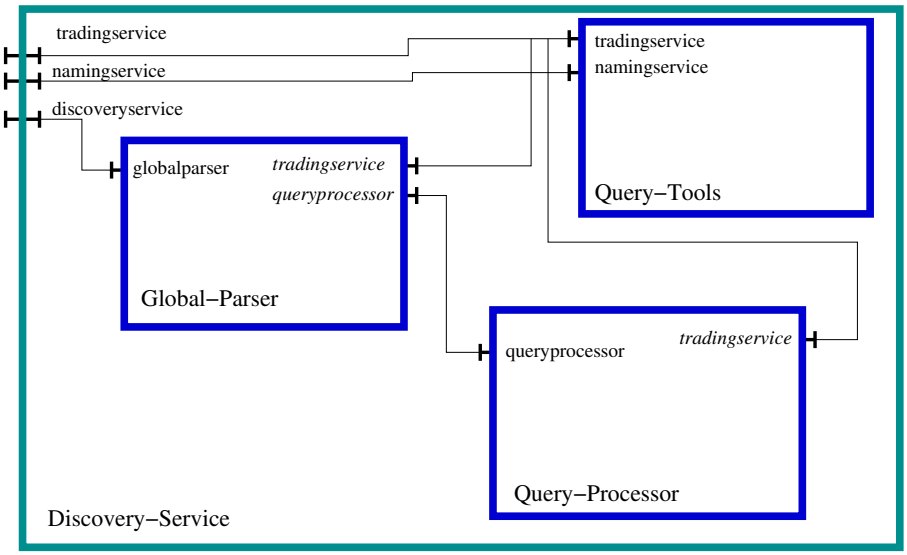

FIG. 4 – *Structure du composant Discovery Service*

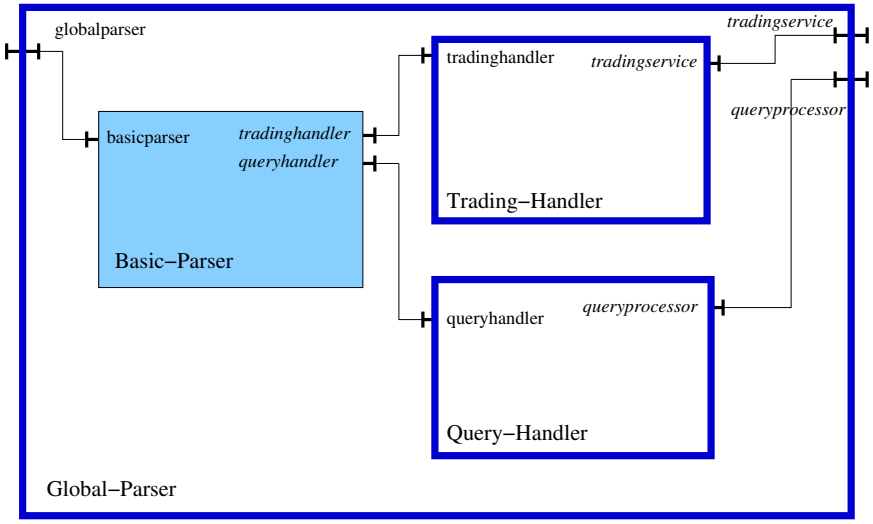

FIG. 5 – *Structure du composant Global Parser*

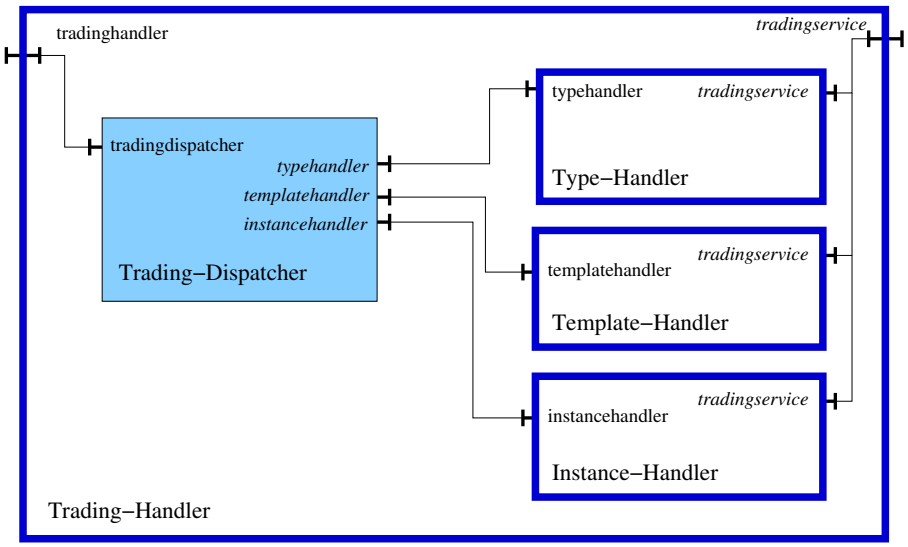

FIG. 6 – *Structure du composant Trading Handler*

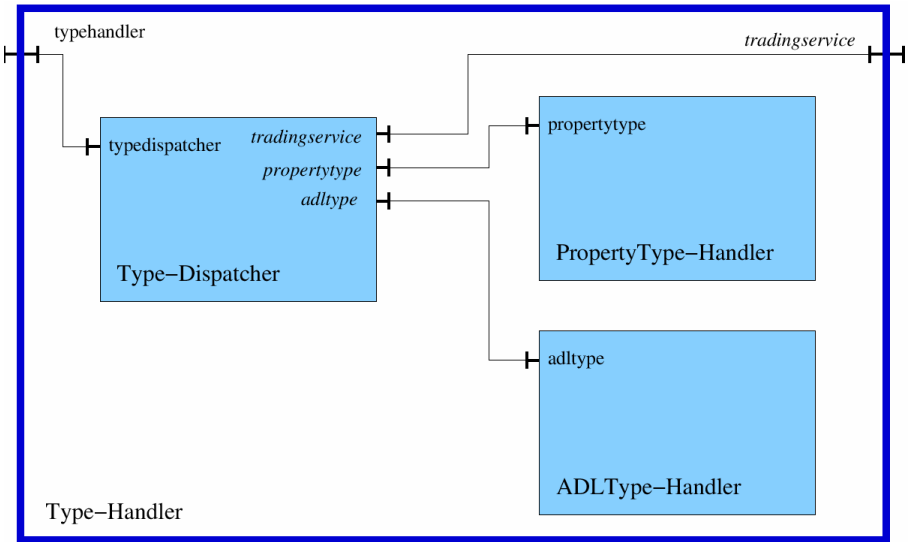

FIG. 7 – *Structure du composant Type Handler*

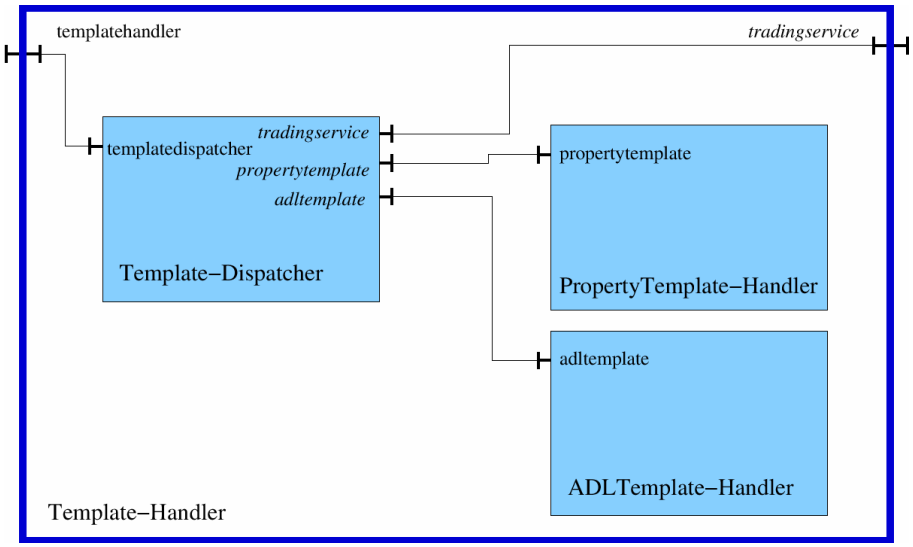

FIG. 8 – *Structure du composant Template Handler*

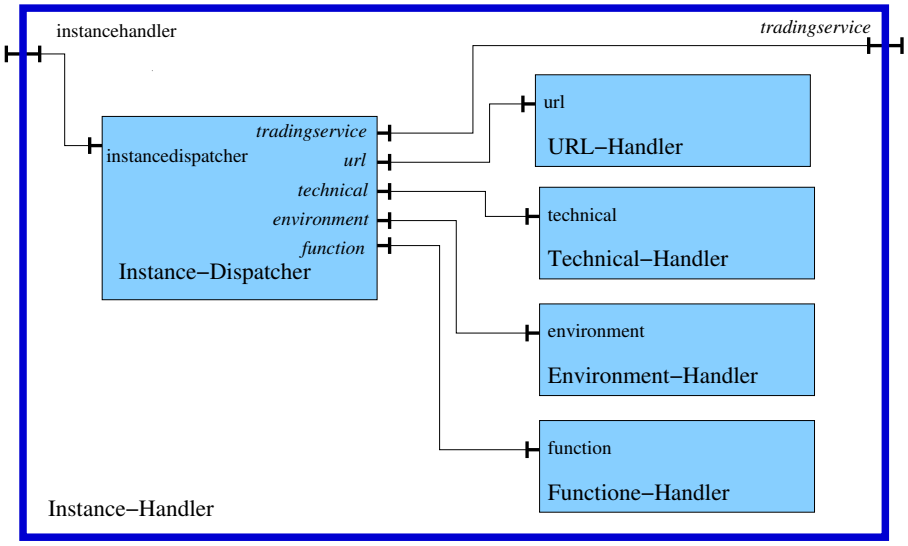

FIG. 9 – *Structure du composant Instance Handler*

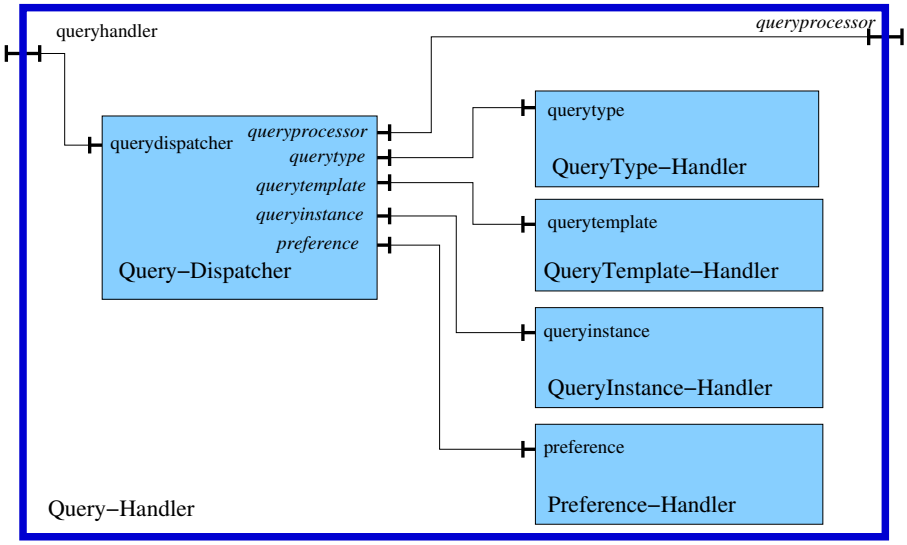

FIG. 10 – *Structure du composant Query Handler*

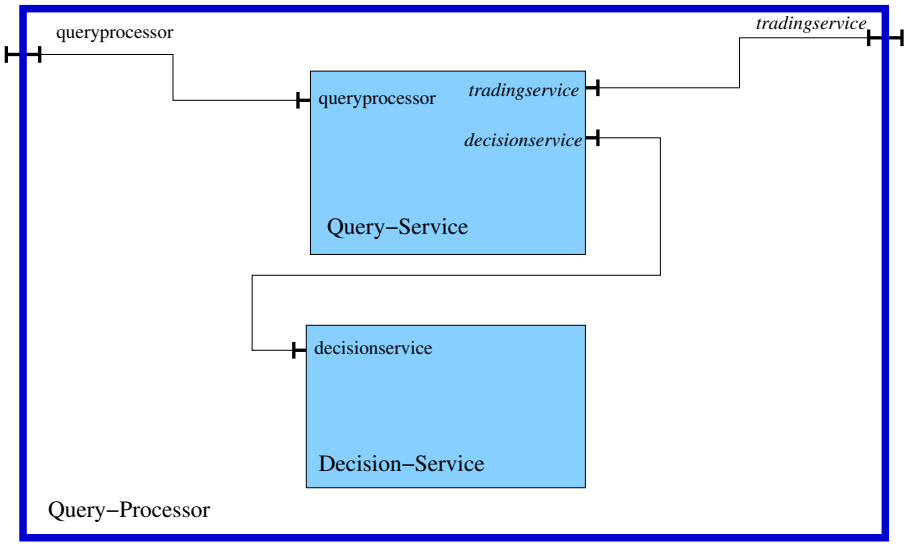

FIG. 11 – *Structure du composant Query Processor*

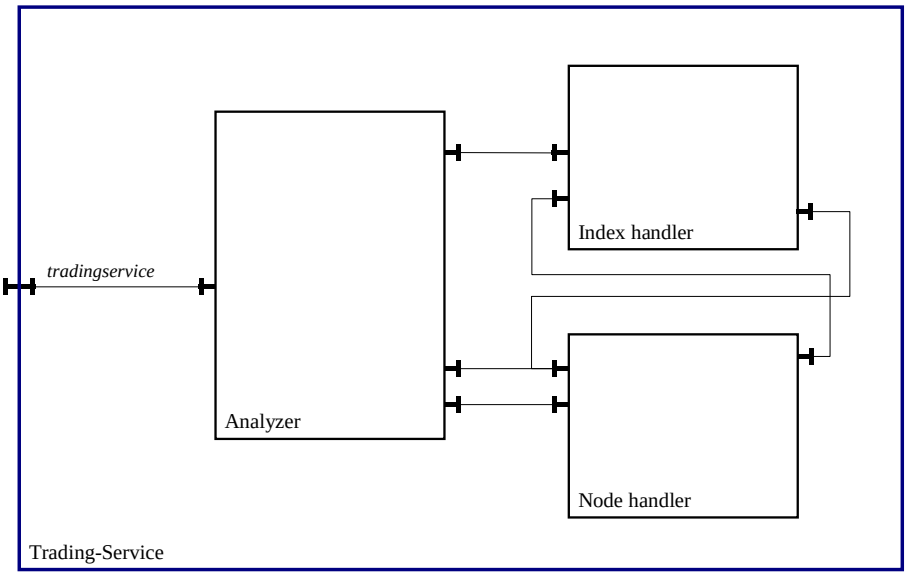

FIG. 12 – *Structure du composant Trading Service*

9 Composants et Liaisons dans le prototype

nom du composant	nombre de liaisons	sous-composants
ManagementComponentsBootStrap	5	1 composant primitif : *CoordOut* 2 composants composites *SuperContract* *QueryAssistant* *Discovery-Service*
SuperContract	1 + (2*Nb de contrats)	(1+Nb de contrats) composants primitifs *CoorIn* *N Contract*
QueryAssistant	3	2 composants primitifs *RequestGenerator* *QueryAssistantMain*
Discovery-Service	6	3 composants composites *Global-Parser* *Query-Processor* *Query-Tools*
Global-Parser	5	1 composant primitif *Basic-Parser* 2 composants composites *Trading-Handler* *Query-Handler*
Trading-Handler	7	1 composant primitif *Trading-Dispatcher* 3 composants composites *Type-Handler* *Template-Handler* *Instance-Handler*
Type-Handler	4	3 composants primitifs *Type-Dispatcher* *PropertyType-Handler* *ADLType-Handler*
Template-Handler	4	3 composants primitifs *Template-Dispatcher* *PropertyTemplate-Handler* *ADLTemplate-Handler*
Instance-Handler	6	5 composants primitifs *URL-Handler* *Technical-Handler* *Environment-Handler* *Function-Handler*

nom du composant	nombre de liaisons	sous-composants
Query-Handler	6	5 composants primitifs *Query-Dispatcher* *QueryType-Handler* *QueryTemplate-Handler* *QueryInstance-Handler* *Preference-Handler*
Query-Processor	3	2 composants primitifs *Decision-Service* *Query-Service*
Pour Query-Tools, nous ne rentrons pas dans le détail	47	32 composants primitifs 6 composants composites
TOTAL	97+ (2*Nb de contrats) liaisons	54 composants primitifs 17 composants composites

10 Détails des moniteurs

Système d'exploitation	Windows	Windows CE	Palm OS	Symbian OS
Émulateur		Pocket PC 2003 emulator	Palm OS emulator	Emulateur EPOC32 fournit avec le Symbian C++ SDK
Machine de test	PC	Pocket PC Ipaq Compaq	Tungsten T3	Non testé
Kit de développement		Embedded Visula C++	CodeWarrior	Symbian C++ Softwares Development Kit
Librairie informations système	oui	oui	oui	non
Librairie threads	oui	oui	Non, disponible version 6	oui
Librairie XLM	oui	oui	Payant	oui
Informations sur le système d'exploitation	oui	oui	non	
Informations sur le processeur	oui	oui	oui	
Informations sur la mémoire	oui	oui	oui	
Informations sur la batterie	non	oui	oui	
Informations sur la carte mémoire	non	non	oui	
Informations sur le réseau infrarouge	non	non	oui	
Informations sur le réseau bluetooth	non	non	oui	

FIG. 13 – *Comparaison d'implantation des différents moniteurs*

Bibliographie

[]

[AAG 01] Øyvind Aagedal J., « Quality of Support in Development of Distributed Systems », Thèse de doctorat, Faculty of Mathematics and Natural Sciences, Université d'Oslo, mars 2001.

[AKS 94] Aksit M., Bosch J., van der Sterren W.etBergmans L., « Real-Time Specification Inheritance Anomalies and Real-Time Filters », Dans *Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP'94), Lecture Notes in Computer Science, Vol. 821*, Bologne, Italie, juillet 1994, Springer-Verlag, pages 86-407.

[AND 01] Andersen A., Blair G., Goebel V., Karlsen R., Stabell-Kulø T.etYu W., « Arctic Beans: Configurable and Re-configurable Enterprise Component Architectures », Dans *Work in Progress session at ACM/IFIP/USENIX International Middleware Conference (Middleware'01)*, Heidelberg, Allemagne, novembre 2001.

[AND 02] Andersen A., « OOPP, A Reflective Middleware Platform including Quality of Service Management », Thèse de doctorat, Department of Computer Science, Université de Tromsø, février 2002.

[aop] « http://aopalliance.sourceforge.net/ ».

[AOS] « http://www.aosd.net/ ».

[Arc] « http://abean.cs.uit.no/ ».

[asp] « http://aspectwerkz.codehaus.org/ ».

[ava] « http://avalon.apache.org/ ».

[BAL 00] Balter R., Bernard G., Geib J.-M., Hagimont D., Jean S., Marangozova V., Marvie R., Merle P., Paradinas P., Pellegrini M., Potonniée O., Putrycz E., Taconet C.etVandewalle J., « Spécifications fonctionnelles », Rapport Technique numéroRNRT 98 CESURE-1, mai 2000, Projet CESURE.

[BAN 02] Banavar G.etBernstein A., « Software Infrastructure and Design Challenges for Ubiquitous Computing Applications », Dans *Communications of the ACM, Vol. 45, No. 12*, décembre 2002.

[BAR 02] Barais O.etDuchien L., « OPAD: Outils Pour Architectures Dynamiques », Dans *Systèmes à Composants Adaptables et Extensibles*, Grenoble, France, 2002.

[BAR 04] Barais O.etDuchien L., « SafArch : Maîtriser l'Evolution d'une Architecture Logicielle », Dans *Langages, Modèles, Objets - Journées Composants (LMO-JC'04)*, volume 10 de *L'objet*, Lille, France, mars 2004, Hermès Sciences, pages 103-116.

[BEN 02] Bennani N., Delot T., Lecomte S., Nemchenko S.etDonsez D., « Advanced Transactional Model for Component-Based Model », Dans *International Symposium on Advanced Distributed Systems (ISADS'02)*, 2002.

[BER 01] Bergmans L.etAksit M., « Composing crosscutting concerns using composition filters », Dans *Communications of ACM*, octobre 2001, pages 51-57.

[BES 97] Besancenot J., Cart M., Ferrié J., Guerraoui R., Pucheral P.etTraverson B., *Les Systèmes Transactionnels*, Hermès, 1997.

[BEU 99] Beugnard A., Jézéquel J.-M., Plouzeau N.etWatkins D., « Making Components Contract Aware », Dans IEEE [].

[BIR 94] Birman K. P.etvan Renesse R., « Reliable distributed computing with the Isis toolkit », Dans *IEEE Computer Society Press*, 1994.

[BLA 00] Blair G. S., Coulson G., Lynne Blair M. C., Costa F., Duran H., Parlavantzas N., Saikoski K.etAndersen A., « A Principled Approach to Supporting Adaptation in Distributed Mobile Environments », Dans *International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*, Limerick, Irlande, juin 2000, IEEE, pages 3-12.

[BLA 01] Blair G., Coulson G., Andersen A., Blair L., Clarke M., Costa F., Duran-Limon H., Fitzpatrick T., Johnston L., Moreira R., Parla-vantzas N.etSaikoski K., « The design and implementation of Open ORB 2 », Dans *IEEE Distrib. Syst. Online 2*, IEEE, septembre 2001.

[BON 04] Bonér J., « AspectWerkz - dynamic AOP for Java », Dans *International Conference on Aspect-oriented software development (AOSD 2004)*, 2004.

[BRO 02] Brose G.etNoffke N., « JacORB 1.4 documentation », Rapport Technique, août 2002, Freie Universität Berlin and Xtradyne Technologies AG.

[BRU 01a] Bruneton E., « Un support d'exécution pour l'adaptation des aspects non-fonctionnels des applications reparties », Thèse de doctorat, Institut national polytechnique de Grenoble, octobre 2001.

[BRU 01b] Bruneton E.etRiveill M., « Experiments with javapod, a platform designed for the adaptation of non-functional properties », Dans *Reflection 2001, LNCS 2192*, septembre 2001.

[BRU 02] Bruneton E., Coupaye T.etStefani J.-B., « Recursive and Dynamic Software Composition with Sharing », Dans *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02)*, Malaga, Espagne, 2002.

[BRU 04] Bruneton E., Coupaye T.etStefani J., « The Fractal Component Model », Rapport Technique numéro2.0-3, février 2004, The ObjectWeb Consortium.

[cae] « http://caesarj.org/ ».

[CCM99] « CORBA Components. Specification. OMG TC Document orbos/99-02-05 », Rapport Technique, 1999, Object Management Group.

[CCM02] « CORBA Components 3.0 - The Container Programming Model chapter », Rapport Technique numéro formal/02-06-72, juin 2002, Object Management Group (OMG).

[CHE 02] Chefrour D.etAndré F., « ACEEL: modèle de composants auto-

adaptatifs », Dans *Systèmes à composants adaptables et extensibles*, Grenoble, France, 2002.

[CHI 95] Chiba S., « A metaobject protocol for C++ », Dans *10th Conference on Object-ORiented Programming Systems, Languages, and Applications (OOPSLA'95)*, Austin, Texas, USA, octobre 1995, pages 285-299.

[CHI 00] Chiba S., « Load-Time Structural Reflection in Java », Dans *14th European Conference on Object-Oriented Programming (ECOOP 2000)*, Sophia Antipolis et Cannes, France, juin 2000, pages 313-336.

[CHI 02] Chiquet A., Donsez D., Estublier J., Hérault C., Leblanc S., Lecomte S., Merle P., Potonniée O.etVessière T., « Architecture Générale », Rapport Technique, 2002, Projet RNRT COMPiTV.

[Cod] « http://www.metrowerks.com/mw/default.htm ».

[COL 05] Collet P.etOzanne A., « Un système de contractualisation pour Fractal: intégration et retour sur expérience », Dans *à paraître dans Journées Composants 2005, 4ème Conférence Francophone autour des Composants Logiciels*, Le Croisic, Presqu'île de Guérande, France, avril 2005.

[Coma] « http://trese.cs.utwente.nl/ ».

[COMb] « http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_01_47.htm ».

[COR95] « Common Object Request Broker Archictecture », Rapport Technique, juillet 1995, Object Management Group (OMG).

[COU 02] Courbis C., Degenne P., Fau A., Parigot D.etVariamparambil J., « Un modèle de Composants pour l'Atelier de Développement SmartTools », Dans *Systèmes à composants adaptables et extensibles*, Grenoble, France, 2002.

[COU 04a] Coulson G., Blair G., Parlavantzas N., Yeung W. K.etCai W., « Applying the Reflective Middleware Approach in Grid Computing », Dans *Concurrency and Computation: Practice and Experience, Vol 16, No 5*, avril 2004, pages 433-440.

[COU 04b] Courbis C., Degenne P., Fau A.etParigot D., « Un modèle abstrait de composants adaptables », Dans *revue TSI, Composants et adaptabilité, 23(2)*, 2004.

[CZA 00] Czarnecki K., U. Eisenecker Methods T.etApplications, *Generative programming*, Addison Wesley, 2000.

[DAV 03] David P.-C.etLedoux T., « Towards a Framework for Self-Adaptive Component-Based Applications », Dans Stefani J.-B., Demeure I.etHagimont D., éditeurs, *Distributed Applications and Interoperable Systems 2003, the 4th IFIP WG6.1 International Conference, DAIS 2003*, volume 2893 de *Lecture Notes in Computer Science*, Paris, novembre 2003, Federated Conferences, Springer-Verlag, pages 1-14.

[DAV 04a] David F., *Model Driven Architecture*, John Wiley and Sons Ltd, janvier 2004.

[DAV 04b] David P.-C.etLedoux T., « Pour un aspect d'adaptation dans le développement d'applications à base de composants », Dans *Actes Journée de l'AS 150, Systèmes répartis et réseaux adaptatifs au contexte*, Paris, France, avril 2004.

[DCE] « http://www.opengroup.org/dce/ ».

[DEM 03] DeMichiel L. G., « Enterprise JavaBeansTM Specification, Version 2.1 », Rapport Technique, novembre 2003, Sun Microsystem.

[DEM 04] Demarey C., Harbonnier G., Rouvoy R.etMerle P., « Benchmarking the Round-Trip Latency of Various Java-Based Middleware Platforms », Dans *The OOPSLA 2004 Component And Middleware Performance Workshop, CMP 2004*, Vancouver, Canada, octobre 2004, Studia Informatica.

[DIJ 76] Dijkstra E. W., *A Discipline of Programming*, Prentice Hall PTR, 1976.

[DJ] « http://www.ccs.neu.edu/research/demeter/DJ/ ».

[DOW 04] Dowling J.etCahill V., « Self-Managed Decentralised Systems using K-Components and Collaborative Reinforcement Learning », Dans *Proceedings of the Workshop on Self-Managed Systems (WOSS'04)*, 2004.

[dso] « http://dsonline.computer.org/middleware/ ».

[DUC 02a] Duclos F., « Environnement de Gestion de Services Non-Fonctionnels dans les Applications à Composants », Thèse de doctorat, Université Joseph Fourier de Grenoble, 2002.

[DUC 02b] Duclos F., Estublier J.etMorat P., « Describing and using non functional aspects in component based applications », Dans *Proceedings of the 1st international conference on Aspect-oriented software development*, Enschede, Pays Bas, 2002, ACM Press, pages 65–5.

[DUR 04] Duran-Limon H. A., Blair G. S.etCoulson G., « Adaptive Resource Management in Middleware: A Survey », IEEE Distributed Systems Online http://dsonline.computer.org, juillet 2004.

[eao] « http://www.emn.fr/x-info/eaop/ ».

[EJB01] « Enterprise JavaBeans Specification. Version 2.1 », Rapport Technique, 2001, Sun MicroSystems.

[FAK 04] Fakih H., Bouraqadi N.etDuchien L., « Aspects and Software Components : a case study of the Fractal Component Model », Dans *International Workshop of Aspect-Oriented Software Development*, Beijing, Chine, septembre 2004.

[Fra] « http://fractal.objectweb.org/ ».

[GAM 95] Gamma E., Helm R., Johnson R.etVlissides J., *Design patterns, catalogue de modèles de conception réutilisables*, Internationl Thomson Publishing France, 1995.

[GEI 98] Geier M., Steckermeier M., Becker U., Hauck F. J., Meier E.etRastofer U., « Support for mobility and replication in the AspectIX architecture », Rapport Technique numéro TR-I4-98-05, 1998, Université de Erlangen-Nuernberg, IMMD IV.

[GEI 01] Geihs K., « Middleware Challenges Ahead », Dans *IEEE Computer Magazine, 34(6)*, volume 34, 2001, pages 24-31.

[gra] « http://www.granddictionnaire.com/ ».

[GRA 81] Gray J., « The Transaction Concept: Virtues and Limitations », Dans *Proceedings of the VLDB Conference*, 1981.

[GRI] Griswold B., Hilsdale E., Hugunin J., Isberg W., Kiczales G.etKersten M., « Aspect-Oriented Programming with AspectJ », Dans *AspectJ.org, Xerox PARC, Tutoriel, disponible à l'adresse http://eclipse.org/aspectj/.*

[GRI 04] Grine H., « Localisation de Services Non-Fonctionnels dans un Modèle à Composants », rapport de Master Recherche 2 AISIH, juillet 2004, ISTV - Université de Valenciennes et du Hainaut Cambrésis.

[GRI 05] Grine H., Hérault C., Lecomte S.etDelot T., « Localisation de services non-fonctionnels dans un modèle à composants », Dans *à paraître dans Journées Composants 2005, 4ème Conférence Francophone autour des Composants Logiciels*, Le Croisic, Presqu'île de Guérande, France, avril 2005.

[HAR 93] Harrison W.etOssher H., « Subject-oriented programming (a critique of pure objects) », Dans *OOPSLA 93*, 1993.

[HAY 97] Hayton R., « FlexiNet Open ORB Framework. », Rapport Technique numéro 2047.01.00, 1997, APM ltd, Poseidon House, Castle Park, Cambridge, UK.

[HEI 01] Heineman G. T.etCouncil W. T., *Component-Based Software Engineering, Putting the Pieces Together*, Addison Weysley, 2001.

[HéR 01] Hérault C., « EJBiTV - Plate-forme à composants serveurs de type EJB dans le contexte de la TV interactive », rapport de DEA, juillet 2001, ISTV - Université de Valenciennes et du Hainaut Cambrésis.

[HéR 02] Hérault C., Bennani N., Delot T., Lecomte S.etThilliez M., « Adaptability of Non-Functional Services for Component Model, Application to the M-Commerce », Dans *IEEE International Symposium on Advanced Distributed Computing (ISADS)*, Guadalajara, Jalisco, Mexique, novembre 2002, pages 93-104.

[HéR 03a] Hérault C.etLecomte S., « Adaptabilité des Services Techniques dans un Modèle à Composants », Dans *3ème Conférence Française sur les Systèmes d'Exploitation (CFSE)*, La Colle sur Loup, France, octobre 2003, pages 488-499.

[HéR 03b] Hérault C.etLecomte S., « Gestion de Personnalités de Services Non-Fonctionnels dans un Modèle à Composants », Dans *Workshop "Objets, Composants et Modèles dans l'ingénierie des Systèmes d'Information" en commun avec le congrès INFORSID 2003*, Nancy, France, juin 2003.

[HéR 04a] Hérault C.etLecomte S., « Gestion de Personnalités de Services Techniques dans un Modèle à Composants », Dans *publié sous forme de note de recherche du LAMIH n° 14/2004 LAMIH / ROI Soumis le 5 mars 2004 à la Revue L'Objet*, 2004.

[HéR 04b] Hérault C., Lecomte S.etDelot T., « New technical services using the component model for applications in heterogeneous environment », Dans *Innovative Internet Community Systems (I2CS)*, Guadalajara, Jaliso, Mexique, juin 2004.

[HéR 04c] Hérault C., Nemchenko S.etLecomte S., « A Component-Based Transactional Service, Including Advanced Transactional Models », Dans *Fourth IEEE International Symposium and School on Advance Distributed Systems (ISSADS 04)*, Guadalajara, Jalisco, Mexique, janvier 2004.

[HUN 99] Hunt G.etBrubacher D., « Detour : Binary Interception of Win32 Functions », Dans *3rd USENIX Window NT Symposium*, Seattle, Washington, USA, juillet 1999.

[IBM] « http://www-306.ibm.com/software/integration/mqfamily/ ».

[ISO 95a] ISO/IEC, « Open distributed processing reference model, part 1: Overview », Rapport Technique numéroITU-T Rec. X.901 | ISO/IEC 10746-1, 1995, ISO/IEC.

[ISO 95b] ISO/IEC, « QoS - Basic Framework, ISO, Report: ISO/IEC JTC1/SC 21 N9309 », Rapport Technique, 1995, ISO.

[ISO 99a] ISO/IEC, « Information Technology - Software product quality - Part 1: Quality model », Rapport Technique numéroISO/IEC 9126-1, 1999, ISO/IEC.

[ISO 99b] ISO/IEC, « Information Technology - Software product quality - Part 2 External Metrics », Rapport Technique numéroISO/IEC 9126-2, 1999, ISO/IEC.

[ISO 99c] ISO/IEC, « Information Technology - Software product quality - Part 3 Internal Metrics », Rapport Technique numéroISO/IEC 9126-3, 1999, ISO/IEC.

[ISO 00] ISO/IEC, « Information Technology - Software product quality - Part 4 Quality In Use Metrics », Rapport Technique numéroISO/IEC 9126-4, 2000, ISO/IEC.

[ist] « http://www.ist-coach.org/ ».

[JAJ 97] Jajodia S.etKerschberg L., « Advanced Transaction Models and Architecture », Dans *Kluwer Academic Publishers*, 1997.

[jav] « http://java.sun.com/products/ejb/ ».

[JBoa] « http://www.jboss.org/ ».

[JBob] « http://www.jboss.org/developpers/projects/jboss/aop/ ».

[jir] « http://jotm.objectweb.org/jironde.html/ ».

[jma] « http://roots.iai.uni-bonn.de/research/jmangler/ ».

[jon] « http://jonas.objectweb.org/ ».

[JOT] « http://jotm.objectweb.org ».

[KAD 04] Kaddour M.etPautet L., « Une approche coopérative des applications mobiles basées sur MobileJMS », Dans *Premières journées francophones sur Mobilité et Ubiquité*, Nice, France, juin 2004.

[KAR 03] Karlsen R., « An Adaptive Transactional System - framework and service synchronization », Dans *The International Symposium on Distributed Objects and Applications, DOA 2003)*, Catania, Sicile, Italie, novembre 2003.

[KIC 91] Kiczales G., des Rivieres J.etBobrow D., « The Art of the Metaobject Protocol », Dans *Cambridge, MIT Press*, 1991.

[KIC 97a] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C. V., Loingtier J.-M.etIrwin J., « Aspect-Oriented Programming », Dans *Proceedings of ECOOP*, Springer-Verlag, 1997.

[KIC 97b] Kiczales G., Lamping J., Menhdhekar A., Maeda C., Lopes C., Loingtier J.-M.etIrwin J., « Aspect-Oriented Programming », Dans Akşit M.etMatsuoka S., éditeurs, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[KIC 01] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J.etGriswold W., « Getting started with ASPECTJ », *Communications of the ACM*, volume 44, numéro 10, 2001, pages 59-65, ACM Press.

[KIN 02] Kinzley J.etGuerraoui R., « AOP does it make sense? The Case of Concurrency and Failures », Dans *European Conference on Object-Oriented Programming (ECOOP 2002)*, Malaga, juin 2002.

[KLE 02] Klefstad R., Schmidt D. C., etRyan C. O., « Towards highly configurable real-time object request brokers », Dans *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, avril-mai 2002.

[KON 00] Kon F., Roman M., Liu P., Mao J., Yamane T., Magalhaes L.etCampbell R., « Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB », Dans *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open DistributedProcessing (Middleware2000)*, Palisades, NY, avril 2000, Springer-Verlag, pages 121-143.

[KON 02] Kon F., Costa F., Blair G.etCampbell R. H., « The Case for Reflective Middleware », volume 45, 2002, pages 33-38.

[KOS 01] Koster R., Black A. P., Huang J., Walpole J.etPu C., « Thread transparency in information flow middleware », Dans *Proceedings of the International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer Verlag, novembre 2001.

[LED 99] Ledoux T., « OpenCORBA: A reflective open broker », Dans *Proceedings of Reflection'99*, Saint-Malo, France, juillet 1999, Springer-Verlag, pages 197-214.

[LEN 04] Lenglet R., « Composition flexible et efficace de transformations de programmes », Thèse de doctorat, France Telecom R&D DTL/ASR, LSR / IMAG / INRIA Rhône-Alpes, INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE - INPG, novembre 2004.

[LES 03] Le Sommer N., « Contractualisation des ressources pour les composants logiciels : une approche réflexive », Thèse de doctorat, Université de Bretagne-Sud, décembre 2003.

[LIE 94] Lieberherr K. J., Silva-Lepe I.etXiao C., « Adaptive object-oriented programming using graph-based customization », *Communications of the ACM*, volume 37, numéro 5, 1994, pages 94–101.

[LIE 95] Lieberherr K. J., *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*, PWS Publishing Co., 1995.

[LOP 95] Lopes C.etHursch W., « Separation of Concerns », Rapport Technique numéroNU-CCS-95-03, février 1995, College of Computer Science, Northeastern University, Boston, MA, USA.

[LUD 00] Ludwig A.etHeuzeroth D., « Metaprogramming in the Large », Dans *2nd International Conference on Generaive And Component-based Software Engineering (GCSE)*, janvier 2000.

[M.A 93] M. Aksit K. Wakita J. B. L. B. A. Y., « Abstracting Object Interactions Using Composition-Filters », Dans *Object-Based Distributed Processing, sous la direction de R. Guerraoui, O. Nierstrasz and M. Riveill*, Springer-Verlag, 1993, pages 152-184.

[MAE 87] Maes P., « Concepts and experiments in computational reflection », Dans *Conference proceedings on Object-oriented programming systems, languages and applications*, ACM Press, 1987, pages 147-155.

[MAF 95] Maffeis S., « Adding group communication and fault-tolerance to CORBA », Dans *Proceedings of the Conference on Object-Oriented Technologies*, 1995, pages 135-146.

[MAS 92] Masuhara H., Matsuoka S., Watanabe T.etYonezawa A., « Object-oriented concurrent reflective languages can be implemented efficiently », Dans *conference proceedings on Object-oriented programming systems, languages, and applications*, ACM Press, 1992, pages 127-144.

[MCK 04] McKinley P. K., Sadjadi S. M., Kasten E. P.etCheng B. H. C., « A Taxonomy of Compositional Adaptation », Rapport Technique numéroMSU-CSE-04-17, juillet 2004, Department of Computer Science and Engineering, Michigan State University, USA.

[MED 00] Medvidovic N.etTaylor R. N., « A Classification and Comparison Framework for Software Architecture Description Languages », Dans *IEEE Transactions on Software Engineering, vol. 26, no. 1*, janvier 2000, pages 70-93.

[MEY 92] Meyer B., « Applying "Design by Contract" », Dans *Computer (IEEE), vol. 25, no. 10*, octobre 1992, pages 40-51.

[MEZ 98] Mezini M.etLieberherr K. J., « Adaptive Plug-and-Play Components for Evolutionary Software Development », Dans *Conference on Object-Oriented*, 1998, pages 97-116.

[MEZ 00] Mezini M., Seiter L.etLieberherr K., « Component Integration with Pluggable Composite Adapters », Dans Aksit M., éditeur, *Software Architectures and Component Technology: The State of the Art in Research and Practice*, Kluwer, 2000, Université de Twente, Pays Bas.

[MON 01] Monson-Haefel R., *Enterprise JavaBeans, 3rd Edition*, O'Reilly, septembre 2001.

[MOS 81] Moss E., « Nested Transactions: An approach to reliable Distributed Computing », Thèse de doctorat, MIT, 1981.

[NEM 04] Nemchenko S., « Modèle de transactions avancées et modèle à composants », Thèse de doctorat, Université de Valenciennes et du Hainaut Cambrésis, septembre 2004.

[OLI 99] Oliva A.etBuzato L. E., « The Desing and Implementation of Guarana », Dans *5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99)*, San Diego, California, USA, mai 1999.

[OMG 00] OMG O. M. G., « CORBA Trading Object Service specification », 2000.

[Opea] « http://openccm.objectweb.org/ ».

[Opeb] « http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/opencom.php ».

[ORBa] « http://www.orbacus.com/ ».

[Orbb] « URL: http://www.iona.com/products/orbix.htm ».

[ORB04] « Orbacus White Paper », Rapport Technique, juin 2004, Iona.

[ORF 99] Orfali R., Harkey D.etEdwards J., *ClientServeur Survival Guide*, 3rd édition, Vuibert, 1999.

[ORL 01] Orleans D.etLieberherr K., « DJ: Dynamic Adaptive Programming in Java », Dans *Reflection 2001: Meta-level Architectures and Separation of Crosscutting Concerns* , Kyoto, Japan, septembre 2001, Springer Verlag, 8 pages.

[OTS03] « Transaction Service Specification, Version 1.4 », Dans *CORBA Services*, OMG, 2003.

[PAR 72] Parnas D. L., « On the criteria to be used in decomposing systems into modules », *Communications of the ACM*, volume 15, numéro 12, 1972, pages 1053–1058, ACM Press.

[PAW 03] Pawlak R., Seinturier L.etDuchien L., « Jac Milestone 2003 », Rapport Technique numéro2003-4, 2003, LIFL, Université des Sciences et Technologies de Lille.

[PAW 04] Pawlak R., Retaillé J.-P.etSeinturier L., *Programmation orientée aspect pour Java/J2EE*, Editions Eyrolles, 2004.

[PEL 00] Pellegrini M.-C., Potonniée O., Marvie R., Jean S.etRiveill M., « CESURE : une plate-forme d'applications adaptables et sécurisées pour usagers mobiles », Dans *Special Issue of the french journal Calculateurs Parallèles, entitled "Évolution des plates-formes orientées objets répartis"*, Hermès, 2000, pages 113-120.

[PES 04] Pessemier N., Seinturier L., Duchien L.etBarais O., « Une extension de Fractal pour l'AOP », Dans *Première journée Francophone sur le Développement de Logiciels par Aspects, JFDLPA 2004*, Paris, France, septembre 2004.

[PET 03] Petit D., « Génération automatique de composants logiciels sûrs à partir de spécifications formelles B. », Thèse de doctorat, Université de Valenciennes et du Hainaut Cambrésis, décembre 2003.

[Pro] « http://prose.ethz.ch/ ».

[PRO 03a] Prochazka M., « A Flexible Framework for Adding Transactions to Components », Dans *The Eighth International Workshop on Component-Oriented Programming (WCOP 2003, in conjunction with ECOOP 2003)*, Darmstadt, Allemagne, juillet 2003.

[PRO 03b] Prochazka M., « Jironde: A Flexible Framework for to Make Components Transactional », Dans *4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2003)*, Paris, France, novembre 2003.

[rea] « http://www.realtime-corba.com/ ».

[REN 95] Renesse R. V., Birman K. P., Glade B. B., Guo K., Hayden M., Hickey T., Malki D., Vaysburd A.etVogels W., « Horus: A flexible group communications system », Rapport Technique numéroTR95-1500, 1995, Department of Computer Science, Cornell University.

[RIV 00] Riveill M.etMerle P., « Programmation par composants », Rapport Technique numéroH 2 759, novembre 2000, Techniques de l'ingénieur, traité Informatique.

[ROM 00] Roman M., Mickunas M., Kon F.etCampbell R. H., « LegORB and ubiquitous CORBA », Dans *IFIP/ACM Middleware 2000 Workshop on Reflective Middleware (RM2000)*, New York, avril 2000.

[ROM 01] Roman M., Kon F.etCampbell R. H., « Reflective middleware: From your desk to your hand », Dans *IEEE Distributed Systems Online, vol. 2, no. 5*, 2001.

[ROU 03] Rouvoy R.etMerle P., « Abstraction of Transaction Demarcation in Component-Oriented Platforms », Dans *ACM/IFIP/USENIX International Middleware Conference (Middleware'03)*, Rio de Janeiro, Brésil, juin 2003.

[SAJ] « http://www-valoria.univ-ubs.fr/Composants/CASA/ ».

[SCH 98] Schmidt D. C., Levine D. L.etMungee S., « The design of the TAO real-

time object request broker », Dans *Computer Communications, vol. 21*, avril 1998, pages 294-324.

[SCH 01] Schantz R.etSchmidt D., « Evolving the Common Structure for Network-centric Applications », Dans *Encyclopedia of Software Engineering, chapter Middleware for Distributed Systems*, Wiley & Sons, 2001.

[SIM 95] Simonyi C., « The Death of Computer Languages, the Birth of Intentional Programming », Dans *The Future of Software, Univ. of Newcastle upon Tyne, England, Dept. of Computing Science*, 1995.

[sma] « http://www-sop.inria.fr/smartool/SmartTools/ ».

[SMI 84] Smith B. C., « Reflection and semantics in LISP », Dans *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, Salt Lake City, Utah, USA, 1984, pages 23-35.

[SZY 02] Szyperski C., *(with Dominik Gruntz and Stephan Murer) Component Software - Beyond Object-Oriented Programming - Second Edition*, Addison-Wesley / ACM Press, 2002.

[TAO] « http://www.wustl.edu/schmidt/TAO.html ».

[TAT 99] Tatsubori M., Chiba S., Killijian M.-O.etItano K., « OpenJava : A Class-Based Macro System for Java », Dans *1st OOPSLA Workshop on Reflection and Software Engineering (OORaSE 1999*, volume 1826 de Lecture Notes in Computer Science, Denver, Colorado, USA, novembre 1999, pages 117-133.

[TER 00] Terzis S.etNixon P., « Towards a Semantically Enhanced Component Trader Architecture », Rapport Technique numéro TCD-CS-2000-23, 2000, Computer Science Department, Trinity College.

[THA 01] Thai T.etLam H., *.NET Framework Essentials*, O'Reilly and Associates, 2001.

[THI 03] Thilliez M., Delot T., Lecomte S.etBennani N., « Hybrid Peer-To-Peer Model in Proximity Applications », Dans *IEEE AINA 2003*, Xi Ang, Chine, 2003.

[TOU 05] Tournier J.-C., Babau J.-P.etOlive V., « Qinna, a Component-Based QoS Architecture », Dans *Component-Based Software Engineering: 8th International Symposium (CBSE 2005)*, St. Louis, MO, USA, mai 2005, Springer-Verlag, page 107.

[TRA 04] Traverson B., « Assemblage de composants par contrats (Le projet ACCORD) », Dans *Workshop Objets Composants et Modèles dans l'ingénierie des SI (OCM-SI)* [].

[VAD 01] Vadet M.etMerle P., « Les conteneurs ouverts dans les plates-formes à composants », Dans *Actes de la Journée Thème Émergent Composants*, Besançon, France, 2001.

[VEN 02] Venkatasubramanian N., « Safe 'Composability' of Middleware Services », volume 45, 2002, pages 49-52.

[WAN 01] Wang N., Schmidt D. C., Kircher M.etParameswaran K., « Adaptive and Reflective Middleware for QoS-enabled CCM Applications », volume 2, 2001.

[WAN 03] Wang N., Schmidt D. C.etKircher M., « Towards an adaptive and reflective middleware framework for QoSenabled CORBA component model applications », Dans *IEEE Distributed System Online, Special Issue on Reflective Middleware*, 2003.

Résumé

Le développement des applications distribuées, amenées de plus en plus à s'exécuter sur des machines aux caractéristiques hétérogènes, tend à se complexifier. Le modèle à composants apporte une réponse à ce problème en favorisant la réutilisabilité du code. Par ailleurs, les services techniques, offerts par les plates-formes à composants, allègent le code source et améliorent la qualité de service. Cependant, ces services techniques, conçus pour une exécution statique sur des machines serveurs, ne s'adaptent pas à d'autres contextes d'exécution. Leur développement sous forme d'objet notoire empêche leur adaptation. De plus, l'ensemble des services techniques fournis par la plate-forme à composants est figé. La première contribution de cette thèse consiste à définir un modèle de conception et de développement des services techniques basé sur le modèle à composants Fractal. Ce processus aboutit à la définition de personnalités d'un service (P1S) répondant chacune à un contexte d'utilisation, ainsi qu'à une redéfinition de l'assemblage formé par l'application et ses services techniques. Notre seconde contribution est un service de gestion de l'adaptation des services techniques qui permet de détecter les changements d'environnement d'exécution grâce à un moniteur, de localiser les services techniques et d'identifier la personnalité la plus adaptée au nouvel environnement grâce à un annuaire dédié aux services techniques. La faisabilité de nos propositions a été validée à travers la conception et l'implantation d'un service transactionnel multi-personnalités ainsi que l'implantation du système de gestion de l'adaptation.

Mots-clés: services techniques, adaptabilité, modèle à base de composants, intergiciel.

Abstract

New distributed applications are increasingly executed on devices that have heterogeneous characteristics. Thus, their development becomes more complex. The component-based model is one possible response to increase the reusability of code. Technical services, provided by component based platforms, lighten the source code and increase the quality of service. Thought, technical services, which are designed as object with initial references in order to be statically executed on servers, do not adapt to other execution contexts. Moreover the set of technical services is fixed. The first contribution of this thesis is a definition of a conception and development model for technical services, which is based on the Fractal component model. This process comes to the definition of personalities of a service (P1S), each of those personalities matching to a particular usage context, and to a redefinition of the assembly constituted by the application and the technical services. Our second contribution is a service dedicated to the management of the adaptation of the technical services. This service incorporates a monitor that allows to detect the modifications of the execution environment, and a directory service, dedicated to technical services, that helps localizing and identifying personalities that are the most adapted to the new execution environment. The feasibility of our proposals has been validated through the conception and implementation of a multi-personality transactional and the implementation of the management system for adaptation.

Keywords: technical services, adaptability, component-based model, middleware.