Utilisation d'agents mobiles pour la construction de services distribués

Siegfried Rouvrais

N° d'ordre: 2518

THÈSE

présentée

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Siegfried Rouvrais

Équipe d'accueil : IRISA
École Doctorale : MATISSE
Composante universitaire : IFSIC

Titre de la thèse :

Utilisation d'agents mobiles pour la construction de services distribués

soutenue le 5 juillet 2002 devant la commission d'Examen

COMPOSITION DU JURY :

M.	Jean-Pierre	Banâtre	Président
MM.	Guy	Bernard	Rapporteurs
	Marie-Pierre	Gervais	
MM.	Ciarán	Bryce	Examinateurs
	Pascal	Fradet	
	Valérie	Issarny	

Remerciements

Je suis très reconnaissant à Jean-Pierre Banâtre, professeur à l'Université de Rennes 1, d'avoir accepté le rôle de président du Jury. Les enseignements qu'il dispense à l'IFSIC ont contribué à me faire suivre le chemin de la recherche et de l'enseignement.

Je remercie Guy Bernard, professeur à l'Institut National des Télécoms d'Evry, et Marie-Pierre Gervais, professeur à l'Université de Paris X, d'avoir rapporté ce document de thèse. L'intérêt que Mr Bernard a porté à ce travail m'a conforté dans la poursuite d'une carrière d'enseignant-chercheur. Les remarques pertinentes de Mme Gervais, sur une première version de ce document, m'ont permis d'en améliorer la clarté.

Je remercie également Ciarán Bryce, maître de conférences à l'Université de Genève, d'avoir bien voulu juger ce travail.

Je remercie chaleureusement mes encadreurs de thèse pour leur sympathie, leur soutien, leur conseil et leur disponibilité... parfois leur patience, qui m'ont permis d'avancer tout au long de ce travail. J'ai beaucoup appris à leurs côtés.

Merci donc à Pascal Fradet, chargé de recherche à l'IRISA, encadreur de cette thèse et de mon stage de DEA, particulièrement pour sa rigueur scientifique et pour le temps qu'il m'a consacré.

Merci à Valérie Issarny, directrice de recherche à l'INRIA-Rocquencourt, qui a dirigé cette thèse, particulièrement pour son expérience, son dynamisme et ses collaborations qui m'ont été très profitables.

En plus des personnes avec lesquelles j'ai pu coopérer sur ce travail, je remercie celles qui ont assisté à la soutenance.

Merci à mes parents, proches et amis, pour leur soutien; certains pour le goût de la connaissance et du travail bien accompli qu'ils ont su me transmettre, d'autres, pour leur joie de vivre et les loisirs qu'ils ont su partager. Finalement, merci à Marilou pour sa présence de tous les instants.

Introduction

Le partage des ressources et de l'information sur un réseau est la principale motivation pour construire et utiliser des systèmes distribués [CDK01]. Ces systèmes s'appuient sur plusieurs types de réseaux de communication, qui vont des réseaux locaux aux réseaux à grande échelle. Ce substrat matériel, constitué de machines inter-connectées, supporte un nombre important d'applications (p.ex. coordination d'activités, entrepôts de données, collecticiels, vidéo à la demande, commerce électronique) [Kra98] qui reposent de plus en plus sur l'Internet. Ce réseau à grande échelle permet, à un nombre sans cesse croissant d'utilisateurs, de faire coopérer leurs machines et d'accéder à une source d'informations gigantesque. Plusieurs schémas d'organisation sont utilisables pour la construction d'une application distribuée, où le procédé de construction comprend les étapes de spécification, de conception (choix de mise en œuvre) et d'implantation (génération de code et installation). Les schémas à mémoire distribuée partagée, à événements ou pairs à pairs (c.-à-d. *peer-to-peer*) sont des solutions prometteuses dans le cadre des réseaux grande échelle. Toutefois, suite au succès qu'il a connu dans les réseaux locaux, le schéma d'organisation client-serveur reste le plus répandu. Il permet de concevoir des applications où des clients accèdent à des fonctionnalités distantes, proposées sous la forme de services par des serveurs. Les ressources des serveurs sont souvent largement supérieures à celles des clients, notamment quand ceux-ci utilisent des assistants personnels ou des téléphones cellulaires.

Cette thèse étudie l'utilisation des agents mobiles dans les architectures de services. Une architecture de services repose sur une infrastructure qui permet à des fournisseurs de décrire, publier et rendre accessible leurs applications sous la forme de services. Les schémas d'organisation avec mobilité du code, c.-à-d. code à la demande, exécution à distance et agents mobiles, peuvent fournir une alternative intéressante aux interactions traditionnelles sur ces architectures. Comparés à des schémas plus classiques largement utilisés tels que le client-serveur, les avantages et inconvénients du code mobile sont principalement de nature non fonctionnelle.

Architectures de service

Les systèmes client-serveur deviennent de plus en plus complexes. Différentes abstractions telles que les processus communicants, les appels de procédure à distance et les invocations distantes de méthode entre objets ont été utilisées pour faciliter la conception des systèmes basés sur un schéma client-serveur. Les paradigmes utilisés dans ces systèmes (p.ex. procédure, objet, composant) ont connu des évolutions récentes afin de s'adapter notamment aux réseaux à grande échelle. L'approche à base de composants, vus comme des services fonctionnels, permet de faciliter l'utilisation, la construction et la réutilisation des systèmes client-serveur et favorise l'assemblage de services distribués. Actuellement, de plus en plus d'organisations, de

compagnies ou d'entreprises souhaitent externaliser leurs services. L'Internet ne se limite donc plus à une simple base d'informations. Ainsi, les architectures émergeantes orientées services, telles que les services Web, proposent et rendent accessibles des services primitifs par le Web. Elles se basent majoritairement sur le schéma d'organisation client-serveur et reposent sur des standards sur lesquels s'harmonisent les vendeurs. Les applications commencent à pouvoir interopérer sans problèmes de langages, de protocoles ou de plates-formes. Des compagnies ou des clients sont ainsi à même de concevoir des services plus complexes en agrégeant des services primitifs disponibles. Le concepteur d'un service distribué complexe tend alors à devenir un intégrateur de services primitifs plutôt qu'un programmeur. Il combine des services, en spécialise certains, pour en former des plus complexes.

Qualité de service

Les concepteurs de services distribués complexes doivent chercher à garantir/optimiser des propriétés qui ne dépendent pas directement de la fonctionnalité de l'application. Certaines de ces propriétés sont plus particulièrement liées à la construction, à l'intégration et à l'évolution du système (p.ex. temps de développement, adaptabilité, ouverture, portabilité, extensibilité, capacité de croissance, réutilisation). D'autres propriétés sont liées aux utilisateurs, à la manière dont ils perçoivent la qualité de l'application proposée. Les principales propriétés non fonctionnelles qui affectent la qualité de service[1], expérimentée par les utilisateurs, sont la performance, la sécurité et la sûreté de fonctionnement [CDK01]. Ces trois propriétés sont essentielles et doivent être prises en compte au plus tôt lors du procédé de construction. Pour garantir/optimiser de telles propriétés, les mécanismes à intégrer sont parfois complexes et nécessitent une expertise importante.

Performance : du point de vue de l'utilisateur, une application distribuée est performante si elle assure un temps d'exécution raisonnable. Dans une architecture de services, la performance se mesure le plus souvent en terme de temps de réponse suite à l'invocation d'un client. Ce temps dépend d'une part des capacités de calcul et de communication des machines (sites du réseau) et d'autres part des canaux de communication (liens du réseau). Pour quantifier ce temps de transfert, il est nécessaire de considérer à la fois la latence et la bande passante fournie par un canal de communication. La latence représente le délai entre la décision de l'envoi (resp. réception) d'un message et son émission effective. La bande passante détermine le volume de données qui peut être transmis entre deux entités d'un réseau en un temps donné. Dans les réseaux à grande échelle, la bande passante est dépendante des délais de routage et de l'engorgement des canaux de communication. Afin d'améliorer la performance d'une application pour le compte de clients, il convient de chercher à réduire le temps de transfert des messages entre les machines. Il existe ainsi des stratégies telles que l'utilisation de caches qui permettent de rapprocher les données des utilisateurs afin de minimiser les transferts à un instant donné. Une étude simplifiée de la performance d'une communication peut revenir à évaluer la taille des messages à faire transiter, connaissant la bande passante des canaux utilisés. Plus les messages à faire transiter sont petits, meilleures seront les performances du système final.

1. Nous utilisons dans ce document le terme qualité de service comme terme englobant pour les propriétés de performance, de sécurité et de sûreté de fonctionnement et non pas comme des garanties temporelles de flux de données audio ou vidéo dans un contexte réseau.

Sécurité : il n'est pas toujours possible d'avoir une confiance aveugle envers les sites et les canaux d'un réseau. Par exemple, pour un service de commerce électronique, il convient de s'assurer que le numéro de carte bancaire d'un client ne pourra être connu d'un individu malveillant, ou encore que les décisions d'achats ne seront pas modifiées par autrui. Les données d'un client doivent donc parfois être protégées, tout comme les données d'un site. De même, les sites doivent pouvoir se protéger contre du code malveillant qu'ils exécuteraient, tout comme ce code doit pouvoir être protégé d'un site malintentionné. Les sujets (p.ex. utilisateurs, processus, périphériques) et les objets (p.ex. ressources, données) sont distingués dans le domaine de la sécurité informatique [BI95]. Du point de vue de l'utilisateur, trois propriétés sont particulièrement importantes : la confidentialité, l'intégrité et la disponibilité. La sécurité est souvent assimilée aux seules propriétés de confidentialité et d'intégrité des objets. La disponibilité est plus difficile à étudier et nécessite de prendre en compte des paramètres dynamiques de l'environnement d'exécution tels que la charge des machines. Afin de garantir les propriétés de confidentialité et d'intégrité, il convient de protéger les objets, les ressources et les canaux de communication contre d'éventuelles attaques. Pour cela, des mécanismes d'authentification, de cryptographie (chiffrement) et de contrôle d'accès [McL94] (p.ex. barrières de protection ou pare-feu) sont généralement utilisés.

Sûreté de fonctionnement : une application distribuée sûre de fonctionnement doit pouvoir continuer de délivrer ses fonctionnalités, conformément à ses spécifications, en présence de défaillances [Lap95]. Une défaillance est un événement qui altère la fonctionnalité d'une unité (c.-à-d. composant matériel, composant logiciel ou le système complet). Elle peut être logicielle ou matérielle et affecter des processus (p.ex. exécution incorrecte, arrêt) ou des canaux de communications (p.ex. perte de message). La sûreté de fonctionnement peut être vue selon des propriétés différentes mais complémentaires [SI99] : la fiabilité, la disponibilité et la sécurité-innocuité. Pour améliorer la sûreté de fonctionnement, il est nécessaire d'annuler les conséquences de défaillances, soit en les interdisant/contrant (p.ex. prévention, vérifications, preuves) soit en les tolérant. Dans le deuxième cas, des mécanismes de masquage de défaillances sont utilisés pour optimiser la propriété de fiabilité. Ils s'appuient le plus souvent sur la redondance de ressources, par réplication spatiale (p.ex. implantation d'un service comme un groupe de services redondants, physiquement indépendants).

Objectifs, approche et contributions

Il nous paraît clair que les notions de mobilité sont amenées à jouer un rôle central dans les architectures de services. Dans une certaine mesure, la mobilité de code permet de pallier des problèmes non fonctionnels. Par exemple, l'utilisation des agents mobiles peut permettre d'améliorer les performances des communications. En déportant des calculs, ils favorisent la diminution de la consommation en bande passante à travers le réseau. Bien que ces agents posent de nouveaux problèmes de sécurité, ils peuvent permettre des interactions distantes robustes sur des réseaux soumis à des défaillances de canaux de communication ou de machines. De plus, la possibilité de faire migrer le code d'une application entre les machines ouvre la voie à l'extensibilité des services offerts [Per97]. Malheureusement, à ce jour, il n'existe que peu d'approches qui permettent de placer dans un même cadre les mécanismes d'interaction à base de code mobile et les mécanismes plus classiques tels que l'invocation distante (p.ex. RPC, RMI). Ainsi, le choix quant aux meilleurs mécanismes d'interaction (p.ex. invocation

distante, migration d'agent) à mettre en place afin de couvrir les besoins en qualité reste un problème ouvert. Les réponses à ces choix dépendent jusqu'ici des expériences et expertises des concepteurs. Cette thèse vise à remédier à cette situation. Elle s'attache à :

1. définir des services complexes par composition de services primitifs, dans un formalisme déclaratif qui supporte à la fois des interactions par invocation distante et par agents mobiles. Les services complexes peuvent être exprimés à deux niveaux d'abstraction :

   (a) au niveau abstrait, un service complexe est décrit par une expression fonctionnelle simple. Elle spécifie clairement la sémantique des dépendances entre les services, à la manière d'un script à flots de données (*workflow*).

   (b) au niveau concret, l'expression décrivant un service complexe explicite les mécanismes d'interactions. Une expression concrète peut être vue comme la compilation d'une expression abstraite.

   Notre approche pour la spécification de services distribués complexes à deux niveaux d'abstraction contribue à l'étude de différentes interactions avec des services primitifs.

2. analyser, du point de vue client, les propriétés de performance, de sécurité et de fiabilité. Cette approche conduit à des techniques et outils destinés à guider au mieux le concepteur de services distribués complexes dans ses choix de mises en œuvre (p.ex. sélection automatique de mécanismes d'interactions, choix des meilleurs itinéraires d'agents). Elle ouvre la voie à la comparaison, dans un même cadre et au regard de critères de qualité, entre les interactions par invocations distantes et agents mobiles dans les architectures de services distribués. Elle permet ainsi de justifier de l'intérêt des agents mobiles dans ces architectures et de traiter les propriétés de qualité au plus tôt dans le procédé de construction.

3. intégrer ce cadre formel et des analyses de qualité dans un environnement de construction de services distribués complexes qui automatise l'approche. Le procédé de construction d'un service complexe comprend les étapes de spécification, de conception (utilisation des analyses de qualité), d'implantation du service complexe et de développement du système distribué (conception et déploiement du système). Pour cela, nous associons notre cadre à un langage de description d'architecture, voué à faciliter la conception du service complexe ainsi que le développement du système d'exécution qui le supporte. Cette intégration nous permet de clairement distinguer les rôles de l'architecte (spécification d'un service complexe), du concepteur (analyses sur le services complexes) et de l'équipe de développement qui a en charge d'implanter les services distribués complexes et de développer les parties de l'environnement d'exécution. L'environnement proposé est validé, en partie, en s'appuyant sur une plate-forme d'exécution qui offre les différents mécanismes d'interactions traitées.

Plan

Le document comprend trois parties.

La première partie, composée des chapitres 1 et 2, situe le contexte de travail de la thèse. Le chapitre 1 présente les principaux schémas d'organisation distribuée. Nous nous focalisons sur le schéma client-serveur, largement prédominant, puis passons en revue les principaux mécanismes d'interaction qui permettent de le mettre en œuvre. Nous étudions ensuite les schémas d'organisation du paradigme code mobile qui offrent une alternative intéressante aux mécanismes d'interactions utilisés dans le schéma client-serveur. Nous décrivons également brièvement les infrastructures nécessaires à l'exécution des agents mobiles.

Le chapitre 2 présente les différents avantages et inconvénients des agents mobiles en termes de propriétés non fonctionnelles. Il s'intéresse plus particulièrement à la performance, la confidentialité, l'intégrité et la fiabilité.

La seconde partie, composée des chapitres 3 et 4, décrit notre cadre pour la spécification de services distribués complexes et les analyses associées. Le chapitre 3 introduit notre cadre formel, en particulier les deux langages de spécification, à savoir le langage abstrait et le langage concret.

Le chapitre 4 présente les différentes analyses de qualité possibles dans ce cadre. Les propriétés de qualité traitées concernent la performance *via* l'évaluation du trafic total engendré par un service complexe dans le réseau, la sécurité au regard des propriétés de confidentialité et d'intégrité, et finalement la fiabilité des interactions. Ces analyses permettent d'étudier différents mécanismes d'interactions possibles pour la mise en œuvre d'un service complexe et de sélectionner ceux qui conviennent le mieux aux besoins de qualité. Des extensions possibles du langage de spécification sont discutées.

La troisième partie, composée du chapitre 5, décrit l'intégration de notre cadre et d'une sélection d'analyses dans un environnement de construction de services distribués complexes. La structure globale de notre environnement est présentée. Il guide, le plus automatiquement possible, la conception du service complexe et le développement du système, tout en respectant des critères de qualité. Cette intégration s'appuie, pour l'ensemble des spécifications, sur un langage de description d'architectures. Son association avec l'environnement de construction de systèmes Aster [IB96] garantit le déploiement systématique d'un système au regard des contraintes de qualité requises par le service complexe. Ce chapitre présente les outils et expérimentations développés autour de cette approche. Il étudie la mise en œuvre de quelques analyses afin de justifier de la faisabilité de l'approche. Nous y décrivons la mise en œuvre de notre proposition à l'aide de la plate-forme Grasshopper [IKV]. Elle offre, de manière uniforme, à la fois des interactions par invocations distantes et par agents mobiles.

Le document se conclut par un bilan de l'approche, une discussion des contributions de ce travail, ainsi que par des perspectives de recherche qu'il suscite.

Chapitre 1

Schémas d'organisation distribuée

Ce chapitre s'attache à présenter les schémas d'organisation distribuée significatifs pour concevoir et déployer des architectures de services. Une architecture de services repose sur une infrastructure qui permet à des fournisseurs de décrire, publier et rendre accessible à leurs clients des applications primitives sous la forme de services.

Dans un premier temps, nous décrivons brièvement les principaux schémas d'organisation utilisés pour concevoir des systèmes/applications distribués. Nous nous intéressons ensuite plus particulièrement au schéma client-serveur, largement utilisé dans les architectures de services et étudions les divers mécanismes d'interaction qui s'y rapportent. Nous décrivons ensuite les technologies basée sur le paradigme code mobile, qui offrent une alternative intéressante aux mécanismes d'interactions utilisés dans le schéma client-serveur. Nous présentons également rapidement les infrastructures nécessaires à l'utilisation des agents mobiles. En conclusion, nous discutons les différents points présentés en fonction de la problématique de cette thèse.

1.1 Contexte

Suite à la réduction du coût des machines, les systèmes distribués se généralisent largement. Ils sont constitués d'un ensemble d'éléments matériels ou logiciels, localisés sur différentes machines, qui interagissent pour atteindre un but commun. Les éléments de ces systèmes coordonnent leurs activités et échangent de l'information par transmission de messages à travers le ou les réseaux de communication qui relient les machines [SK88]. Les réseaux grande échelle, particulièrement l'Internet, sont de plus en plus utilisés. Dans ces réseaux, des machines puissantes (p.ex. macro-ordinateurs du type *mainframe*) côtoient des unités de calcul fixes ou mobiles à ressources plus restreintes (p.ex. micro-ordinateurs, assistants personnels, téléphones cellulaires UMTS, cartes à puces). Le partage de données et de ressources devient une motivation centrale lors de la conception d'applications sur ces réseaux de machines hétérogènes. Plusieurs schémas d'organisation sont éligibles. Le choix du placement des éléments sur le réseau, leurs rôles et la manière dont ils communiquent influent particulièrement sur les propriétés non fonctionnelles d'une application distribuée. Nous présentons ici brièvement les schémas d'organisation distribuée client-serveur, à mémoire partagée, à événements et pair à pair, avant de détailler plus avant le schéma client-serveur.

Client-serveur

De nombreux systèmes distribués peuvent être construits entièrement sous la forme de clients qui interagissent avec des serveurs. Les fonctionnalités d'une application sont alors encapsulées dans des services et proposées/exécutées au sein de serveurs. Les serveurs sont ainsi vus comme des fournisseurs de services. Deux types de client-serveur prédominent [GG96], selon le type de service déporté : (i) le client-serveur de données, dans lequel un client accède à des données localisées sur un serveur (p.ex. service de données d'un SGBD), (ii) le client-serveur de procédure dans lequel un client fait sous-traiter des exécutions à un serveur.

Le client-serveur est avant tout une technique de dialogue entre deux processus. C'est le client, demandeur d'un service, qui établit une interaction avec un serveur. Il orchestre les interactions avec les différents services requis. Ainsi, seul le client, qui est à la base du dialogue, est une application au sens propre du terme. Le serveur a pour seul objet de répondre aux demandes des processus clients. Il est indépendant des applications qui l'interrogent. Toutefois, un serveur pouvant interagir avec d'autres serveurs, un même processus peut être à la fois client et serveur.

Un échange entre un client et un serveur consiste en la transmission au serveur d'une requête. Ce message décrit l'opération à exécuter (c.-à-d. le service) ainsi que ses données paramètres. On dit alors que le client invoque une opération sur le serveur. Le serveur exécute le service demandé par le client et retourne la réponse (résultat). Une interaction complète entre un client et un serveur (c.-à-d. donnée requête et donnée réponse) est appelée une invocation distante [1]. Le schéma de base est l'exécution synchrone. Le processus client émet une requête et se bloque, en attente de la réponse. L'exécution du service nécessite [FPV98] [2] :

1. le code du service. Le code est le programme réalisant le service (c.-à-d. les opérations à effectuer) et représente le « savoir-faire ».

2. une unité d'exécution qui est une machine qui encapsule l'état d'exécution (c.-à-d. pile, tas, compteur ordinal).

3. un ensemble de ressources qui représentent les données passives ou les unités physiques (p.ex. processeur).

Afin d'accomplir un service pour le compte d'un client, ces trois éléments doivent se trouver, à un instant donné, sur un même site.

La Figure 1.1 propose schématiquement l'interaction, où le client émet une donnée requête au serveur afin de récupérer, après exécution, un résultat. Dans le schéma client-serveur, le code, l'unité d'exécution et les ressources sont présents sur le serveur et réalisent un service primitif. Les calculs sont donc effectués sur le site serveur (cf. boîte grisée sur la figure).

Mémoire distribuée partagée

Le schéma d'organisation à mémoire distribuée partagée [BZS93] consiste en des processus distribués qui interagissent *via* une mémoire de grande taille qui sert d'espace de communication. Ce schéma est particulièrement adapté aux applications où le partage est important,

1. Dans toute la suite du document, ce terme est employé comme terme englobant pour l'appel de procédure à distance et l'invocation distante de méthode ou de service.
2. Ce découpage, issu de [FPV98], sera réutilisé par la suite pour présenter les schémas avec mobilité.

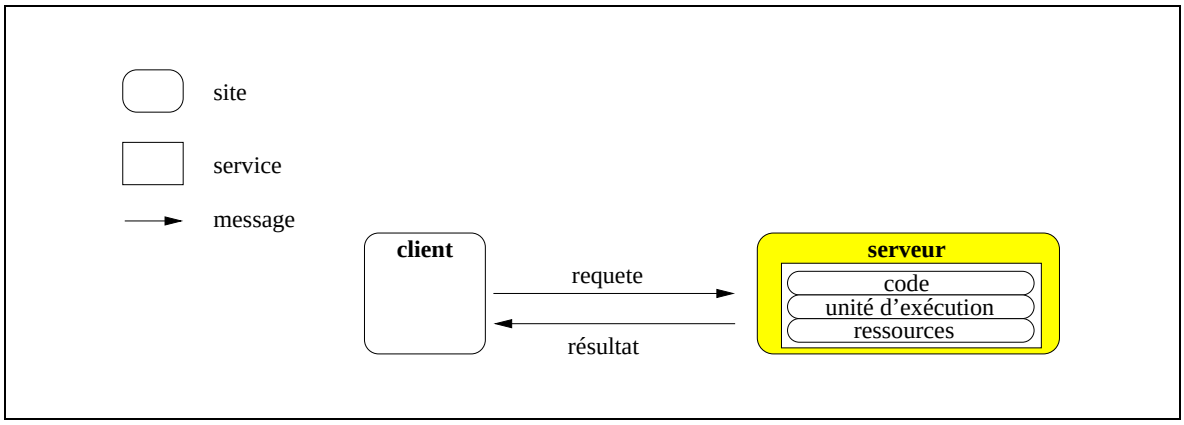

FIG. 1.1 – *Schéma d'interaction client-serveur*

comme c'est le cas dans les applications coopératives (p.ex. collecticiels). Les premiers prototypes de recherche basés sur ce schéma datent des années 80 dans le cadre des réseaux locaux (p.ex. Argus). Ce schéma s'est ensuite orienté vers une organisation par objets partagés repartis. Ainsi, le système PerDiS [FSB+00], adapté aux réseaux à grande échelle, s'appuie sur un espace d'objets global pour le partage et la persistance des objets. D'autres environnements reposent sur un espace de n-uplets (p.ex. Linda [Gel85]) avec une abstraction du type « pot commun ». Ils permettent la lecture, l'écriture, le dépôt, le retrait ou la consultation d'objets dans un espace donné. Ainsi, un environnement tel que JavaSpace [Sun98] propose la mise en œuvre d'applications distribuées en se basant sur le partage et la circulation d'objets. Les participants peuvent insérer et retirer des objets dans un JavaSpace, qui est un multi-ensemble d'entrées où une entrée représente un groupe d'objets.

Événements

Le schéma d'organisation à événements permet la communication entre plusieurs processus mais ne propose pas de communication directe entre les processus. Un événement est une occurrence asynchrone dans un système distribué, tel un clic utilisateur pour interagir avec une application. Les processus communiquent entre eux par l'intermédiaire d'une base, qui est un gestionnaire d'abonnements et d'événements. Cette base se charge de faire correspondre un événement à ses abonnés. Il y a deux types de processus participants : les fournisseurs qui envoient ou publient des événements dans la base, et les consommateurs qui s'abonnent ou souscrivent à certaines catégories d'événements. Quand la base reçoit une notification d'événement par un processus, elle y associe des réactions et émet alors des événements vers les processus abonnés. Un fournisseur ne connaît pas nécessairement l'identifiant des consommateurs. Un processus se faisant notifier un événement n'a pas nécessairement connaissance de l'initiateur de cet événement.

Ces systèmes sont souvent construits à l'aide d'un intergiciel orienté messages (MOM, *Message Oriented Middleware*). Ces intergiciels servent de sous-couche pour la mise en œuvre des schémas à événements qui s'appuient le plus souvent sur des communications asynchrones pour les échanges de messages. Dans ces intergiciels, les éléments distribués interagissent par des mécanismes de fil de messages ou de publication/souscription.

Pair à pair

Le schéma d'organisation pair à pair [p2p] s'appuie sur les flots de communication unidirectionnels entre deux processus voisins. Il est particulièrement adapté aux réseaux dynamiques (c.-à-d. à topologie changeante). Il est basé sur une équité des rôles de tous les acteurs de l'application. Ces acteurs sont vus comme des « *servent* » (c.-à-d. fusion des termes *server* et *client*), qui interagissent par paire (c.-à-d. liaison point à point). Chaque *servent* dispose à un instant donné d'un ensemble de *servents* voisins, avec lesquels il interagit pour son compte ou pour le compte d'un *servent* qui l'a contacté. Il n'y a pas de point central dans le système et les données sont totalement décentralisées.

L'environnement JXTA [Sun01] fournit des abstractions pour les mécanismes de communications point à point (p.ex. *peerRendez-vous*, *peergrouping*) et y associe un certain nombre de services (p.ex. recherche de ressources). Sur le réseau Internet, des infrastructures telles que Gnutella ou FreeNet, reposent sur ce schéma pour la recherche de fichiers (Gnutella) ou le stockage de documents anonymes (FreeNet). Un utilisateur émet une demande de recherche de fichier à son ou ses voisins, qui eux-mêmes retournent cette demande à leurs voisins propres et ainsi de suite. Dès qu'un voisin de rang n dispose du fichier requis, il retransmet l'information à son voisin demandeur de rang $n-1$ et ce jusqu'à ce que l'information remonte jusqu'à l'utilisateur initial. À aucun moment, un voisin n'a à connaître l'identifiant de l'utilisateur dont la requête circule.

Synthèse

Une application distribuée peut reposer sur différents schémas d'organisation. Les réseaux sous-jacents et les critères non fonctionnels sont au cœur du choix de l'un de ces schémas lors de la conception. Certains schémas supportent bien l'extensibilité et l'adaptabilité (p.ex. pair à pair), d'autres sont plutôt adaptés aux applications où le partage est important (p.ex. mémoire partagée). La facilité de mise en œuvre de mécanismes destinés à garantir/optimiser les propriétés de performance, de sécurité ou de sûreté de fonctionnement est également liée au schéma choisi.

Dans les systèmes distribués, le schéma client-serveur reste le plus largement répandu. Grâce à la mise en commun de services (p.ex. procédures, méthodes), il permet de faire cohabiter dans un même système des machines aux performances très différentes. Les clients peuvent faire sous-traiter des exécutions à des machines puissantes. La construction des applications est favorisée par la réutilisation de services existants. Ce schéma d'organisation est donc particulièrement adapté aux applications distribuées. Nous étudions dans la section suivante les mécanismes d'interaction qui se sont développés pour la mise en œuvre du schéma client-serveur, puis présentons ensuite des variations de ce schéma, supportant du code mobile, qui peuvent en être dérivées.

1.2 Mécanismes d'interaction du schéma client-serveur

Plusieurs mécanismes d'interaction sont apparus pour mettre en œuvre le schéma d'organisation client-serveur. Ils sont liés aux abstractions de plus en plus haut niveau qui sont employées dans l'ingénierie des systèmes afin de réduire la complexité du procédé de construction. Nous décrivons ici le mécanisme de l'appel de procédure à distance puis survolons les

approches à base d'intergiciel[3] utilisant l'invocation de méthodes. Nous étudions ensuite les approches actuelles qui tendent à s'orienter vers des interactions entre entités à plus gros grain : les composants.

1.2.1 Appel de procédure

Dans les années 80, l'extension de l'appel de procédure en appel de procédure à distance [BN84] ou RPC (*Remote Procedure Call*) a permis aux processus d'appeler des procédures qui s'exécutent sur des machines distantes. Les différents formats propriétaires (p.ex. ONC *Open Network Computing* [Sun91], DCE *Distributed Computing Environment* [osf91]) liés au RPC ont conduit à proposer une couche d'abstraction supplémentaire qui unifie les systèmes RPC propriétaires et la création d'applications distribuées dans les réseaux locaux. Ainsi, les concepts de RM-ODP [ISO92], standardisé par l'ISO, ont notamment initié l'abstraction qui est aujourd'hui à la base de l'élaboration d'un système distribué, à savoir la transparence envers la localisation. Cette abstraction permet de développer un système distribué sans se soucier de la localisation des processus. Il se crée ainsi l'illusion que toutes les procédures accédées par un client sont situées sur le même site.

1.2.2 Invocation de méthode

Par souci initial de réutilisation et pour faciliter la conception et la maintenance, le domaine du génie logiciel s'est porté vers la programmation orientée objets. Un objet est une abstraction conceptuelle qui encapsule des données, associées à un ensemble de méthodes. Les systèmes basés sur des objets distribués adoptent le plus souvent un schéma client-serveur. Les objets sont gérés par des serveurs et les clients invoquent leurs méthodes [Par90, omg91] (RPC objet, appelé RMI, *Remote Method Invocation*).

Pour faciliter l'implantation et rendre la distribution transparente, des outils logiciels à base d'intergiciels orientés objets ont été proposés (OOM : *Object Oriented Middleware* tels que CORBA [omg91], DCOM (*Distributed Component Object Model*) [Ses97] ou J2EE (ORB JavaRMI [Sun97])). Un intergiciel à invocation distante a en charge de rediriger les appels de méthodes (requête) et les résultats entre les deux acteurs d'une invocation. En plus de masquer la localisation des objets, un tel intergiciel assurent également le masquage de l'hétérogénéité des machines, des réseaux et éventuellement des systèmes d'exploitations ou des langages de programmation utilisés. Son architecture s'articule autour d'un logiciel de communication, l'échangeur de requêtes entre objets (ORB ou *Object Request Broker*), également appelé médiateur d'objets ou bus à objets. Un intergiciel repose sur trois notions : le talon client qui simule un objet local pour le client, le module de communication qui supporte le protocole requête-résultat, et le squelette serveur qui fait exécuter la méthode invoquée. Ainsi, l'OMG™ (*Object Management Group*) a proposé avec CORBA2 (*Common Object Request Broker Architecture*) une norme pour les architectures ouvertes client-serveur à objets distribués. Les spécifications contiennent un modèle, une architecture et un langage de définition d'interface (c.-à-d. IDL pour *Interface Definition Language*) pour :

1. offrir l'interopérabilité entre les applications sur différents sites dans un environnement distribué.

3. Les termes médiaticiel ou interstitiel sont également utilisés pour ces bus à objets (*middleware*).

2. standardiser les mécanismes par lesquels les objets émettent des messages et reçoivent des résultats indépendamment de leurs mises en œuvre (p.ex. C++, Java™). Pour cela, les objets serveurs exposent leurs fonctionnalités *via* des interfaces typées, décrites à partir d'un langage de définition d'interface. Une interface décrit les signatures de l'ensemble des méthodes qu'un client peut invoquer sur un objet, sans spécifier leurs mises en œuvre.

3. standardiser les services intergiciels (services objet communs) nécessaires à la gestion d'objets distribués (p.ex. les services de nommage, sécurité, persistance, transactions, événements, courtage *trading*).

CORBA permet l'inter-opération entre des mises en œuvre d'ORB propriétaires avec l'utilisation du protocole IIOP (*Internet Inter-ORB Protocol*), qui est un format commun d'échange d'objets sur TCP/IP.

Les dernières abstractions dans le domaine de l'ingénierie distribuée sont de plus en plus conceptuelles et permettent ainsi de s'éloigner des complexités matérielles et logicielles. En ce sens, les récents développements en terme de composants permettent aux programmeurs de voir leurs systèmes avec une granularité plus large que celle des objets. Cette approche facilite la construction des applications par assemblage/composition et favorise la réutilisation. CORBA CCM (*Component Model*), synthétisant les idées de DCOM et EJB, étend ainsi le modèle objet de CORBA2 et permet de décrire la composition de composants et le déploiement [4].

CORBA reste la seule plate-forme intergicielle non propriétaire, mais d'autres plates-formes de dernière génération connaissent un réel succès (p.ex. .Net, SunONE *Open Network Environment*). Afin de les fédérer, l'OMG a récemment choisi de proposer les architectures dirigées par les modèles. Ce modèle abstrait permettra de spécifier les systèmes à l'aide de la notation UML™, afin de supporter le cycle de vie de la conception, du déploiement et de la maintenance des applications, indépendamment des plates-formes cibles.

1.2.3 Invocation de service

De plus en plus, les architectures reposant sur un schéma client-serveur s'appuient sur l'Internet. Cependant, les protocoles existants, tels que IIOP ou celui de DCOM s'adaptent difficilement aux environnements à grande échelle. Ils nécessitent un support d'exécution dédié. Des spécifications de protocole pour l'invocation distante telles que SOAP (*Simple Object Access Protocol*) [W3C01] émergent afin de standardiser les communications entre les applications à travers l'Internet, où les clients et serveurs s'affranchissent d'un ORB. SOAP défini un format de messages XML [5] favorisant l'interopérabilité. Pour les communications, l'infrastructure SOAP utilise le protocole HTTP. À la différence de DCOM ou IIOP, un client peut communiquer avec un serveur à travers un pare-feu [6] sans nécessiter d'ouvrir des brèches dans celui-ci. Les messages d'un protocole SOAP sont de taille assez importante et donc consommateurs de bande passante.

La tendance actuelle dans l'ingénierie des applications distribuées est de construire des systèmes complexes et ouverts par assemblage de composants haut niveau. SOAP permet de

4. Le déploiement réfère à la distribution des composants, leur installation et configuration.

5. XML est un langage, défini par le consortium W3C, pour décrire des structures complexes de données ou de documents [W3C00].

6. Un pare-feu est un filtre localisé sur l'entrée d'un réseau local et qui est responsable de traiter les paquets provenant de l'Internet et s'assurer qu'ils n'accèdent pas à des ports privilégiés du réseau local.

réaliser des appels de méthodes sur le Web et a ainsi conduit à l'engouement récent envers les services Web [IBM01]. Cette technologie émergeante [CaSW01] d'intégration d'applications dans un environnement Web, favorise et étend la collaboration entre les entreprises et leurs clients par externalisation de services. Un service Web peut être une application J2EE (Java), un composant .Net (C# ou VisualBasic) ou une application basée CORBA. Actuellement, ce type de services se développe rapidement suite à la simplicité d'utilisation. Les services peuvent être en accès gratuit ou bien accessibles sous un mode « paiement-par-usage ». Le Web devient donc une architecture de services. L'architecte, voire même le client, sont de plus en plus amenés à vouloir composer ou assembler ces services présents sur le réseau.

Dans une architecture de services Web, les services sont répertoriés dans des annuaires (répertoires UDDI *Universal Description, Discovery and Integration*), qui peuvent être vus comme un service de « pages jaunes ». Ils permettent aux clients de rechercher les fonctionnalités accessibles. Ces répertoires contiennent des documents, en langage WSDL (*Web Services Description Language*), au format XML. Ces documents décrivent les interfaces des services et le mécanisme de communication associé. Les interfaces décrites en WSDL supportent les opérations d'invocation distante (bidirectionnel du type requête-réponse) et d'échange de messages (unidirectionnel du type émission ou notification). Les détails d'un service, tels que son nom, sa localisation, les types et ordres de ses paramètres et résultats sont ainsi représentés dans ces documents.

1.3 Schémas d'organisation avec mobilité

Le code mobile offre de nombreux atouts par rapport aux mécanismes d'interaction classiques tels que l'invocation distante. La recrudescence de l'intérêt envers le code mobile est venue principalement d'une nouvelle famille de langages supportant la mobilité [Tho97]. Un code mobile peut être envoyé d'un site pour être exécuté sur une autre. Avec la croissance de la taille des réseaux et plus particulièrement le développement de l'Internet, l'approche qui consiste à déplacer les calculs devient un paradigme de conception important. Dans les architectures d'applications, il est possible de voir les schémas d'organisation avec code mobile comme une variation du schéma client-serveur [CPV97]. Les schémas à base de code mobile (c.-à-d. évaluation distante, code à la demande, agent mobile) peuvent être distingués en considérant la localisation des différentes entités du système (c.-à-d. ressources, code et unité d'exécution; cf. introduction client-serveur page 8), avant et après l'exécution d'un service [FPV98, Pic98].

1.3.1 Évaluation distante

Dans une interaction par évaluation distante, un composant[7] client envoie un code à un autre site. Le composant récepteur exécute le programme, le code de l'application. Ce code peut contenir des données. Éventuellement, une interaction additionnelle délivre ensuite les résultats du service au composant émetteur [SG90]. L'unité d'exécution (c.-à-d. compteur ordinal, pile, tas) et les ressources sont fixes, seul le savoir-faire est mobile. La Figure 1.2 présente le schéma à la manière de la Figure 1.1, page 9, illustrant le schéma client-serveur. Le service est réalisé sur le composant serveur (c.-à-d. boîte grisée), à la réception du code.

7. Nous utilisons ici le terme composant comme un acteur d'un système et non pas comme il est utilisé dans la section 1.2.2 page 12.

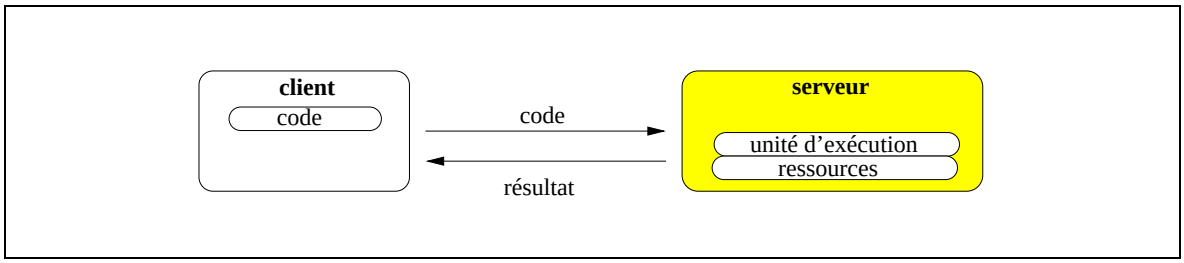

FIG. 1.2 – *Évaluation distante*

Un exemple d'évaluation distante est la commande `rsh` du système Unix. Elle permet à un utilisateur d'exécuter une suite d'instructions (c.-à-d. *script*) sur un site distant. Les interactions avec les imprimantes PostScript sont également réalisées par ce schéma. La ressource d'impression reçoit le code sous la forme d'un fichier. Ce fichier est ensuite interprété par l'unité d'exécution de l'imprimante. Les requêtes SQL (code à interpréter), émises vers un serveur de base de données ou encore les *servlets* Java présentes sur les serveurs HTTP, sont également réalisées par ce schéma.

1.3.2 Code à la demande

Dans ce schéma, le composant client interagit avec un composant distant afin de récupérer le code nécessaire à la réalisation d'un service. La requête ne contient pas de données paramètres à utiliser sur le site distant. Le composant distant fournit le code du service qui sera exécuté sur le site client (voir Figure 1.3). Ainsi, en permettant à des composants de récupérer du code, cette technologie permet d'étendre dynamiquement le comportement d'une application. L'exemple largement diffusé de cette abstraction est l'appliquette (*applet*) Java où un programme est chargé à partir d'une page Web pour être exécuté sur le poste du client.

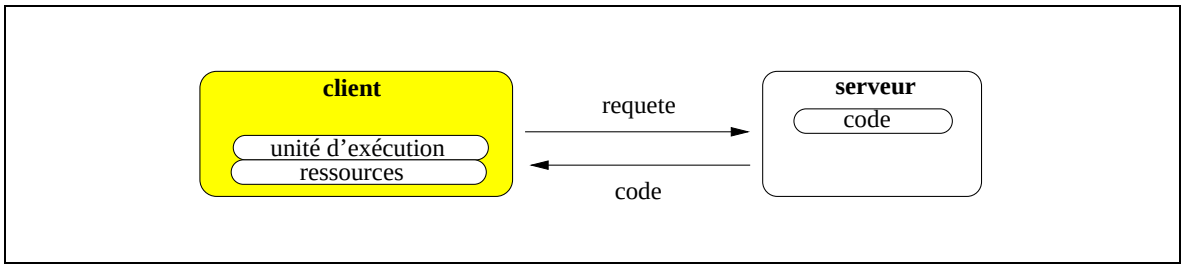

FIG. 1.3 – *Interaction par code à la demande*

1.3.3 Agents mobiles

Les agents mobiles sont inspirés de travaux sur le calcul intensif initiés au sein de Xerox [SH82]. La notion d'agent mobile a été introduite pour la première fois en 1994 par White [Whi94] qui décrit l'environnement Telescript. Dans cet environnement, des processus (code et unité d'exécution) pouvaient[8] se déplacer d'eux-mêmes d'un site du réseau à un autre pour interagir localement avec des ressources d'autres sites. Cette technologie est alors

8. Cet environnement n'est plus disponible.

apparue comme prometteuse pour la conception d'applications distribuées [CHK94, Car99]. Dans ce schéma, le savoir-faire appartient au client. Les agents mobiles peuvent être vus comme une généralisation de l'évaluation distante, où en plus du code, l'unité d'exécution associée est mobile. Les agents mobiles peuvent transporter des données paramètres. Afin d'accéder aux ressources, les processus agents migrent de manière autonome (c.-à-d. instructions dans le code) vers les composants qui les proposent (voir Figure 1.4). Après exécution et une fois son/ses service(s) réalisé(s) (cf. boîte grisée sur la Figure), l'agent mobile retourne éventuellement vers le composant client.

Au niveau de la mise en œuvre, les migrations d'agents peuvent s'effectuer selon deux modes :

1. la migration forte (p.ex. Telescript, Ara, AgentTcl), où la totalité de l'agent (c.-à-d. code, données et unité d'exécution) migre vers le nouveau site. Pour cette migration réelle (migration par déplacement), l'agent est suspendu (capturé) avant d'être transféré. Une fois arrivé sur le site distant, il redémarre son exécution (restauration) au point de contrôle précédent, en conservant l'état du processus (unité d'exécution).

   $$\text{MigrationForte} \equiv \text{code} + \text{unité d'exécution} + \text{données-paramètres}$$

   Une autre possibilité proposée consiste à stopper l'exécution de l'agent avant la migration puis d'en créer une copie distante identique sur le site distant (migration par réplication).

2. la migration faible où seul le code et les données migrent.

   $$\text{MigrationFaible} \equiv \text{code} + \text{données-paramètres}$$

   Une fois arrivé sur le site distant, l'agent est réexécuté. Le programmeur doit donc préserver, dans les données, les informations d'état permettant la poursuite logique de l'exécution.

À la différence de l'évaluation distante, avec un agent mobile, l'exécution du code est initiée sur le composant client et continuée sur une collection de composants visités. Un agent mobile à migration faible, dont l'itinéraire est limité à un unique site, migrant dès le début de son exécution, peut correspondre au schéma d'évaluation distante (l'état du processus transporté par les données-paramètres étant vide).

Par leur nature, les agents mobiles traitent la distribution en interne. Un agent mobile est donc un processus migrant volontairement. Il peut se déplacer de sites en sites en suivant un itinéraire en fonction des tâches à réaliser.

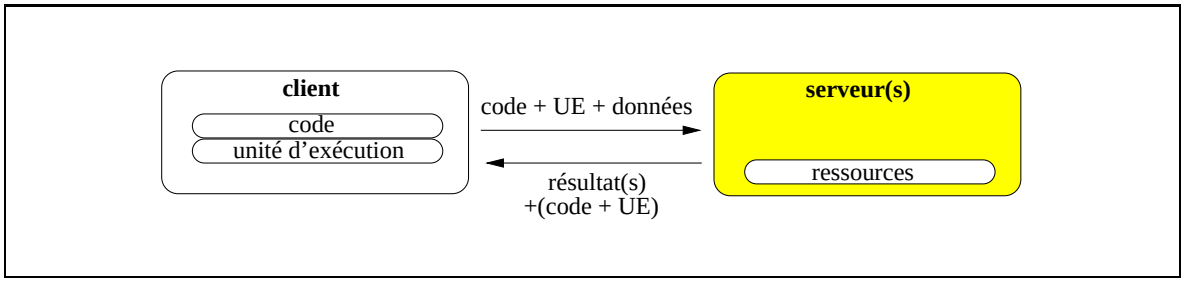

FIG. 1.4 – *Schéma d'interaction par agent mobile (migration forte)*

1.4 Infrastructures pour agents mobiles

En 1994, la société General Magic [Whi94] a proposé le premier système supportant les agents mobiles. Ce système propriétaire était peu adapté à un environnement tel que l'Internet. L'intérêt suscité par la technologie à base d'agents mobiles a contribué ensuite à un nombre important de développements de plates-formes. Une liste [Hoh] répertorie plus d'une soixantaine de plates-formes à ce jour. Certaines d'entre elles sont utilisées dans le cadre de travaux de recherche. D'autres sont fournies sous la forme de produits commerciaux. Il existe des études comparatives de certaines de ces plates-formes (p.ex. [UB97, Ber99]). Une étude sur la performance en termes de rapidité à l'exécution et de trafics générés a été proposée dans [SSM+00]. Bien que les techniques mises en œuvre pour les exécutions des agents et la gestion de leur mobilité diffèrent, les plates-formes existantes présentent pour la plupart des fonctionnalités communes. Plusieurs services intergiciels associés aux plates-formes sont proposés. Ces services incluent notamment des supports pour la sécurité, la communication entre les agents ou encore la persistance.

Dans cette section, nous nous intéressons aux langages de programmation qui sont employés au sein de ces plates-formes. Nous discutons brièvement des différentes techniques de gestion de la mobilité des agents pour le langage Java, largement utilisé. Nous étudions ensuite les principales caractéristiques des plates-formes agents mobiles existantes.

1.4.1 Java : langage de programmation pour agents mobiles

Plusieurs langages supportent la mobilité du code [Tho97] (p.ex. Java, Obliq, Telescript, Tcl). Les plates-formes à agents mobiles s'appuient sur ces langages. Toutefois, le langage Java reste aujourd'hui le plus répandu. L'environnement Java a connu un rapide succès pour le développement d'applications distribuées, en partie du fait que la machine virtuelle Java est portée sur la plupart des systèmes courants. Elle permet une indépendance envers le réseau physique et les systèmes d'exploitation des sites hôtes. Mis en œuvre en Java, les agents peuvent migrer entre des sites hétérogènes. Java permet la mobilité du code et des objets grâce au mécanisme de sérialisation [Sun]. La sérialisation permet de capturer et de restaurer l'état des objets avant et après un déplacement entre sites.

L'environnement Java actuel ne fournit pas d'API permettant de capturer ou restaurer l'état d'un processus (c.-à-d. *thread*) au cours de son exécution, la pile d'un flot Java n'étant pas accessible. Il n'est donc pas possible de faire migrer simplement un agent entre deux machines virtuelles. Java ne supporte donc pas la mobilité forte dans la mesure où il n'est pas possible de redémarrer un agent au point où il s'est interrompu sur un autre site. Toutefois, des travaux ont été proposés (p.ex. [IH99]) afin d'étendre la machine virtuelle, mais ils ne sont pas encore acceptés par Sun. L'état et les données non constantes doivent donc être empaquetés avant la migration. Cette opération, à la charge du programmeur, permet d'envoyer le code, les données et l'état de l'agent sur un site distant après sérialisation. L'agent est alors détruit sur le site d'origine puis un nouveau *thread* est crée sur le site destination. Bettini a récemment proposé un premier procédé syntaxique [BN01] pour transformer des programmes qui utilisent la mobilité forte en programmes qui reposent sur de la mobilité faible pour les migrations.

1.4.2 Intergiciels

Un agent mobile nécessite un environnement d'exécution spécifique sur tous les sites potentiels de son itinéraire. Un tel environnement doit permettre au minimum la création des agents, leur exécution et leurs migrations.

1.4.2.1 Standard

La plupart des plates-formes d'agents mobiles ont introduit un ensemble de concepts similaires qui ont servi d'apports dans la standardisation des intergiciels agent. Cette standardisation, élaborée d'une part par l'OMG (c.-à-d. le standard MASIF [MBB+98] (*Mobile Agent System Interoperability Facility*), proposé en 1997) et d'autre part la FIPA[9] (*Foundation for Intelligent Physical Agents*), a donné lieu à la définition d'une plate-forme abstraite dont les éléments sont :

1. les agents qui sont définis par un identifiant, leur état et leur code. Ils représentent l'entité de base pour la structuration des applications.

2. les agences (*agencies*) qui représentent les environnements d'exécution sur un site. Une agence assure l'exécution, l'administration, le contrôle, le transport et les communications des agents. Des agences peuvent être regroupées au sein d'un même domaine (*region*), tel qu'un Intranet, afin de faciliter les opérations d'administration. Une région peut ainsi être associée à une seule autorité, par exemple pour fournir une politique de sécurité commune. Chaque agence comprend une ou plusieurs places. Une place est caractérisée par une machine virtuelle exécutant les agents. Elle représente l'abstraction de base correspondant à un nœud de ressources, qui est une entité plus fine que le site hôte[10]. Les agents se déplacent de place en place et peuvent interagir avec les services offerts par la place ou avec d'autres agents.

3. un collection de services intergiciels et d'interfaces construits au-dessus du standard CORBA, tels que :

   - la localisation d'agents afin de les contrôler et permettre qu'ils se retrouvent pour interagir.
   - l'administration afin de créer, détruire, suspendre ou copier les agents.
   - le transport pour transférer les agents suivant différents protocoles.
   - la communication pour permettre les interactions entre une agence et les agents, indépendamment de leurs localisations.
   - la sécurité afin de protéger les places des accès non autorisés (p.ex. contrôle d'accès), de signer les agents ou de chiffrer les données.
   - la persistance pour sauvegarder périodiquement les états des agents.

9. Le standard FIPA s'attache plus particulièrement à la coopération entre les agents.
10. Dans la suite du document, nous utilisons indifféremment le terme site pour caractériser une place, hormis dans le chapitre 5.3, où la terminologie liée aux plates-formes à agents mobiles est conservée.

1.4.2.2 JINI

Il existe des intergiciels qui tirent partie de la mobilité du code sous une forme différente de celle des plates-formes à agents mobiles présentées. C'est le cas notamment de l'infrastructure JINI [AWO+99] (*Java Intelligent Network Infrastructure*), qui est un intergiciel pour systèmes dynamiques, coopératifs et spontanés, porté vers l'informatique nomade. JINI est strictement orienté service et est construit autour de l'environnement et du langage Java. Un service peut être matériel (p.ex. ressources de calcul, stockage, imprimante, fax) ou bien logiciel (p.ex. correcteur, conversion de formats, achat en ligne, gestion de stock). Cette technologie permet de découvrir et connecter dynamiquement des services distribués sur un réseau en utilisant un service de recherche sous la forme d'un *broker*. Il est possible d'ajouter, supprimer ou contacter un service dans la base de recherche qui est organisée sous la forme d'un JavaSpace (cf. section 1.1 page 9). Ce service a en charge de retrouver les services et d'assurer le lien. Le mécanisme d'interaction est basé sur l'invocation distante Java, où les objets d'une machine virtuelle cliente communiquent avec des objets d'une autre machine par réception des objets *proxy* qui mettent en œuvre l'interface du service demandé. Ces objets mobiles peuvent introduire de nouveaux codes dans les processus où ils sont déplacés. Le plus souvent, ils retransmettent simplement les appels au service qu'ils représentent. JINI n'est pas une technologie à agents mobiles, bien qu'il puisse être utilisé pour développer un tel système. L'infrastructure ne supporte que la mobilité à itinéraire simple (c.-à-d. une seule migration) et s'appuie en quelque sorte sur l'abstraction du code à la demande. Un client contacte le service de recherche afin de récupérer la liste des services enregistrés sur le réseau. Il invoque alors à distance le service désiré et récupère ainsi le *proxy* du service sous la forme d'un objet mobile. Les objets sont donc déplacés sur les clients. Si les clients disposent de peu de ressources de calcul ou de mémoire, les performances peuvent être mauvaises.

1.5 Discussion

Des méthodes de conception et de développement, orientées objets, se sont imposées pour améliorer le temps de développement, la maintenance et la réutilisation des systèmes ou applications distribués. Désormais, une abstraction conceptuelle supplémentaire basée sur l'utilisation de composants autonomes tend à se généraliser. Cette approche favorise la construction d'application par intégration de composants existants (*programming in the large*). Dans les architectures de services (p.ex. service Web), le couplage entre les composants repose sur l'invocation distante, basée sur le schéma d'organisation client-serveur, qui reste le plus répandu au sein des intergiciels. Ce schéma facilite le développement des applications en favorisant la réutilisation de services existants. Les interactions sont majoritairement synchrones, bidirectionnelles et se limitent à la participation de deux processus, le client et le serveur. Grâce à la mise en commun de services, le schéma client-serveur permet de faire cohabiter dans un même système des sites hétérogènes en termes de ressources. Cependant, ce schéma ne permet de décrire l'interaction qu'entre deux processus et repose sur la seule mobilité des données. Pour le concepteur se pose le problème de répartition des traitements entre les clients et les serveurs. Quels traitements peut-on laisser aux clients et quels sont ceux que l'on va déporter sur des serveurs sous la forme de services? Ce choix influe particulièrement sur les propriétés non fonctionnelles du système final. Par exemple, la justesse de la répartition permet d'éviter de trop nombreux messages entre les clients et les serveurs. Une première heuristique lors de l'étape de conception est de placer statiquement les traitements exclusifs (c.-à-d. particuliers)

sur les clients et de distribuer sous la forme de services les traitements communs.

Nous avons présenté les différentes technologies à base de code mobile, vus comme des variations du schéma client-serveur [FPV98, Pic98]. Le code mobile permet de déplacer le savoir-faire entre les sites une fois le système développé. Les agents mobiles sont certainement un mécanisme d'interaction à fort potentiel. Ils traitent la distribution en interne et ont donc la capacité de migrer de manière autonome entre plusieurs sites (c.-à-d. itinéraire). Ils ouvrent ainsi la possibilité à un client de réaliser un service complexe (c.-à-d. composition de services primitifs) où des traitements exclusifs [Per97] sont déportés à travers le réseau.

Chapitre 2

Avantages/inconvénients des agents mobiles et modèles

Les propriétés de performance, de sécurité et de sûreté de fonctionnement sont essentielles et doivent être prises en considération au plus tôt dans le procédé de construction des applications distribuées [CDK01]. Elles sont particulièrement liées au type d'applications traitées. Par exemple, des applications de commerce électronique ont des besoins forts en terme de sécurité. Ces besoins peuvent être traités en partie par le système de communication. Ils le sont de plus en plus par des services intergiciels qui permettent de développer un système distribué sans se soucier directement du réseau et de ses contraintes (p.ex. latence, bande passante, sécurité, pannes). Bien que les intergiciels actuels proposent une variété de services spécifiques pour chercher à garantir/optimiser des propriétés de qualité, la mise en oeuvre des mécanismes associés induit souvent des dégradations au détriment d'autres propriétés.

Toute application mise en œuvre à partir d'agents mobiles peut aussi bien l'être avec des interactions plus traditionnelles telle que l'invocation distante. Les bénéfices potentiels des agents mobiles sont liés aux critères non fonctionnels. Dans le cadre des architectures de services, nous sommes convaincus que l'utilisation des agents mobiles, en plus des interactions par invocation distante, permet de concevoir des systèmes hybrides aptes à optimiser des propriétés de qualité. Le fait de considérer les propriétés de qualité plus en amont dans le procédé de construction peut favoriser leur traitement. L'argument *End2end* de Saltzer, Reed et Clarke [SRC84] va en ce sens. Dans un contexte intergiciel, il stipule qu'il n'est pas toujours adéquat de sous-traiter les propriétés de qualité intégralement dans les couches systèmes. Il est parfois plus favorable de les traiter directement au niveau applicatif. Au regard des propriétés de qualité, il n'est pas toujours bénéfique de considérer une vue idéale où le réseau est fiable, de latence nulle, de bande passante infinie, sécurisé, avec une topologie fixe et administrée par un seul individu. Ainsi, nous considérons que l'architecte et le concepteur doivent être conscients de la localisation des différentes services du système [WWWK97, Pap00]. La qualité du système construit dépend directement de leurs localisations. Une connaissance a *priori* de la localisation des services peut ouvrir la voie à la sélection «optimale» d'interactions, au plus tôt dans les étapes du procédé de construction.

De nombreux arguments quant à l'utilisation des agents mobiles ont dans un premier temps été présentés au sein de la communauté système travaillant sur le code mobile [LO99]. La validité de ces arguments est rarement expérimentée dans des applications réalistes. La question «pourquoi utiliser des agents mobiles?» a du mal à trouver une réponse définitive. En

effet, la plupart des travaux sur ce domaine se sont focalisés sur l'aspect technique. Les agents mobiles supportent les **opérations déconnectées** grâce au mode d'interaction asynchrone. Les agents opèrent et restent actifs sans nécessiter une liaison permanente avec le client. Bien que des compagnies télécom s'attachent régulièrement à installer de nouvelles infrastructures de communication, ces efforts ne suffisent pas toujours à garantir la qualité des connexions (p.ex. connectivité). Beaucoup d'utilisateurs sont soumis à de fréquentes déconnexions (p.ex. unités portables mobiles dans les réseaux sans fils). L'agent est indépendant du site client qui peut se déconnecter puis se reconnecter afin de récupérer ultérieurement l'agent et ses résultats.

Dans les applications distribuées basées sur le schéma d'organisation distribuée client-serveur, un service accessible par les clients est fixé a *priori* à travers une interface définie statiquement. Il arrive qu'un service ne soit pas entièrement adapté aux besoins d'un client. Pour les besoins spécifiques d'un client, le fournisseur du service doit donc envisager l'introduction de nouvelles fonctionnalités sur le serveur. Cependant, les services sont le plus souvent proposés pour un grand nombre de clients et ne sont pas personnalisés pour chaque client. Les agents mobiles permettent la personnalisation des services présents sur un réseau [Per97]. Un client peut adapter un service à ses besoins personnels en l'étendant dynamiquement à l'aide de traitements spécifiques, encapsulés dans le code d'un agent. Le service réalisé par un agent pour le compte de son client n'est donc plus limité aux simples fonctionnalités des services offerts sur un serveur. Le service étendu (c.-à-d. le service ne se restreint pas à une interaction bidirectionnelle comme dans le cas du client-serveur) apporte ainsi de la **flexibilité**[1] au service primitif.

Les agents mobiles favorisent, dans une certaine mesure, la **capacité de croissance** d'un système. En effet, les déports d'exécution facilitent la répartition de charge. De même, les agents favorisent parfois le **passage à l'échelle**, particulièrement quand la taille des données offertes par un service tend à s'accroître considérablement. À l'aide d'interactions locales et de traitements *in situ*, cette augmentation tend à être moins sensible.

Les principales propriétés de qualité qui suggèrent l'utilisation ou non des agents mobiles dans le domaine des architectures de services portent sur la **performance**, la **sécurité** et la **fiabilité**. Dans la suite de ce chapitre (sections 2.1, 2.2 et 2.3), pour chacune de ces propriétés, nous rappelons les concepts associés et passons en revue les travaux notoires qui y sont liés dans le cadre des agents mobiles. Notre but ici n'est pas d'établir un catalogue exhaustif et détaillé des différents travaux, mais plutôt de présenter les différents concepts qui seront utilisés par la suite dans le document et dont nous étudierons les impacts.

Ensuite, dans la section 2.4, nous passons en revue les travaux notoires qui portent sur les modèles pour systèmes à agents mobiles. La dernière section discute les points étudiés dans ce chapitre au regard de la problématique de cette thèse.

1. Nous utilisons par la suite le terme flexibilité au sens de la capacité de personnalisation/adaptation suite aux traitements déportés.

2.1 Performance

Nous nous plaçons ici dans un cadre simplifié en considérant la performance comme directement proportionnelle au volume de données à faire transiter sur un réseau. Plus ce volume de données est petit, meilleures seront les performances du système final. Dans une organisation client-serveur classique, le choix du placement des éléments (p.ex. client, services) sur les sites du réseau et leurs rôles influent particulièrement sur la performance. La technologie agent mobile peut être intégrée dans les architectures d'applications. Les agents mobiles peuvent réduire la consommation en bande passante en déportant les calculs vers les données (c.-à-d. déplacer la logique des interactions vers la source) plutôt que de rapatrier ces données pour les traiter localement à la manière d'une invocation distante (p.ex. RPC ou RMI). Les agents peuvent alors interagir en local avec un service. Si le service impose des interactions multiples, le trafic réseau et la latence s'en retrouvent largement réduits. Quelques travaux ont cherché à comparer les invocations distantes avec les agents mobiles en terme de performance. Certains sont quantitatifs et validés uniquement pour une application donnée sur un réseau fixé. Par exemple, Johansen [Joh98] a comparé les performances entre les interactions par RPC et par un agent sur des applications de surveillance météo et de récupération d'images satellites. Sahai et Morin [SM98] ont mesuré la consommation en bande passante et les temps de réponse dans une application de gestion de réseau. Ismail et Hagimont [IH99] ont étudié les temps d'exécution pour de la collecte d'informations dans une base de données. À l'aide de tests de performance, ces trois travaux (entre autres) ont conforté l'idée que l'usage de la technologie agent mobile pouvait être bénéfique en terme de performance par rapport à des invocations distantes.

D'autres études (c.-à-d. [CPV97, CK97, SS97]) se sont plus portées sur le côté analytique afin de comparer les schémas d'organisation. Nous présentons ici ces études/analyses de performance (sur les volumes de données) vouées à guider le choix de l'utilisation d'invocations distantes et/ou d'agent mobiles. Dans les trois études proposées, les hypothèses portent sur un réseau fiable uniforme, où le coût des communications ne dépend pas de la distance entre les sites. Tous les sites sont accessibles à partir de tout nœud du réseau, sans contrôle d'accès ni procédure d'authentification. Les mécanismes de sécurité et de tolérance aux fautes qui influent directement sur la taille des messages ne sont donc pas pris en compte. Le coût des communications est déterminé par le nombre d'octets transmis sur le réseau (les temps de réponse ne sont pas présentés ici). Pour évaluer ce coût, les différentes études s'appuient sur des objets/données communs mais identifiés différemment. Nous proposons ici un cadre uniforme afin de ne pas « embrouiller» le lecteur. Dans les analyses proposées, un client/agent accède à des services sous un mode requête-réponse :

- un service s_i fournit un résultat de taille $Out(s_i)$ à partir d'une requête de taille $In(s_i)$.
- la donnée en entrée d'un service $In(s_i)$, au cour d'une interaction, peut correspondre à la donnée rendue par le service précédent $Out(s_{i-1})$ (services composés).
- pour tout s_i, $In(s_i)$ et $Out(s_i)$ sont des constantes. Ces tailles de données sont toutes prédéfinies pour les analyses.
- la taille du code d'un agent est supposée constante durant la vie de l'agent et est notée $source$.

2.1.1 Étude d'une application de collecte de documents

Carzaniga, Picco et Vigna [CPV97] ont proposé d'analyser les consommations en bande passante d'une application de collecte de documents (pour du *data mining*), mise en œuvre selon trois schémas d'organisation. Dans l'application, un client souhaite récupérer des documents de taille constante b (*body*), en interagissant avec n sites. Tous les documents ont la même taille. Chaque site dispose d'un même nombre D de documents. Pour chaque site, le client récupère dans un premier temps les en-têtes des documents, de taille h (*header*). Un service s_i (sur chaque site) permet de récupérer ces en-têtes des documents, c.-à-d. $Out(s_i) = D * h$. Le client sélectionne ensuite, à partir des en-têtes, les documents qu'il souhaite télécharger. Le service s'_i permet alors de récupérer les corps des documents, c.-à-d. $Out(s'_i) = D * b$. Les documents sont supposés uniformément répartis entre les sites. k ($0 \leq k \leq 1$) représente le ratio entre les documents à télécharger et le nombre de documents présents sur le site (sélectivité). Les requêtes émises par le client ont une taille fixe r (demande d'en-tête et demande de document), c.-à-d. pour tout i, $In(s_i) = r$ et $In(s'_i) = r$.

Les interactions par RPC, évaluation distante et agent toujours mobile ont été évaluées (cf. Figure 2.1), quand les n sites proposent les mêmes services :

1. pour des interactions à base d'invocation distante (cf. section 1.2), le client interagit à distance avec les n sites. Pour chaque site, il émet dans un premier temps D requêtes de taille r afin de récupérer les en-têtes de documents (taille $Out(s)$). Ensuite, suivant le ratio k fixé, il émet D requêtes afin de récupérer les corps de documents désirés. Le coût suivant ce schéma est :

$$n(D * r + Out(s) + k(D * r + Out(s'))$$

2. en utilisant des évaluations distantes (cf. section 1.3.1), il devient possible de choisir les documents à télécharger directement sur le site distant (la sélection se fait à distance). Ainsi, pour les n sites, le client émet du code exécutable (sélectivité *source*) et sa requête r, puis récupère seulement les documents désirés, de taille $k * Out(s')$. Le coût induit est donc moindre qu'avec des invocations distantes si la taille de *source* est faible par rapport à la taille des en-têtes de documents et des requêtes. Le coût suivant ce schéma est :

$$n(source + r + k * Out(s'))$$

3. l'utilisation d'un seul agent mobile (cf. section 1.3.3) pour parcourir séquentiellement les n sites induit un coût plus important que dans les deux cas précédents (l'itinéraire est indifférent vu que tous les sites sont identiques). En effet, la requête et le code de l'agent migrent $n + 1$ fois (les n sites et le retour chez le client), ce qui est sensiblement équivalent à l'évaluation distante (n fois). Toutefois, les documents récupérés par l'agent en local sont placés dans son sac à dos au fur et à mesure de son itinéraire. Ainsi, les résultats du premier service s'_1 vont transiter à travers les $n-1$ sites suivants avant d'être retournés au client. Au cour de la $(i+1)$nième interaction, les documents accumulés par l'agent ont une taille de $i * k * Out(s')$. La taille des documents en transit correspond à la somme de 1 à n du coût de ces migrations i. Le coût suivant ce schéma est :

$$(n + 1)(source + r + k(n/2) * Out(s))$$

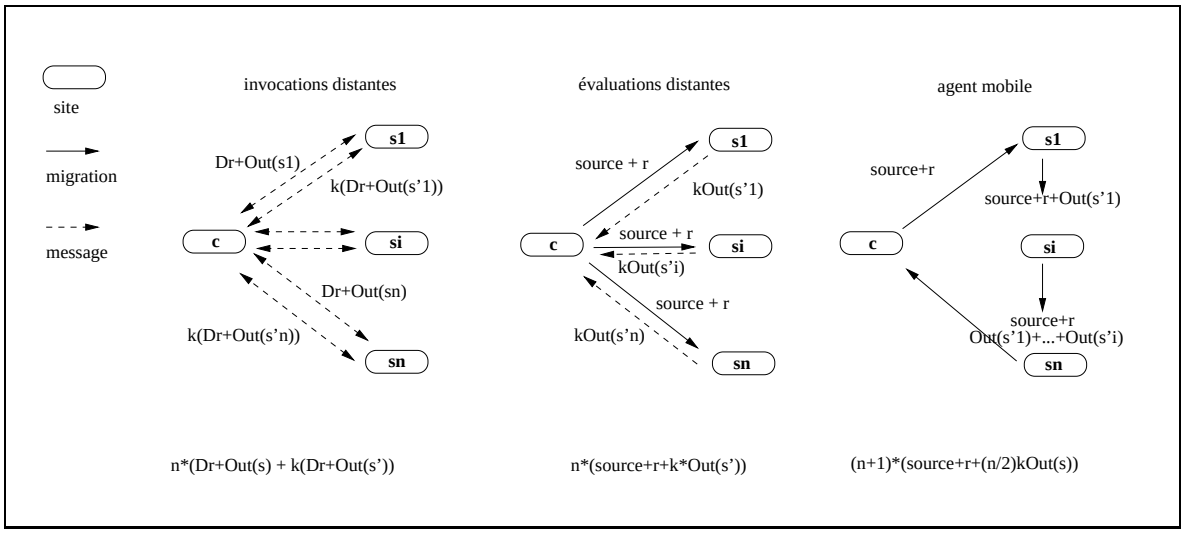

FIG. 2.1 – *Différents coûts pour de la collecte de documents*

Cette première analyse permet de raisonner sur la meilleure technologie à exploiter pour l'application traitée. Pour cette application et avec les hypothèses données, les auteurs montrent que les schémas client-serveur (par invocation distance, ici RPC) et évaluation distante sont nettement plus favorables que l'utilisation d'un agent mobile (problème du cumul des documents sur l'itinéraire).

2.1.2 Étude de la mobilité des agents

L'étude précédente s'est restreinte à un agent qui ne réutilise pas les données accumulées et qui interagit toujours localement suivant un itinéraire fixé (toujours mobile). Chia et Kannapan [CK97] ont proposé trois études de cas afin de comparer le nombre d'octets transmis sur un réseau pour une application de transformation de données implantée à l'aide d'un agent. L'application étudiée par les auteurs porte sur l'utilisation de deux services s_1 et s_2 qui fournissent un résultat à partir d'une requête. Á la différence de l'analyse précédente, les deux services s_1 et s_2 sont donc composés. La donnée en entrée du service s_2 ($In(s_2)$) au cour d'une interaction correspond à la donnée rendue par le service précédent ($Out(s_{i_1})$). Initialement, les données du client sont $In(s_1)$.

Le cas d'interactions multiples avec un service est envisagé. Toutefois, la donnée fournie en résultat est réutilisée directement en tant que nouvelle entrée du service. Ainsi, interagir m fois avec un service s_i par invocation distante engendre une taille de donnée $Isize(s_i)$ égale à m fois la somme de $In(s_i)$ et $Out(s_i)$. $Isize$ est le volume total du aux m interactions par invocation distante à travers le réseau.

$$\forall i,\ Isize(s_i) = m(In(s_i) + Out(s_i))$$

Avec ces informations, trois interactions sont étudiées, au regard du trafic engendré (voir Figure 2.2) :

1. agent toujours stationnaire : les interactions se font à distance entre les sites (invocation distante). Le trafic généré est équivalent à la somme des m interactions avec s_1 et s_2,

soit $Isize(s_1)$ et $Isize(s_2)$. Le coût de ce schéma est donc :

$$Isize(s_1) + Isize(s_2)$$

2. agent toujours mobile : un agent, initié par le site client, migre successivement vers le site prestataire de s_1 puis celui proposant s_2 avant de retourner les résultats sous la forme d'un message à son client. Le trafic engendré sur le réseau correspond ainsi à deux fois la taille du code de l'agent ($source$), la donnée initiale du client $In(s_1)$, auxquels s'ajoutent les résultats fournit par s_1 et s_2 (c.-à-d. $Out(s_1)$ et $Out(s_2)$). Il n'y a pas cumul des données de l'agent, p.ex. $Out(s_1)$ est consommé sur le site prestataire de s_2. Le volume de données $Isize$ produit par les m interactions n'est pas à prendre en compte, celles-ci se faisant en local. Le coût de ce schéma est :

$$2 * source + In(s_1) + Out(s_1) + Out(s_2)$$

3. solution mixte : les auteurs ont étudié le cas où l'agent migre vers le site de s_1, interagit à distance avec s_2 et fournit finalement le résultat sous la forme d'un message à son client. Dans ce troisième cas, l'agent a la possibilité de s'approprier des ports de communication sur un (des) site(s) de son itinéraire pour interagir par invocation distante (p.ex. synchrone) avec des services localisés sur d'autres sites. Le trafic engendré correspond à une seule fois la taille du code de l'agent, le coût de l'interaction distante avec s_2 et finalement le résultat de s_2 (retransmis au client). Le coût de cette interaction mixte est :

$$source + In(s_1) + In(s_2) + 2 * Out(s_2)$$

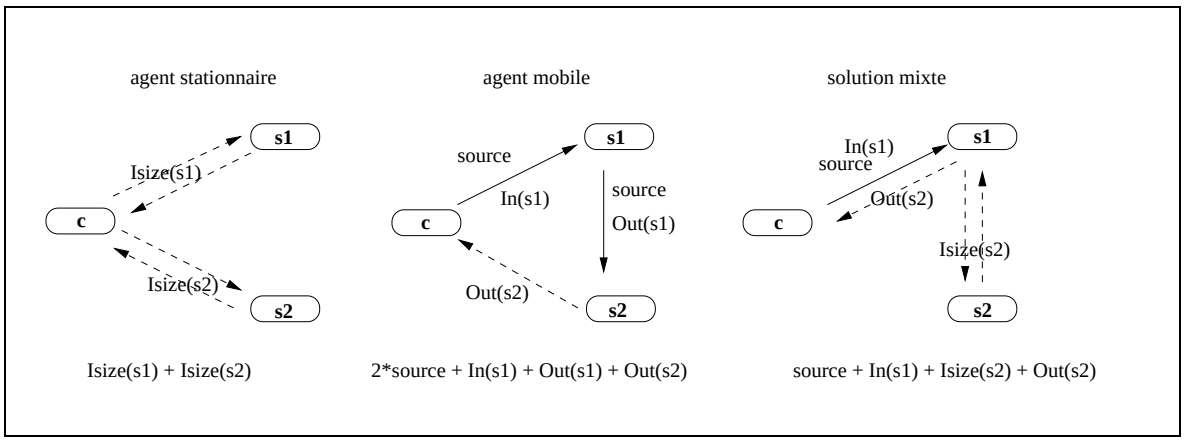

FIG. 2.2 – *Étude de trois stratégies de mobilité*

Cette étude de cas, pour deux services composés, avec l'analyse statique des coûts associée (tailles de données et de l'agent préfixées) a été validée dans un prototype logiciel qui a permis de démontrer que l'un des cas extrêmes, toujours mobile ou toujours stationnaire, n'est pas nécessairement optimal pour l'application traitée. Si $m = 1$ (c.-à-d. $Isize(s_2) = In(s_2) + Out(s_2)$), l'agent stationnaire est défavorable par rapport à l'agent mobile quand la taille de $source$ est plus petite que $In(s_2)$. L'agent mobile est défavorable par rapport à la solution mixte quand la taille de $source$ est plus importante que $Out(s_2)$. Le cadre analytique peut s'étendre à plusieurs services composés, bien qu'à notre connaissance cela n'ai pas été proposé par les auteurs.

2.1.3 Analyse de Straßer et Schwehm

Straßer et Schwehm [SS97] ont proposé une étude qui intègre des particularités des deux études précédentes. Elle s'applique à n services non composés et traite des solutions mixtes. Dans les analyses, deux services s_i et s_j peuvent être localisés sur un même site S. L'étude se limite à un seul agent mobile, disposé à réaliser des invocations distantes au cour de son itinéraire (solution mixte). Cet agent a donc la possibilité, en plus de migrer à travers un site pour utiliser des services locaux, de s'accaparer des ressources (p.ex. ports) pour interagir avec des services extérieurs à ceux proposés en local. L'agent peut rester chez le client (toujours stationnaire), migrer à chaque étapes (toujours mobile), ou être mixte. En ce sens, cette étude tend à devenir un cadre analytique qui permet de comparer différentes mises en œuvre d'un agent devant interagir avec n services indépendants. Ce cadre permet d'identifier les situations où l'utilisation d'un agent mobile est avantageuse par rapport à des invocations distantes. Les auteurs ont développé des équations concernant le trafic et les temps d'exécution pour des stratégies mixtes d'invocations distantes et d'agents mobiles (les cas toujours stationnaire et toujours mobile étant des cas particuliers).

Pour une invocation distante entre les deux sites S_1 et S_2, la définition du coût en terme de volume de données, connaissant la taille de la requête In et du résultat Out, est :

$$Perf_{RPC}(S_1, S_2, In, Out) = \ 0 \ si \ S_1 = S_2, \ In + Out \ sinon$$

Pour un agent mobile qui migre d'un site S_1 vers un site S_2 (distinct), le volume de données engendré est la taille du code de l'agent, les données qu'il transporte (In) et la taille des résultats. L'équation prend en compte la réduction de la taille du résultat d'un service (Out) à l'aide d'un facteur σ ($0 \leq \sigma \leq 1$). Ce facteur correspond à un traitement délocalisé de l'agent (sélectivité de l'agent). Si l'agent ne modifie pas le résultat, $\sigma = 1$.

$$Perf_{AM_{step}}(S_1, S_2, In, Out) = \ 0 \ si \ S_1 = S_2, \ source + In + (\sigma * Out) \ sinon$$

Pour une interaction avec un unique service, ces deux équations ont été validée expérimentalement et ont montré qu'un agent mobile est plus favorable qu'une invocation distante si la taille du résultat du service est importante (Out) et que l'agent a une large sélectivité.

À partir des deux équations précédentes, l'approche à été étendue à une séquence fixée d'interactions avec n services ($Seq = (I_1, ..., I_n)$). Une interaction I_i est décrite statiquement par :

$$I_i = \{S_i, m_i, In(s_i), Out(s_i), \sigma_i\}$$

où S_i est le site avec lequel la communication a lieu. Chaque communication consiste en m_i invocation (locale ou distante) du service s_i avec la requête de taille $In(s_i)$ et le résultat de taille $Out(s_i)$ (résultat ensuite modifié par une sélectivité σ_i). Après une interaction, la taille d'un agent est (cumul des résultats) :

$$B_{A_i} = source + B_{data_i}, \ avec \ B_{data_i} = B_{data_{i-1}} + m_i \sigma_i Out(s_i)$$

B_{A_0} correspond à la taille du source de l'agent et des requêtes initiales ($In(s_i)$). Le coût d'une migration d'agent en cour d'itinéraire devient :

$$Perf_{AM_{cumul}}(S_1, S_2, B_{A_i}) = \ 0 \ si \ S_1 = S_2, \ source + B_{A_i} \ sinon$$

Pour un agent stationnaire, les interactions se font par invocation distante avec chaque sites de la séquence d'interactions. Les m échanges doivent être effectués et la sélectivité σ ne peut

être appliquée à distance. Pour un agent toujours mobile, les interactions se font en local avec chaque service. Les m échanges n'induisent pas de trafic supplémentaire et la sélectivité permet de faire décroître la taille du résultat de chaque service. Toutefois, les résultats sont accumulés le long de l'itinéraire. Pour permettre l'analyse d'un agent mixte, les auteurs utilisent un vecteur de n étapes ($E = (E_0, ..., E_n)$). Il représente l'itinéraire statique de l'agent, où chaque E_i est un identificateur de site élément de $\{S_1, ..., S_n\}$. Pour un agent stationnaire, ce vecteur contient n fois l'identificateur du site client. Pour un agent toujours mobile, il correspond exactement aux sites de la séquence d'interactions, avec la source $E_0 = site\_client$. Pour la $i$ème interaction (cf. Seq), l'agent migre vers la destination E_i (si $E_{i-1} \neq E_i$). Le coût pour un agent mixte à itinéraire fixé est calculé comme suit :

$$Perf_{AM}(Seq, E, B_{A_0}) = \sum_{i=1}^{n} (Perf_{AM_{cumul}}(E_{i-1}, E_i, B_{A_i}) + m_i Perf_{RPC}(E_i, S_i, In(s_i), Out(s_i)))$$

Le coût correspond aux coûts des migrations de l'agent sur l'itinéraire (vecteur d'étapes) plus les coûts des invocations distantes.

Cette analyse statique de performance a été validé par des mesures sur un scénario d'interactions avec quatre sites où de nombreux itinéraires d'agents ont été considérés. Les choix toujours stationnaire (p.ex. pas de sélectivité) et toujours mobile (p.ex. cumul de résultats) ne sont pas forcement optimaux, particulièrement en fonction des m_i, σ_i et de la taille du source d'un agent.

2.2 Sécurité

La sécurité [RG91] consiste à empêcher les accès non autorisés aux informations, où les sujets (p.ex. utilisateurs, processus, périphériques) et les objets (p.ex. ressources, données) sont distingués. Elle se décompose en trois sous-propriétés :

1. la confidentialité qui correspond au fait que l'information contenue dans les objets n'est pas divulguée à un sujet non autorisé.

2. l'intégrité qui correspond au fait que les objets ne sont pas modifiés de façon non autorisée.

3. la disponibilité de service qui correspond au fait que le service peut être accédé par tout sujet autorisé. La non-délivrance d'un service suite à une attaque incapacitante, de type déni de service, peut être vue comme un trou de sécurité. Cette propriété est également associée aux propriétés de sûreté de fonctionnement.

La sécurité est souvent assimilée aux seules propriétés de confidentialité et d'intégrité (la disponibilité nécessitant de prendre en compte des paramètres dynamiques, elle est plutôt associée aux propriétés de sûreté de fonctionnement). La mise en œuvre des propriétés de sécurité [BI95] s'appuie sur des mécanismes d'authentification (p.ex. service *Passport* de .NET), de cryptographie (p.ex. SSL) et de contrôle d'accès [McL94] (p.ex. barrière de protection ou pare-feu). L'authentification permet de certifier l'identité d'un sujet. La cryptographie est l'étude des algorithmes et protocoles utilisés pour préserver la confidentialité de l'information et garantir son intégrité. Elle s'appuie sur la cryptologie qui est un ensemble de techniques qui permettent de protéger des informations grâce à du chiffrement (c.-à-d. symétrique à clé

secrète ou asymétrique à clé publique). Le contrôle d'accès, mis en œuvre dans un système, permet de vérifier qu'un sujet donné a le droit de communiquer avec un autre sujet et qu'il a le droit d'accéder à des données du système.

Dans les systèmes distribués client-serveur, les problèmes de sécurité ont des solutions partielles largement employées. Afin de garantir au mieux les propriétés de confidentialité (non-divulgation) et d'intégrité (non-modification) des objets, les données et les ressources critiques sont localisées sur des machines sécurisées (matériel ou logiciel). La protection peut être réalisée en sécurisant les objets, les ressources (menaces endogènes) et les canaux de communication (menaces exogènes).

La sécurité est un des problèmes les plus délicats posés par les agents mobiles [Ord96]. Cette propriété de qualité est souvent citée [Joh00] comme la raison majeure qui en a limité la diffusion. En effet, les agents mobiles posent de nouveaux problèmes de confidentialité, d'intégrité et de disponibilité. Ces problèmes peuvent se diviser en deux catégories. D'une part, le besoin de protéger les sites d'accueil des agents contre du code malveillant et d'autre part la protection des agents (p.ex. leur code, leur état et leurs données) envers des hôtes mal intentionnés. La première catégorie de problèmes commence à être relativement bien traitée alors que la seconde est intrinsèquement difficile et les solutions se restreignent pour l'instant à des travaux de nature théorique.

2.2.1 Protection d'un site hôte envers un agent mobile

Un système d'exécution, s'il n'est pas protégé, devient vulnérable dès qu'il exécute du code venant de l'extérieur. Un agent, s'il a accès à la mémoire ou aux fichiers de la machine hôte, peut facilement violer les propriétés de confidentialité (p.ex. lecture) ou d'intégrité (p.ex. écriture). L'agent peut également s'attaquer à la disponibilité des systèmes hôtes. Par exemple, il peut bloquer le système en réalisant des opérations non conformes, consommer de manière illimitée et incontrôlée des ressources, se cloner indéfiniment ou bien encore migrer sans fin.

Un premier moyen de contrer ces problèmes de sécurité consiste à s'appuyer sur une authentification de l'agent avant son exécution. L'environnement d'exécution, connaissant la provenance de l'agent, est alors capable de vérifier le niveau de confiance de l'agent reçu en fonction de son origine. Il limite, à sa convenance, les droits d'accès aux ressources et données locales. Toutefois, même du code issu d'un site supposé sain (domaines de confiance) peut contenir des erreurs non intentionnelles qui peuvent se répercuter sur la sécurité du système hôte. D'autre part, cette approche, en limitant la provenance des agents à quelques clients connus, est très restrictive.

Trois techniques peuvent être employées pour contrer les agents malveillants :

1. les **interpréteurs sûrs** représentent la solution la plus largement utilisée pour résoudre le problème du code malveillant. Une machine virtuelle permet l'utilisation d'un gestionnaire de sécurité qui contrôle/vérifie chacune des instructions et choisit de les exécuter ou non. Différents interpréteurs gèrent la sécurité (p.ex. Safe-Tcl, Ocaml, Java). Ils peuvent être utilisés dans le cadre de plates-formes à agents mobiles afin de proposer une machine virtuelle protégée pour l'exécution des agents. Plus particulièrement, la machine virtuelle Java utilise une vérification statique du code (p.ex. syntaxe, typage) et un gestionnaire de sécurité afin de contrôler les instructions du *byte code* Java. Les accès aux ressources critiques sont ainsi contrôlés, à la manière d'un pare-feu local. Toutefois, ces mécanismes ne sont pas infaillibles quand l'interpréteur n'est pas prouvé (cf. les nombreuses corrections qui ont été apportées au sein de l'interpréteur Java). La sécurité du

système hôte est alors entièrement dépendante de la correction de la politique mise en œuvre par l'interpréteur. De plus, la sécurité dépend du programmeur local du site qui a en charge de définir les permissions ou encore les insertions de contrôles.

2. la technique de l'**isolation** ou de confinement (*Software Fault Isolation*), permet de charger le code non sûr à exécuter dans un espace d'adresse restreint (c.-à-d. bac à sable) [WLAG93]. Le code reçu par le site hôte est instrumenté (par le site ou auparavant) puis vérifié au chargement, afin de garantir que chaque chargements, sauvegardes ou instructions de sauts se font dans une adresse du bac à sable. Ainsi, les accès mémoire sont confinés dans un espace et les données privées du site sont protégées. Les appels systèmes sont traités à part *via* une table de sauts.

3. la **vérification** automatique du code de l'agent sur le site hôte avant son exécution s'appuie sur la vérification d'une preuve de conformité. Lee et Necula [NL98] ont proposé une approche (PCC, *Proof Carrying Code* ou code auto-certifié) où le propriétaire de l'agent se doit de produire une preuve formelle que son agent garantit des propriétés de sécurité définies par un site hôte. Le code et la preuve sont envoyés simultanément sur le site hôte. L'agent migre donc avec des preuves de propriétés fixées qu'il satisfait. L'hôte vérifie que la preuve est formellement correcte et exécute l'agent seulement s'il est en accord avec la politique. Cette approche se restreint à des propriétés de bon typage dont la preuve est produite par le compilateur (dit certifiant).

2.2.2 Protection d'un agent envers un site hôte

Les environnements d'exécution des agents sur les sites doivent avoir accès au code et état des agents afin de pouvoir les exécuter. Durant l'exécution sur un serveur, le code, l'état et les données de l'agent sont donc exposés à des problèmes de confidentialité et d'intégrité. Les attaques possibles ne sont pas encore toutes clairement identifiées [Hoh98a].

Pour la confidentialité, un site malveillant peut examiner un agent (p.ex. code, données) afin de découvrir des informations privées transportées par l'agent. L'étude du flot de contrôle de l'agent ou de son itinéraire peut permettre de déduire de l'information sur des états antérieurs de l'agent. Un site peut exploiter ces informations pour modifier le comportement de ses services.

En ce qui concerne l'intégrité, (i) le site hôte peut rejouer l'exécution d'un agent, cloner l'agent ou incorrectement exécuter son code. Il peut aussi fausser les résultats des appels systèmes faits par l'agent et ainsi duper l'agent. (ii) Le site hôte peut modifier le code (p.ex. itinéraire), l'état (p.ex. flot de contrôle) ou les données d'un agent. Ces modifications peuvent générer des actions malveillantes envers des hôtes que l'agent visitera ultérieurement, afin de monter une attaque. Si une plate-forme réceptrice ne peut pas déterminer si un agent accueilli a été falsifié, il convient alors de le traiter comme un agent anonyme et lui donner des permissions minimales.

La propriété de disponibilité peut être affectée en terme de refus d'exécution. L'environnement d'accueil d'un agent peut par exemple décider à tout moment de terminer brutalement l'exécution de l'agent ou empêcher sa migration.

Les solutions à ces divers problèmes de sécurité peuvent être matérielles ou bien logicielles. Dans le cas des agents limités à une seule migration (c.-à-d. un aller-retour client-serveur), il est possible d'établir une confiance bilatérale par authentification (c.-à-d. protection des portions

de code source d'un agent contre les modifications en utilisant des signatures). Cependant, l'utilisation d'agents itinérants (c.-à-d. migration entre plusieurs sites) complique le problème.

La solution la plus triviale pour le propriétaire d'un agent est de limiter son itinéraire à des sites de confiance. La seule réelle défense concrète des agents contre des sites malveillants provient du domaine du matériel. Cette approche [WBS98, WSB99] impose d'exécuter les agents exclusivement sur du matériel prouvé bienveillant par un tiers (c.-à-d. label avec audit indépendant, certification AFAQ). Un tel composant matériel dispose d'une machine virtuelle servant d'environnement d'exécution. Ce composant est connecté à un service offert sur le réseau et est régulièrement vérifié par le tiers.

Pour les approches logicielles, du fait de l'exécution de l'agent par le site hôte, il paraît impossible de protéger l'agent. Il existe cependant des solutions partielles qui traitent des sites mal intentionnés afin de garantir certaines propriétés de sécurité. Par exemple, il est possible de chiffrer des données d'agents afin de les protéger sur un itinéraire. Young et Yung [YY97] ont proposé un moyen de chiffrer les données accumulées (de taille raisonnable) et non-réutilisées sur l'itinéraire d'un agent. L'agent peut alors traverser des sites douteux en gardant la confidentialité/intégrité de petites données accumulées. La technique se base sur un chiffrement à clé publique des données sensibles. L'algorithme de chiffrement et la clé sont définis dans le code de l'agent. La donnée chiffrée ne peut être déchiffrée que sur le site du client (source de l'agent) qui seul possède la clé privée. La clé privée ne circule donc pas avec l'agent.

L'utilisation de signatures permet de prouver qu'un site a été néfaste envers un agent. Ces solutions portent principalement sur l'intégrité et consistent à prouver, a *posteriori* qu'il y a eu falsification de l'agent en faisant participer les serveurs.

1. Vigna [Vig97] suggère l'utilisation d'une **trace d'exécution** des agents afin de détecter, a *posteriori*, les falsifications de l'exécution. Quand un serveur a terminé l'exécution d'un agent, il chiffre la trace d'exécution (c.-à-d. hachage de la trace) pour la transmettre au propriétaire de l'agent. Il est nécessaire, pour chaque plate-forme de l'itinéraire d'un agent, de créer et sauvegarder une trace des actions réalisées par l'agent. L'agent voyage avec ses traces d'exécution. Ce mécanisme permet de détecter les manipulations d'agent et de s'assurer des exécutions conformes au code. L'agent peut ainsi être protégé en terme d'intégrité (p.ex. modifications non autorisée) mais pas en terme de confidentialité. Un site hôte peut « inspecter » (c.-à-d. violation de la propriété de confidentialité) les traces d'exécution réalisées par l'agent sur les sites précédents si elles ne sont pas chiffrables. Ces traces paraissent difficilement chiffrables selon la méthode de [YY97] (taille). L'inconvénient de cette approche est qu'elle nécessite de garder de nombreuses traces de tailles importantes. En cas de suspicion la détection est utilisée. Dans certains cas, la détection a *posteriori* des falsifications n'est pas satisfaisante, notamment quand des représailles ne sont pas possibles.

2. l'**authentification partielle des résultats** [Yee99] permet de détecter, a *posteriori*, la falsification sur les données. Cette méthode utilise des codes d'authentification partielle (PRAC). Un agent, visitant n sites dans son itinéraire, est envoyé par le client avec n clés secrètes $k_1, ..., k_n$. Sur un serveur i, l'agent utilise sa clé k_i pour signer les résultats de son exécution, produisant ainsi un PRAC. Il détruit ensuite sa clé k_i avant de migrer vers le serveur $i + 1$. Un serveur malhonnête de rang i ne peut donc pas retrouver les

résultats partiels d'un agent arrivant sur son site et qui a visité les sites de 1 à $i-1$. Il peut toutefois supprimer ces résultats. Au final, les PRAC permettent au propriétaire de l'agent de vérifier automatiquement chaque résultat partiel contenu dans l'agent.

Pour protéger la confidentialité des parties de codes et des données de l'agent, il existe des solutions partielles qui chiffrent le code et les données tout en les gardant exécutables.

1. le **brouillage** de code [Hoh98b] cherche à protéger l'agent contre une analyse de l'attaquant. L'idée est de complexifier le code et les données de l'agent afin qu'il soit très difficile d'inférer des informations utiles dessus. Ce nouvel agent « boîte noire » réalise le même service. La conversion conceptuelle utilise des techniques telles que :

   – le découpage et recomposition des données,
   – la modification du flot de contrôle en le rendant dépendant des variables,
   – l'introduction de code/données inutiles, etc.

   Le brouillage rend le code et les données de l'agent difficilement analysables (p.ex. analyse de flot ou *slicing*) au moins pendant un certain laps de temps. L'agent est signé avec une date limite d'utilisation. Passé ce délai, l'agent ne peut être exécuté. Cela prévient toute tentative d'analyse sophistiquée du code et tout rejeu. Les contraintes temporelles de l'approche limitent son utilisation. De plus, elle induit un coût en terme de temps d'exécution et de trafic.

2. Sander et Tschudin [ST98] ont proposé une idée théorique afin de préserver la confidentialité de parties de code d'un agent. L'idée est de **chiffrer** certaines fonctions de l'agent. Le site hôte exécute une fonction chiffrée d'un agent, sans être capable de discerner la fonctionnalité initiale. Les résultats ne peuvent être déchiffrés que sur le site client.

   Par exemple, supposons que Alice (le client) dispose d'un algorithme pour calculer une fonction sensible t. Bob (le site distant) dispose d'une donnée d et doit calculer $t(d)$ pour Alice. Alice souhaite que Bob ne puisse connaître la fonction t. Cette fonction peut être chiffrée par Alice afin de produire la fonction $C(t)$. Alice peut alors créer le programme $P(C(t))$, qui met en œuvre $C(t)$, et l'envoyer à Bob dans un agent. Bob exécute l'agent en appliquant $P(C(t))$ sur d, et retourne les résultats à Alice qui les déchiffre pour obtenir $t(d)$.

   L'approche reste théorique et le chiffrement ne fonctionne que sur un ensemble restreint de fonctions (p.ex. fonctions polynomiales et rationnelles). À ce jour, cette approche ne peut être appliquée à du code procédural général et est limitée au cas où seul l'initiateur de l'agent utilise le résultat du calcul chiffré, c.-à-d. ce résultat ne peut être réutilisé sur un site suivant de l'itinéraire. Comme toutes les autres, elle n'est pas générale envers la sécurité (p.ex. pas de protection pour l'intégrité, le déni de service ou la réexécution d'agents).

2.3 Fiabilité

Le domaine de la sûreté de fonctionnement est assez vaste et intervient dans le matériel, le logiciel ou les systèmes distribués. Nous rappelons ici partiellement le vocabulaire et

la problématique du domaine pour le lecteur non initié. Nous nous référons aux définitions fournies dans [Lap95, Cri91] et à la classification des propriétés de sûreté de fonctionnement définie dans [Sar99]. Toutefois, nous les avons parfois adaptées au contexte des architectures de services.

La sûreté de fonctionnement (*dependability)* d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre [Lap95]. Une application distribuée sûre de fonctionnement doit pouvoir continuer de fonctionner en présence de défaillances dans le système. Une défaillance est une déviation du comportement d'un composant envers sa spécification [Lap95]. Une défaillance se produit à la suite d'une erreur (état incorrect du système). La cause d'une erreur est une faute. Une faute peut être induite par un phénomène extérieur (p.ex. phénomène physique) ou une imperfection dans le système (p.ex. faute de conception).

Dans une architecture de services, les défaillances de sites peuvent conduire à un comportement défaillant d'un service et donc le rendre inutilisable au client (p.ex. le client ne récupère pas ses résultats ou ils sont incorrects). Plusieurs types de défaillances peuvent être considérées [CP97] :

- une défaillance par arrêt (p.ex. *crash*, panne) quand un serveur ne rend pas de résultats suite à des invocations répétées.
- une défaillance par omission quand un serveur omet de répondre à son client,
- une défaillance temporelle quand la réponse du serveur est fonctionnellement correcte mais n'est pas arrivée dans un intervalle de temps donné,
- une défaillance de valeur quand le serveur rend des résultats incorrects,

2.3.1 Propriétés de sûreté de fonctionnement

Le domaine de la sûreté de fonctionnement combine plusieurs sous-propriétés qui imposent l'utilisation de mécanismes spécifiques pour être garanties/optimisées. Nous nous appuyons ici sur la classification de Saridakis [Sar99] pour identifier ces différentes propriétés. Cette classification formelle repose sur une logique et a pour but de guider l'intégration de mécanismes voués à optimiser des propriétés de fiabilité sur un intergiciel client-serveur. Elle s'appuie sur trois sous-propriétés qui sont des raffinements de la propriété de sûreté de fonctionnement :

1. la fiabilité (*reliability*) qui correspond au fait qu'une unité fonctionnelle est conforme à ses spécifications, c.-à-d. qu'elle délivre correctement le service attendu.
2. la disponibilité (*availability*) qui correspond au fait qu'une unité fonctionnelle est accessible à un instant donné.
3. la sécurité-innocuité (*safety*) qui correspond au fait qu'une unité fonctionnelle n'entraîne pas de défaillances catastrophiques (p.ex. incendie).

Une adaptation de cette classification est donnée dans la Figure 2.3. Une boîte représente une propriété. Elle est grisée quand elle est associée à un mécanisme. Un arc entre deux boîtes correspond à une relation de raffinement. La disjonction de propriétés est représentée par les arcs issus d'une même boîte. Ainsi, au sens de Saridakis [Sar99], la propriété de sûreté de

fonctionnement se définit comme la disjonction des propriétés de sûreté, de fiabilité et de disponibilité.

Pour améliorer la sûreté de fonctionnement, il est nécessaire d'annuler les conséquences des défaillances, soit en interdisant/contrant les fautes (p.ex. prévention, vérifications, preuves) soit en tolérant les fautes. Dans le cadre des défaillances logicielles, des mécanismes de tolérance aux fautes sont utilisés pour optimiser la propriété de fiabilité. Ces mécanismes portent par exemple sur la détection des fautes et leurs masquages [Lap95, SI99]. Le masquage s'appuie le plus souvent sur la redondance de ressources (réplication de données ou de serveurs).

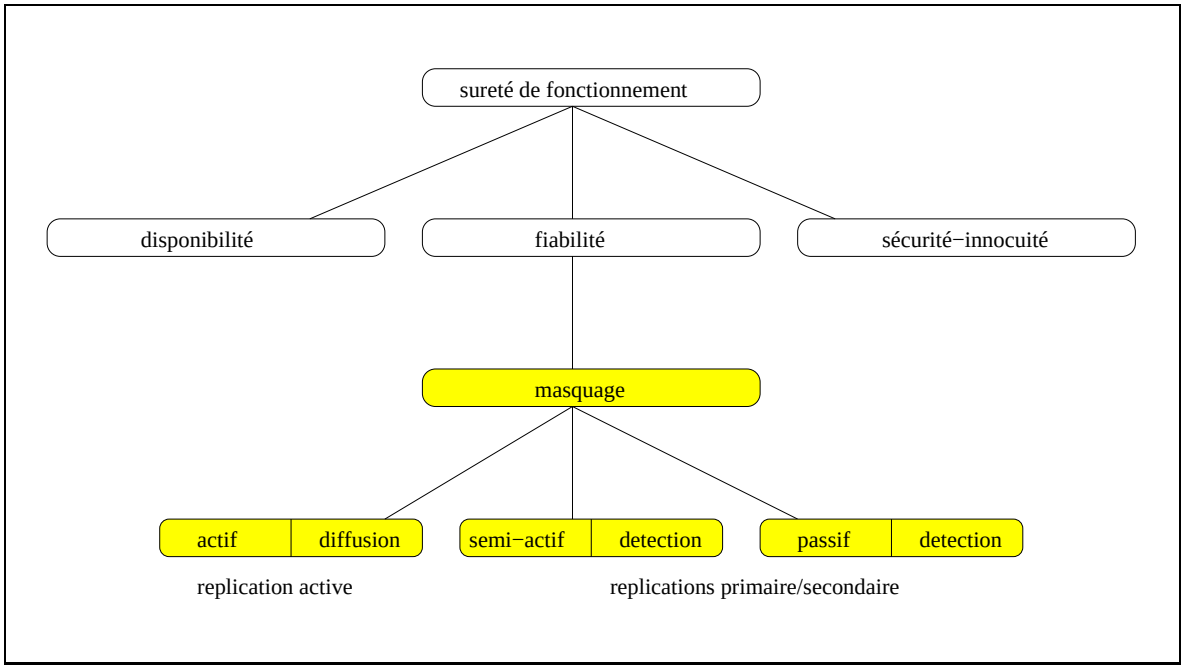

FIG. 2.3 – *Classification de propriétés de sûreté de fonctionnement*

Nous étudions ici les grandes familles de ces mécanismes proposés dans un cadre client-serveur. Ils sont également utilisés dans le cadre des agents mobiles pour favoriser des exécutions plus fiables d'agents. Nous ne nous intéressons pas en détail aux défaillances des communications qui pourraient induire une erreur lors de la transmission d'un message entre un client et un service. Pour pallier ces défaillances, des mécanismes largement utilisés par les systèmes de communication portent sur de la redondance d'informations sur les messages (p.ex. *Cyclic Redundancy Code*, codes de Hamming, code correcteur) afin qu'un message erroné soit détecté, voire évité.

2.3.2 Tolérance aux fautes

Dans le cadre des invocations distantes, divers mécanismes de tolérance aux fautes sont communément utilisés. Au sein des intergiciels, la tolérance aux fautes consiste à interposer des services systèmes entre l'application et le système d'exploitation (p.ex. le service OTS de CORBA, qui garantit l'atomicité des défaillances). En général, pour la propriété de fiabilité, les défaillances sont palliées grâce à des techniques de masquage qui utilisent de la réplication. Une première méthode, la réplication temporelle, consiste à exécuter chaque service plusieurs fois en utilisant les mêmes ressources. L'exécution est considérée comme correcte si

les différentes exécutions aboutissent et ont des résultats identiques. Une seconde méthode, la réplication spatiale, consiste à implanter le service comme un groupe de n services redondants, physiquement indépendants (c.-à-d. un même service sur plusieurs sites). Ainsi, si l'un des sites défaille, les autres peuvent encore assurer le service. Les techniques de réplication spatiales sont divisées en deux catégories [CP97] :

1. la réplication active, où chaque site du groupe exécute le service demandé. Le nombre de répliques composant le groupe doit être suffisant pour calculer en permanence un état correct du système. Ce mécanisme ne nécessite pas de détection d'erreurs. En effet, les valeurs de sortie des services répliqués sont comparées entre elles par un unificateur et ne sont propagées au destinataire que si elles sont identiques. Sinon, un arrêt est simulé.

2. la réplication primaire/secondaire, où un site primaire exécute le service tandis que les $n-1$ autres sites secondaires mémorisent une copie de l'état de la réplique primaire afin de s'y substituer en cas de défaillances. Dans la réplication semi-active, la réplique primaire transmet des synchronisations aux répliques secondaires pour qu'elles puissent s'exécuter de manière déterministe. Seule la réplique primaire génère les valeurs de sortie et règle les sources d'indéterminisme. Avec la réplication passive, la réplique primaire transmet régulièrement aux secondaires une copie de son état (point de reprise) ainsi que les informations nécessaires pour garder un groupe déterministe. En cas de défaillance, la nouvelle réplique primaire active cet état.

2.3.3 Tolérance aux fautes pour agents mobiles

Les agents mobiles nécessitent d'envisager de nouveaux mécanismes de tolérance aux fautes. Ces mécanismes sont majoritairement conçus sur les sites et sont invisibles aux agents et programmeurs d'agents. Un agent mobile effectue des actions (p.ex. interaction locale, exécution d'une partie de son code) sur les sites de son itinéraire. Son itinéraire est constitué de n étapes (c.-à-d. vecteur $E = (E_0, ..., E_n)$) qui correspondent aux n sites où sont appliquées les actions. Le site E_0 correspond au site source et le site E_n au site destination.

Dans le cas de défaillances de sites, pour assurer la continuité des actions réalisées par un agent, il est nécessaire de les répliquer sur plusieurs serveurs distincts formant un groupe. Divers mécanismes de tolérance aux fautes ont été proposés afin d'assurer des exécutions fiables d'agents mobiles.

2.3.3.1 Réplication active

Dans la réplication active, chaque étape de calcul d'agent est répliquée. Un calcul d'agent est alors exécuté simultanément sur n sites distincts d'une étape. Schneider [Sch97] a proposé d'utiliser une technique de réplication active avec vote afin de masquer les effets de sites défaillants. L'auteur considère le cas d'un agent dont l'itinéraire comporte n sites. Le site source est le site à partir duquel l'agent initie son exécution (c.-à-d. le site du client). Le site destination est celui où il termine son exécution (pas nécessairement le client). Afin d'optimiser la fiabilité, l'approche proposée suppose que les sites source et destination sont fiables, c.-à-d. qu'ils exécutent l'agent sans fautes. Le plus souvent, le site destination est identique au site source, c.-à-d. le site du client. La correction d'une étape de calcul sur un site i dépend de la correction des étapes précédentes de l'itinéraire de l'agent. Afin de rendre l'intégralité

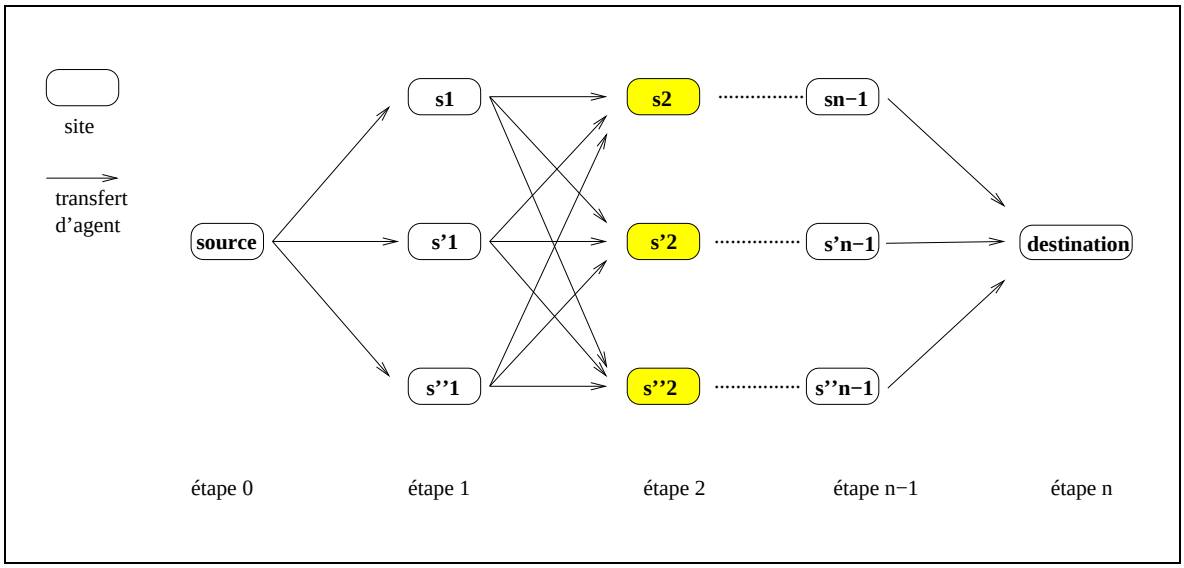

FIG. 2.4 – *Réplication active avec vote*

du calcul d'un agent tolérant aux défaillances, Schneider utilise à chaque étape (hormis à la source et à la destination) un mécanisme de réplication active (voir Figure 2.4) :

- chaque site d'une étape i dispose des agents ayant effectué l'étape $i-1$. Un vote permet de ne conserver qu'un seul des agents reçus pour l'exécuter sur le site hôte.
- après exécution de l'agent, la réplication sur chaque site d'une étape i permet de transmettre les agents à l'étape $i+1$.

Un seuil d'agents valides reçus étant suffisant pour le vote, ce mécanisme permet de ne pas avoir à attendre les agents les plus lents pour continuer les étapes de calcul. La technique proposée induit un coût en terme de performance dû à l'envoi et l'exécution simultanée d'agents, ainsi qu'aux votes.

2.3.3.2 Réplication primaire/secondaire

Dans la réplication primaire/secondaire, à chaque étape, un agent mobile est répliqué en n exemplaires. Toutefois, une seule des n répliques effectue le calcul. Les $n-1$ autres répliques sont passives et ne prennent la relève qu'en cas de défaillance de la réplique active. Straßer et Rothermel [SR98] choisissent d'utiliser un protocole qui garantit la propriété «exactement une fois» (*exactly-once*) des exécutions d'agents à chaque étape. Cette propriété, déjà définie dans le cadre RPC (c.-à-d. sémantique de défaillance d'une procédure distante), est étendue ici à une séquence d'étapes (itinéraire statique d'un agent). L'exécution d'un agent vérifie la propriété «exactement une fois» si et seulement si l'agent réalise son itinéraire dans l'ordre fixé et chacune des actions est exécutée exactement une fois par étape. La propriété est garantie à l'aide d'une série de transactions puis de votes. En reposant sur les files de messages transactionnels, une transaction sur un site est définie par :

- prendre l'agent dans la file d'entrée,

- exécuter intégralement le code une et une seule fois,
- placer l'agent dans la file de sortie,
- valider la transaction (c.-à-d. *commit*).

Pour réaliser une transaction, plusieurs sites ont en charge l'exécution à chaque étape. Le site primaire exécute l'agent (en grisé sur la Figure 2.5), alors que des sites secondaires observent par intervalles réguliers l'exécution (c.-à-d. le site primaire émet régulièrement des messages du type *IamAlive* aux sites secondaires). Si le site primaire défaille, un des sites secondaires reprend la main. Le choix du site reprenant l'exécution est déterminé par un vote (protocole de validation en deux ou trois phases [SPZ98]).

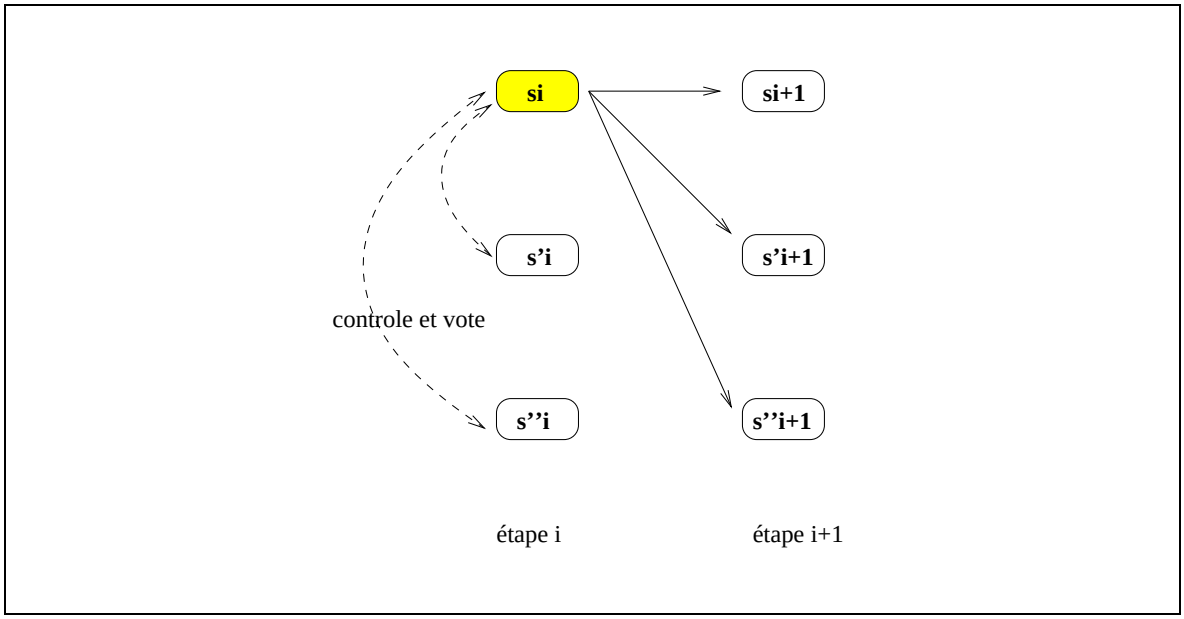

FIG. 2.5 – *Réplication semi-active avec vote*

2.4 Modèles formels de la mobilité

Plusieurs travaux portent sur la description de systèmes à agents mobiles à partir du développement des idées des calculs de processus, orientées vers la distribution [BGL00], ou à partir du λ-calcul. Ces calculs, portés sur la mobilité, ne s'intéressent pas au schéma client-serveur qui est fondé sur la seule migration de données. Les entités migrantes ont en général un contenu exécutable. Par exemple, des variantes du π-calcul de Milner [Mil98], étendues avec la notion de domaine, permettent de modéliser la mobilité des calculs et environnements d'exécution et d'en formaliser la sémantique. Un domaine regroupe des entités, qui sont soit d'autres domaines, soit des ressources ou des processus (p.ex. agents mobiles). Par la suite, nous nous focalisons sur quatre modèles. Ils portent sur la mobilité de domaines pour les trois premiers, notion plus englobante que la seule mobilité d'agents. Le dernier ne traite que de la mobilité de code :

1. le **Join-calcul distribué** [FGL+96], dérivé du π-calcul, est un modèle de programmation

mobile qui autorise une implémentation distribuée. Ce modèle est doté d'une sémantique chimique (RCHAM *Reflexive Chemical Abstract Machine*). Il définit deux entités : (i) les domaines (*localité-définition*) et (ii) les processus (*local solution*). Un domaine représente par exemple une région, une place, un agent ou une ressource. Le Join-calcul manipule des noms de canaux et des noms de domaines et se porte vers l'étude des défaillances partielles. À la π-calcul, les processus peuvent émettre des noms de canaux ou des processus [San93] à d'autres processus.

2. le calcul des **Ambients** [CG98] est utilisé pour modéliser les domaines de sécurité. Les Ambients sont des domaines qui contiennent des processus, ceux-ci étant des compositions parallèles d'actions. Le calcul manipule des noms, régis par une règle de portée lexicale.

3. le **Seal-calcul** [VC98] est un calcul de la distribution « grande échelle ». Il est utilisé pour exprimer les propriétés de sécurité de programmes Internet, notamment en considérant les mécanismes de protection. Il définit trois entités, (i) les domaines, (ii) les ressources et (iii) les processus. Un domaine est un Seal, qui est une localité contenant des processus s'exécutant et d'autres Seal. Les ressources considérées sont les canaux de communication qui sont utilisés pour synchroniser des processus concurrents (p.ex. agents) et pour communiquer des noms. Les entités nommées sont les canaux et les domaines. Les processus sont représentés à l'aide d'instructions de migration et de communication. Comme le Join-calcul, ce calcul est fondé sur le π-calcul et se focalise sur la mobilité de code et les contrôle d'accès aux ressources.

4. le calcul **λdist** [SY97], à la différence des extensions précédentes sur le π-calcul, est une extension du λ-calcul. Il a été utilisé pour comparer et discuter les différences entre les mécanismes de migration d'agents (comprenant les migration de code, données et états d'exécution). Un agent peut être représenté dans le calcul λdist qui contient, en plus des constructions de base du λ-calcul « appel par valeur », une primitive de migration. Ce calcul permet de représenter la mobilité de code et illustrer de mécanismes de mouvement des données d'un état d'exécution. Ces mécanismes de mouvement des données des états d'exécution varient suivant les langages pour systèmes à agents mobiles. Par exemple la migration peut être faible ou forte, suivre un schéma d'évaluation distante ou encore le contexte d'exécution peut rester sur le site source (référence distante). Les auteurs ont fournit une proposition d'encodage en λdist des mécanismes de migration des constructions des langages Telescript, Obliq, Facile et Java.

Dans les extensions du π-calcul pour la mobilité (c.-à-d. Join, Ambient et Seal), la notion de domaine est centrale. Ces calculs considèrent qu'un domaine abrite des processus. Les domaines et agents sont ainsi représentés par des processus qui communiquent/migrent à travers des canaux. Un domaine peut être désigné, en tant que cible pour une communication ou pour une migration. L'espace des domaines introduits par ces calculs est structuré hiérarchiquement sous la forme d'un arbre. Dans une arborescence, un noeud représente un domaine, abritant ses propres processus et possédant des domaines fils. Cette structuration des domaines permet d'expliciter la migration des domaines les uns par rapport aux autres, à partir d'une reconfiguration locale de l'arborescence. La structure d'imbrication des domaines évolue dynamiquement. À la différence des Join, Ambient et Seal calculs, où l'entité migrante est un domaine ou un processus, il n'existe pas de notions de domaines dans le calcul λdist, la structure est plate, non imbriquée et non hiérarchique (un domaine est restreint à un site).

La sémantique de domaine prend diverses orientations selon la vocation première des modèles étudiés ici :

- communication pour les Join, Ambient et Seal calculs,
- migration pour les Join, Ambient, Seal et λdist calculs,
- sécurité pour l'Ambient et le Seal calculs,
- défaillance pour le Join-calcul.

Ces sémantiques sont présentées dans les sections suivantes.

2.4.1 Sémantique de communication

Dans les modèles à base de π-calcul avec domaines, les entités d'un même domaine sont unifiées par un mode de communication qui peut-être :

1. local quand une communication est intra-domaine,
2. distant quand une communication est extra-domaine.

La désignation de l'entité réceptrice de la communication est directe, c.-à-d. basée sur le nom primitif du domaine, de la ressource ou du processus. Un processus qui reçoit un nom de domaine acquiert le droit de désigner ce domaine à des fins de communication ou de migration.

- le Join-calcul autorise, sur un canal, à la fois les communications locales et les communications distantes. Les arguments de la communication sont des noms ou des séquences d'arguments. Ce calcul peut donc traiter les communications de type invocations locales et invocations distantes.
- le calcul des Ambients ne traite que les communications locales asynchrones. Ainsi, si l'on s'intéresse aux agents mobiles, ils n'ont pas la possibilité d'interagir à distance avec un site, un service applicatif ou d'autres agents. Un agent doit nécessairement migrer vers le domaine de son interlocuteur pour pouvoir engager une communication.
- le Seal-calcul permet la communication locale synchrone entre deux processus d'un même domaine sur un canal local. En plus des communications locales, il permet de modéliser les communications distantes. Toutefois, celles-ci sont restreintes à des communications entre domaine père-fils. Ainsi, pour une communication extra-domaine, les domaines en jeu doivent être directement lié dans la structure hiérarchique. De plus, pour que la communication puisse se réaliser, l'autorisation doit en être explicitement donnée (utilisation de capacités). Dynamiquement, la communication est contrôlée explicitement par un *firewall* qui ouvre l'accès au canal utilisé.

2.4.2 Sémantique de mobilité

Le Join, les Ambients et le Seal permettent la modélisation des migrations d'agents mais sont plus généraux. En effet, grâce à la notion de domaine, la modélisation des migrations se porte également aux domaines (p.ex. environnement d'exécution). Grâce à l'espace des

domaines structuré hiérarchiquement, la migration d'un domaine entraîne la migration de ses domaines fils.

- dans le Join-calcul, seuls les domaines peuvent migrer. Il n'y a pas possibilité pour un processus s'exécutant dans un domaine de quitter celui-ci indépendamment. La migration est représentée dans le calcul par une instruction de migration qui a pour effet de déplacer le domaine englobant (les domaines fils de la hiérarchie et les processus qu'il contient) pour en faire un sous-domaine de la destination de la primitive de migration.

- pour le calcul des Ambients, un domaine peut migrer de manière atomique. Les processus abrités par un domaine en assurent le contrôle (p.ex. décision de migration). Une instruction de migration désigne directement, par son nom, la destination de la migration. Il y a deux instructions restrictives de migration dans ce calcul. La première permet à un domaine de sortir du domaine qui le contient (devient un frère de son père). La seconde permet d'entrer chez un voisin (devient le fils d"un frère). Lors d'une migration, les Ambients fils de la hiérarchie et les processus qu'il contient sont emportés.

- en Seal-calcul, un domaine est également une entité qui peut migrer. La migration s'exprime comme un envoi de message. En envoyant le nom d'un domaine sur un canal, ce domaine est déplacé vers le domaine destination. Comme pour la communication, la migration est contrôlée explicitement par un *firewall* qui ouvre l'accès au canal.

- dans le calcul λdist, plusieurs sémantiques de mobilité peuvent être représentées. En effet, le but de ce calcul est de permettre la comparaison de mécanismes de mouvement de code dans des langages utilisés pour mettre en œuvre la mobilité d'agents (p.ex. Telescript, Java). La sémantique du calcul est opérationnelle et est définie à l'aide de contextes. Sur un site, un contexte est représenté par une « table locale» contenant les données des états d'exécution. La table locale dispose des opérations standards (p.ex. allocation, ajout, suppression, mise à jour). Une expression en λdist est une λ-expression comprenant une expression du code de l'agent et des opérations sur les tables. Les auteurs ont décrit plusieurs types de mouvement des données des états d'exécution (p.ex. migration faible et forte, évaluation distante) en étudiant comment les tables d'un site source et d'un site destination changent avant et après une migration. Quand une valeur est déplacée vers un autre site, elle peut être copiée ou devenir une référence distante. La sémantique opérationnelle est définie comme suit : selon un site source et une table de ce site, la sémantique associe à une expression e une expression e' (après une étape de réduction de e) et la nouvelle table. Si la réduction porte sur une migration, la table destination et la table source de la migration sont mises à jour.

2.4.3 Sémantique de sécurité

Pour la sécurité, l'Ambient et le Seal calculs ont des sémantiques qui rendent explicites les concepts de frontière et d'autorisation d'entrée dans un domaine, ou encore l'autorisation d'accès à une ressource hébergée par un domaine (Seal). Le concept de frontière et d'autorisation d'entrée dans un domaine est explicite. La restriction sur l'utilisation des noms de canaux (portée lexicale des noms) permet de contrôler, en terme de sécurité, les communications (p.ex. accès aux ressources) et la migration d'un domaine. Les noms connus d'un processus

sont restreints. Un nom ne peut sortir de la frontière définie par sa portée.

- pour le calcul des Ambients, le nom d'un domaine constitue un « mot de passe» qui doit être connu afin de quitter un domaine et entrer dans un autre. Les droits d'accès à un domaine sont capturés par la portée des noms de domaines. Dans ce calcul, le contrôle dynamique de sécurité (en cas de violation) utilise une dissolution de la frontière d'un domaine, ce qui a pour effet de lever les droits d'accès aux processus du domaine.

- le but du Seal-calcul est de permettre de garantir des propriétés de sécurité. En Seal-calcul, un Seal est un domaine de sécurité. Le calcul permet d'examiner les endroits où peuvent se porter des attaques pour en déduire les mécanismes de protection adéquats. Les canaux de communication peuvent être sécurisés avec des protocoles dédiés. Un agent doit être identifié (nommage) et des droits d'accès lui sont accordés. Le calcul traite la protection des ressources d'une structure d'accueil contre des agents malveillants ainsi que celle des agents contre les structures d'accueil. Le calcul prend notamment en compte les attaques suivantes :

  - divulgation et modification non autorisée d'informations, par la prise en compte de la portée lexicale de noms et l'utilisation de capacités linéaires pour protéger les ressources (de site ou d'agents). Un processus ne peut utiliser que les canaux dont il connaît les noms. L'utilisation des ressources peut être contrôlée, ainsi que les migrations et les communications (modèle de protection hiérarchique père-fils).
  - introduction de chevaux de Troie dans un système.

  Toutefois, dans ce calcul, un site peut falsifier les informations transmises à un domaine, suivre l'itinéraire d'un domaine, usurper son identité ou espionner les communications [BGL00].

2.4.4 Sémantique de défaillance

Dans le Join-calcul, la sémantique de domaines facilite l'explicitation de protocoles de détection et de contrôle en terme de défaillances. À un domaine est associé un état qui détermine s'il est défaillant ou non. Les défaillances traitées dans ce calcul sont permanentes et ne portent que sur les domaines. L'état d'un domaine peut être observé et donc sa défaillance peut être détectée depuis n'importe quel autre domaine non défaillant. Quand un domaine devient défaillant, il entraîne la défaillance de tous les processus qu'il abrite et de tous les domaines fils. Cette approche permet de représenter un groupe de processus liés par un même comportement défaillant (p.ex. blocage de l'émission/réception de messages ou de la mobilité). Un domaine défaillant ne peut abriter aucune réaction, émettre/recevoir aucun message ni aucun domaine migrant. Dans le calcul, cette sémantique de défaillance facilite l'explication de protocoles de détection et de contrôle des défaillances.

2.4.5 Commentaires

Les Join, Ambient et Seal calculs ont pour vocation de permettre l'observation et/ou le contrôle des domaines depuis l'application. Ces calculs sont très généraux et puissants et sont motivés par l'hypothèse d'un environnement où les agents et leurs actions changent constamment. Les Join, Ambient et Seal calculs se placent dans le domaine du contrôle dynamique de propriétés non fonctionnelles d'un domaine ou de son état, plutôt que dans l'analyse statique

de celles-ci. Ces calculs ayant un pouvoir d'expression puissant, il est relativement délicat de proposer des analyses statiques globales sur des spécifications de système. En pratique, il est donc difficile de les utiliser pour analyser statiquement des propriétés globales de qualité de service (p.ex. performance, sécurité, sûreté de fonctionnement).

Le calcul λdist, voué à comparer les mécanismes de migration de code de langages existants, ne propose pas d'analyses non fonctionnelles. Il se focalise sur les sémantiques de migration.

2.5 Discussion

Les agents mobiles ont suscité un intérêt considérable dans la communauté de recherche système et dans l'industrie. En effet, cette technologie est prometteuse et pose de nouveaux problèmes de recherche. Le domaine est encore jeune et il reste de nombreux problèmes techniques à traiter. Les recherches engagées pour les pallier sont encore récentes. Nous avons présenté une partie de l'état de l'art des travaux qui portent sur les agents mobiles pour évaluer et/ou assurer les propriétés de performance, de sécurité et de fiabilité. Ces travaux, pour certains, sont conceptuels et n'apportent pas encore de solutions complètement satisfaisantes. Les avantages de la mobilité des agents sont tactiques, par exemple pour améliorer l'efficacité en privilégiant les interactions locales afin de réduire la taille ou le nombre de messages transmis sur le réseau. Ils sont stratégiques quand les agents permettent d'étendre les fonctionnalités ou services offerts par les serveurs (c.-à-d. flexibilité). Ils permettent à un client d'adapter/personnaliser un service à ses besoins spécifiques. De plus, l'attribut de mobilité peut faciliter la conception d'applications ayant à faire à des utilisateurs mobiles. L'agent, une fois déposé sur le réseau, peut accomplir les actions requises par l'utilisateur sans souffrir des déconnexions potentielles.

La diffusion des agents mobile dans les réseaux ouverts est limitée par la crainte d'importer du code malveillant. La confidentialité et l'intégrité des objets des sites envers des agents malicieux commencent toutefois à être relativement bien traités grâce aux environnements interprétés (p.ex. Java). Ils permettent de restreindre l'accès aux ressources du site hôte, à la manière de ce qui existe déjà sur le Web avec les appliquettes (*applets*) Java. Le déni de service est difficile à prévenir comme c'est le cas dans les systèmes plus classiques. Il est envisageable de maîtriser ce point directement sur les environnements d'accueil des agents en limitant les calculs, leur type, le nombre des agents, le clonage ou l'accès aux ressources [SIB98]. Ces points sont de nature technique, une fois les agents déployés, et ne sont pas un réel frein à l'utilisation des agents.

La protection des agents reste un problème à part entière. Un agent peut difficilement transporter des données critiques entre des sites sans niveau de confiance. Le problème du site malfaisant est loin d'être totalement traité, particulièrement dans les systèmes ouverts. Du point de vue client, la sécurité est fondamentale dans certaines applications (p.ex. commerce électronique).

Les agents mobiles sont particulièrement adaptés pour les applications « haut niveau », mais restent lourds techniquement pour les applications à grain fin comme c'est le cas dans les réseaux actifs. Dans bien des cas, leur utilisation potentielle est liée aux réseaux à grande échelle qui tendent à devenir les futures plates-formes distribuées. Ces réseaux offrent de nombreuses ressources exploitables par les agents. Sur l'Internet, c'est par exemple le cas avec

les serveurs locatifs d'application (ASP *Application Service Provider*, p.ex. Tomcat Server d'Apache, WebSphere d'IBM, Weblogic de BEA, IIS de Microsoft, OrbixE2A de IONA) qui proposent des applications légères et génériques. Les entreprises peuvent déposer et ainsi externaliser leurs applications sur de tels serveurs puissants. Ce type de réseau est un terrain particulièrement favorable aux agents mobiles. Toutefois, les services sur ces réseaux interagissent le plus souvent par invocations distantes. Ceci tient en partie au fait qu'il existe très peu de systèmes distribués grande échelle conçus à partir de la technologie agents mobiles. En effet, pour un déploiement effectif, l'utilisation des agents nécessite des environnements d'exécution sur toutes les plates-formes visitables.

Certains estiment que les agents mobiles sont une solution encore à la recherche d'un problème. Il manque une application impressionnante (*killer application*) afin de justifier la technologie. Notons que tout ce qui est réalisable par des agents mobiles l'est aussi par des interactions traditionnelles à base d'invocation distante. De notre point de vue, dans les architectures de services, les agents mobiles se justifient uniquement par les propriétés non fonctionnelles. Beaucoup de conclusions sur la technologie des agents mobiles sont de nature qualitatives et subjectives au regard des propriétés non fonctionnelles. À ce jour, il existe peu de travaux théoriques ou pratiques pour justifier de l'intérêt des agents mobiles envers ces propriétés et au regard des interactions plus classiques. Ce constat a motivé et guidé ce travail de thèse.

Chapitre 3

Un cadre formel pour la spécification de services distribués complexes

3.1 Introduction

Nous avons vu dans la première partie de cette thèse que les architectures de services (p.ex. services Web) connaissent un essor important. Afin de formaliser la composition de services primitifs, nous proposons dans ce chapitre un cadre fonctionnel pour la spécification abstraite de services distribués complexes. Il se base sur un langage abstrait qui permet d'exprimer simplement une large classe de services complexes. Le plus souvent, les interactions mises en œuvre sur les architectures de services reposent sur de l'invocation distante (cf. section 1.2). Nous sommes toutefois convaincus de l'intérêt du paradigme code mobile (p.ex. technologies code à la demande, exécution à distance et agents mobiles) au sein de ces architectures. Pour cela, le cadre proposé considère un deuxième langage pour la description des choix d'interactions sur lequel peuvent s'associer des analyses simples. Ce deuxième langage, plus concret, peut être vu comme un modèle d'exécution qui supporte les mécanismes d'interactions à base de code mobile (hormis le code à la demande). Il permet d'expliciter le type des interactions (c.-à-d. exécution locale, invocation distante, évaluation distante et migration d'agents). Ainsi, le cadre proposé permet d'exprimer les schémas d'interaction à base d'invocation distante, d'exécution à distance et d'agents mobiles, pour un service complexe décrit par une expression appartenant au langage abstrait. Il est également possible de générer automatiquement les expressions concrètes qui mettent en œuvre une expression abstraite donnée.

Notre cadre fonctionnel a été choisi afin de permettre un traitement simple et uniforme d'analyses de propriétés non fonctionnelles dans les architectures de services. En plus, il permet donc d'étudier différents schémas d'interaction et de choisir les plus adaptés au regard de critères de qualité. Il est possible de raffiner automatiquement une expression abstraite en une concrète en choisissant le nombre, les sites d'exécution des traitements et l'itinéraire des agents. À partir d'une expression abstraite, il est donc possible de générer des expressions concrètes fonctionnellement équivalentes et de les comparer simplement suivant des propriétés non fonctionnelles (c.-à-d. performance, sécurité, fiabilité). Nous recherchons à mettre en œuvre (choix des interactions) un service complexe « optimal » à la charge d'un client.

Aux vus du domaine d'application et des propriétés étudiées (viabilité des analyses), le cadre fonctionnel de spécification est minimal bien qu'il permette de décrire et analyser de manière automatique une majorité de services distribués complexes. Notre objectif est de

justifier de l'intérêt des agents mobiles dans les architectures de services, organisées, le plus souvent, sous un schéma client-serveur. Dans le cadre de cette thèse, nous nous restreignons donc au concept d'agents mobiles vus comme une variation du schéma client-serveur. Un agent, initié par un client, peut migrer entre différents sites pour interagir avec des services statiques (éventuellement mis en œuvre par des agents) et effectuer des traitements personnalisés (définis dans le code de l'agent) sur les sites visités. Les hypothèses de travail de ce cadre fonctionnel sont :

1. **itinéraire statique :** dans les mécanismes d'interactions par agents mobiles choisis, l'itinéraire est défini statiquement. En effet, il est difficile d'étudier, de manière unifiée, des propriétés non fonctionnelles telles que la sécurité, la performance ou encore la sûreté de fonctionnement sans avoir connaissance des sites à visiter. Dans la communauté de recherche sur les agents mobiles, issue initialement de la communauté système, un agent mobile est le plus souvent vu comme un processus autonome en terme de mobilité et non pas comme une entité intelligente qui prend des initiatives. Un itinéraire est le plus souvent défini statiquement dans le code de l'agent. Toutefois, la majorité des plates-formes à agents mobiles offrent des mécanismes de communication entre les agents. Les agents mobiles peuvent ainsi coopérer (sociabilité), à distance ou en local, par envoi de messages. Il devient alors possible de construire des systèmes multi-agents dynamiques, où les agents peuvent pro-activement [1] modifier leurs itinéraires (ce qui nécessite le plus souvent des mécanismes pour les retrouver).

   Dans le domaine d'application considéré (cf. architectures de services), au stade de la technologie actuelle, il n'est pas souhaitable d'utiliser des agents complètement autonomes en terme d'itinéraire. Par exemple, dans beaucoup de services de commerce électronique, ne pas maîtriser les migrations de ses agents peut aboutir à des problèmes de disponibilité ou de responsabilité du propriétaire. Un agent doué de la capacité de modifier pro-activement son itinéraire ou ses actions est susceptible de ne pas retourner au client dans un temps raisonnable ou encore de réaliser des services qui ne sont pas connus (ou requis) du client.

2. **invocations distantes sur site client :** les invocations distantes se font uniquement à partir du site représentant le(s) client(s). Pour une interaction par agent, les invocations de services sont faites en local sur les sites de l'itinéraire. Si les sites d'un réseau peuvent être utilisés comme centre d'invocations distantes par des agents mobiles, ces sites deviennent en quelque sorte des caches clients. Nous considérons que dans les architectures « grande échelle» de services, les prestataires extérieurs mettent leurs ressources (accueil des agents pour réaliser un calcul) à disposition des agents clients, dans la mesure où ces agents utilisent un service primitif *in-situ* et lui apporte éventuellement une valeur ajoutée à l'aide d'un traitement lié à ce service. À moins qu'il ne soit spécifiquement défini comme tel, un site de prestataire n'a pas à être encombré par des agents qui l'utilisent comme un centre d'appel. Les ressources d'un prestataire n'ont pas à être accaparées pour des activités qui ne sont pas liées au service proposé en local.

3. **pas de communication à distance entre agents :** le cadre fonctionnel ne considère pas de primitives de communication entre les agents du service complexe, hormis sur le site du client pour transmettre des résultats. Ces résultats sont retransmis au client ou

1. Un agent est pro-actif quand il peut agir (prendre l'initiative) en fonction d'un but à accomplir.

bien utilisés comme données arguments pour d'autres interactions, initiées à partir de ce même client. Les agents mobiles utilisés pour la mise en œuvre d'un service complexe ne communiquent donc pas entre eux sur des sites distants.

4. **mono-entrée :** la spécification d'un service complexe est mono-entrée, c.-à-d. qu'un service complexe est défini pour un même ensemble de données d'entrée. Si des données différentes sont introduites comme entrée d'un service complexe déjà construit, il est nécessaire de ré-étudier ce service complexe avec ces nouvelles données pour s'assurer de la garantie des critères non fonctionnels.

Dans ce chapitre, après avoir introduit notre approche à la composition de services, nous présentons le corps du langage abstrait pour la spécification haut niveau de services distribués dans la section 3.2. La sémantique des constructions de base est donnée et illustrée à l'aide d'exemples. Nous présentons de la même manière le langage concret qui permet la description de mises en œuvre de services distribués en section 3.3. Dans la section 3.4, nous nous intéressons de manière théorique à l'adéquation entre les deux langages en étudiant le moyen de réécrire une expression abstraite en une ou des expressions concrètes. Finalement, dans la section 3.5, nous discutons des autres formalismes qui auraient pu être utilisés et justifions les limitations de l'approche (hypothèses de travail) en terme de pouvoir d'expression.

3.1.1 Composition fonctionnelle de services primitifs

Dans un réseau de services distribués, des prestataires mettent à disposition de leurs clients potentiels des services de base, appelés primitifs par la suite. Ces services sont des applications, décrites et publiées par les prestataires (p.ex. dans un annuaire), puis localisées et invoquées à travers un réseau. Un service primitif est généralement invoqué sous un schéma client-serveur (cf. section 1.1). La requête d'un client, associée à des données d'entrée, est traitée par le service primitif pour rendre un résultat qui peut servir de données d'entrée pour d'autres services. Ainsi, un service complexe est issu de la composition de services primitifs. De plus, les services sont personnalisables à l'aide de traitements. Un service complexe peut être construit par un client ou être proposé par un prestataire que les clients verront comme un nouveau service primitif.

Nous proposons de spécifier les services primitifs comme des fonctions (primitives) et les services complexes comme des composition de ces fonctions. Un service primitif est alors représenté par une fonction, définie par son interface. Une telle formalisation permet de spécifier abstraitement l'interprétation sémantique d'un service complexe. Un service complexe est représenté sous la forme d'une expression fonctionnelle qui appartient à un de nos deux langages, engendrés par les grammaires définies par la suite.

3.1.2 Objets

Afin de permettre une spécification simple de services complexes, nous choisissons de considérer trois types d'objets abstraits, représentés par un identificateur dans les expressions :

- les données, attendues par un service primitif ou complexe, qui appartiennent au client (p.ex. requête, document). Elles peuvent être vues comme des fonctions d'arité nulle.

- les services primitifs, qui sont les fonctions offertes par les serveurs. Chacune porte un nom unique et offre un service primitif donné. Un service est décrit, au travers

de son interface fonctionnelle, par un prestataire de services (p.ex. annuaires UDDI, cf. section 1.2.3). Le résultat d'un service primitif dépend généralement des données passées en arguments de la fonction. Nous donnerons, dans la section 3.2.3 page 50, plusieurs exemples de tels services primitifs.

- les traitements, qui sont les fonctions primitives définies par les clients. Ces fonctions permettent d'étendre, par composition, les fonctionnalités primitives (personnalisation). Les traitements sont décrits dans des bibliothèques prédéfinies ou directement proposés par les clients (cf. exemples section 3.2.3).

Ces objets sont définis à l'aide du type *Data*, qui peut être primitif ou construit (construction n-uplet) :

$$Data ::= Int \mid Bool \mid ... \mid Data_1 \times ... \times Data_n$$

Ce type représente un univers de valeurs abstraites, telles que les nombres, les booléens, les tuples et autres structures.

La notation utilisée dans le document pour représenter l'interprétation sémantique d'une fonction s'appuie sur le λ-calcul [Han94]. Ce formalisme permet d'exprimer la notion très générale de fonction (construction et évaluation), sans préciser de domaine de départ ou d'arrivée. L'ensemble T des termes du λ-calcul est défini comme suit :

soit V un ensemble infini de variables,
$x \in V \ \Rightarrow \ x \in T$
$M \in T,\ x \in V \ \Rightarrow \ \lambda x.M \in T$ (λ-abstraction)
$M \in T,\ N \in T \ \Rightarrow \ MN \in T$ (application)

La correspondance entre les objets syntaxiques (c.-à-d. identificateurs) de nos langages et les valeurs abstraites se fait au moyen d'un environnement (e), qui associe à un identificateur d'objet (c.-à-d. donnée, service ou traitement) son interprétation sémantique, par exemple, sur le domaine des entiers :

$$\begin{aligned} e \ &: \ Id \ \rightarrow \ (Data \ \cup \ (Data \ \rightarrow \ Data) \ \cup \ \{erreur\}) \\ e \ &= \ \lambda x.\ si\ x = d_1\ alors\ 3 \\ &\qquad sinon\ si\ x = d_2\ alors\ 2 \\ &\qquad sinon\ si\ x = add\ alors\ \lambda(x,y).x + y \\ &\qquad ... \\ &\qquad sinon\ si\ x = sqr\ alors\ \lambda x.x * x \\ &\qquad sinon\ erreur \end{aligned}$$

L'environnement (e), appliqué à un identificateur d'objet id_i (noté $e\ id_i$ en λ-calcul), fournit soit une valeur constante de type *Data*, soit une fonction de type $Data \ \rightarrow \ Data$, soit *erreur* si l'identificateur n'est pas présent. Par exemple, l'application de e à l'identificateur de fonction (service ou traitement) *add* correspond à :

$$e\ add \ = \ \lambda(x,y).x + y$$

où $\lambda(x,y).x + y$ est une fonction à un argument (doublet) qui rend en résultat la somme des deux éléments du doublet. L'application de cette fonction à un argument doublet prend la forme de la réduction suivante :

$$(\lambda(x,y).x + y)\ (3,2) \ = \ 3 + 2 \ = 5$$

3.1.3 Intérêts de deux niveaux de spécification

Un des objectifs de cette thèse est de permettre la spécification de services distribués complexes et de guider la sélection de mécanismes d'interaction à l'aide d'analyses simples. Pour cela, nous proposons d'exprimer les services complexes à deux niveaux d'abstraction :

1. un premier niveau pour spécifier simplement la sémantique d'un service complexe, sans faire état de mécanismes d'interaction particuliers entre le client et les services. Il permet la spécification abstraite des dépendances entre les services/traitements utilisés, sous la forme d'une composition de fonctions.

2. un second, plus précis, pour la description de mises en œuvre. Il permet de spécifier de manière non ambiguë des interactions de type invocation distante et des interactions à base de code mobile, plus particulièrement d'agents mobiles. Le langage associé à ce niveau, appelé concret, rend les choix de mises en œuvre explicites. L'avantage des expressions concrètes est qu'elles peuvent être facilement analysées et comparées [FIR00] en terme de propriétés non fonctionnelles (p.ex. propriétés de qualité du type performance, sécurité ou sûreté de fonctionnement, cf. chapitre 2).

3.2 Langage abstrait

Le langage abstrait, destiné à la spécification de services distribués complexes, permet d'exprimer très simplement la composition de services et de traitements. Nous présentons ici le cœur de ce langage. Certaines constructions plus évoluées seront discutées dans la section 4.4, page 94, du chapitre suivant.

3.2.1 Syntaxe abstraite

Le langage abstrait est présenté dans la Figure 3.1. La spécification d'un service complexe est une expression qui est soit :

- un n-uplet d'expressions,

- une fonction primitive f appliquée à une expression. Les identificateurs de fonctions (ensemble *Primitif*) correspondent à des fonctions unaires et dénotent soit des services primitifs (ensemble *Service*), soit des traitements client (ensemble *Traitement*),

- une donnée d fournie en entrée, où les identificateurs de données (ensemble Id_d) correspondent à des valeurs de type *Data*.

$$E ::= (E, ..., E) \mid fE \mid d$$

$$\text{où} \quad f \in \mathit{Primitif} = \mathit{Service} \cup \mathit{Traitement} \text{ et } d \in Id_d$$

FIG. 3.1 – *Langage abstrait*

3.2.2 Sémantique

La définition d'une sémantique pour nos langages est essentielle dans la mesure où nous souhaitons établir dans un cadre formel des analyses de qualité. Nous utilisons une sémantique dénotationnelle [Sch86] définie sur un univers de valeurs abstraites (le type *Data*). À l'aide d'une telle sémantique, une expression abstraite Exp_a est traduite en une valeur abstraite de type *Data*.

L'interprétation sémantique des expressions abstraites (voir Figure 3.2) est relative à l'environnement (e) qui associe une sémantique aux identificateurs de *Primitif* et leurs valeurs aux identificateurs de données. L'interprétation sémantique d'une expression abstraite correspond simplement à l'évaluation fonctionnelle. La correspondance entre les expressions syntaxiques du langage abstrait et les valeurs asbtraites se fait au moyen de la fonction de valuation sémantique $\mathcal{E}_a$, de type $Exp_a \rightarrow Env \rightarrow Data$. La traduction d'une expression, représentative d'un service complexe, se fait par induction structurelle, c.-à-d. la valeur d'une expression est déterminée à partir des valeurs dénotées par les sous-structures de l'expression.

$$
\begin{array}{lcl}
\mathcal{E}_a \ : \ Exp_a \ \rightarrow \ Env \ \rightarrow \ Data & & \\
\mathcal{E}_a[\![(E_1, \ldots, E_n)]\!] \ e & = & (\mathcal{E}_a[\![E_1]\!] \ e, \ldots, \mathcal{E}_a[\![E_n]\!] \ e) \\
\mathcal{E}_a[\![f \ E]\!] \ e & = & e \ f \ (\mathcal{E}_a[\![E]\!] \ e) \\
\mathcal{E}_a[\![d]\!] \ e & = & e \ d
\end{array}
$$

FIG. 3.2 – *Sémantique du langage abstrait*

Par exemple, à une expression abstraite Exp_a du type $add \ (sqr \ d_1, d_2)$ va correspondre un entier au niveau de l'interprétation sémantique :

$$
\begin{array}{rcl}
\mathcal{E}_a[\![add \ (sqr \ d_1, d_2)]\!] \ e = & \mathcal{E}_a[\![sqr \ d_1]\!] \ e \ + \ \mathcal{E}_a[\![d_2]\!] \ e & \\
= & \mathcal{E}_a[\![d_1]\!] \ e \ * \ \mathcal{E}_a[\![d_1]\!] \ e \ + \ 2 & \\
= & 3 * 3 \ + \ 2 \ = \ 11 &
\end{array}
$$

3.2.3 Exemples

Avant de décrire des spécifications de services distribués complexes, nous donnons tout d'abord quelques exemples de services primitifs. Dans le domaine des services Web (cf. section 1.2.3), le site Internet *Xmethods* [Xme02] liste de plus en plus [2] de services qu'il est possible d'invoquer sous un schéma client-serveur. Cette liste contient (ou va contenir), entre autres, des services de :

- conversion de format (p.ex. documents, monnaie), calculatrice, traduction (p.ex. BabelFish d'AltaVista), annuaires téléphoniques inversés,
- consultation de stocks, de la valeur d'une enchère d'un produit (eBay), d'une valeur boursière ou du prix d'un ouvrage à partir de son identifiant ISBN (p.ex. Barnes & Noble Price Quote),

2. À ce jour, près de 200.

- consultation/réservation (p.ex. `priceline.com`, `expedia.com`) de vols, de voiture ou d'hôtels,

- récupération d'informations météo, traitements d'images (p.ex. rotation, redimensionnement),

- suivi de colis (p.ex. Colissimo, UPS, FedEx), pour donner aux clients des informations sur le déroulement de leur envoi postal,

- inventaire d'une entreprise,

- authentification (p.ex. le service *Passport* de Microsoft destiné à authentifier un usager auprès de toute infrastructure .NET).

Par exemple, un service de conversion de monnaie en euros prend en entrée deux arguments : (i) la valeur à convertir qui correspond à un réel, (ii) l'identificateur qui détermine le type de cette valeur (p.ex. francs, marks). Le résultat du service, suite à une invocation à l'aide de ces deux données, est un réel. Afin d'illustrer notre langage de spécification, nous donnons ici deux exemples simples de services distribués complexes, spécifiés à l'aide du langage abstrait.

Exemple 1

Nous proposons ici de spécifier un service de presse électronique orienté client sur un réseau grande échelle. Un client souhaite profiter de prestations de rendus de documents électroniques (p.ex. articles de presse, informations du jour) pour se construire un service complexe, adapté à ses besoins. Il interagit avec des services primitifs pour récupérer tout ou une partie de documents afin de les visualiser sur son terminal (p.ex. PC, ordinateur de poche). Ces services, connus du client, sont définis statiquement par leurs interfaces. Le client est à même de les composer et de les étendre (personnalisation) avec des traitements qui lui sont propres. Les documents sont fournis dans un format fixé (p.ex. texte, XML).

Supposons que le client souhaite récupérer chez deux prestataires différents des articles sur un thème fixé (p.ex. sport, économie).

1. un premier service *journal* propose un document « du jour », en français, cumulant plusieurs thèmes. Le client ne désire lire que les articles relatifs au thème fixé. Il traite ce document (à l'aide d'un traitement `select`) afin d'en extraire un nouveau document restreint au thème (p.ex. filtrage par mots-clés sur les titres, recherche de balises sur le thème).

2. un second service *actu* propose une liste de dépêches internationales sur des thèmes fixés, dans une langue étrangère et sous un format standard primitif (p.ex. texte). Le client compose ce service avec un service de traduction *trad*, supportant le format.

3. les deux documents produits, sur le thème fixé, sont ensuite traités (`fusion`) afin de produire un seul document dans un format propre au terminal du client. Ce traitement classe les informations chronologiquement ou les ordonne en fonction de critères ou de mots-clés. Le client dispose alors d'un document personnalisé, sur son thème et dans sa langue natale.

Ce service complexe peut être représenté par une expression abstraite, où les données en entrée sont les requêtes de demande de documents (jour courant d_1 et thème d'intérêt d_2) et le langage choisi pour la traduction (d_3, p.ex. français). Les traitements propres au client sont select et fusion. Ils permettent la personnalisation des résultats de services primitifs. Le service complexe est spécifié par l'expression fonctionnelle suivante :

$$\mathsf{fusion}(\mathsf{select}(journal\ d_1), trad(actu\ d_2, d_3))$$

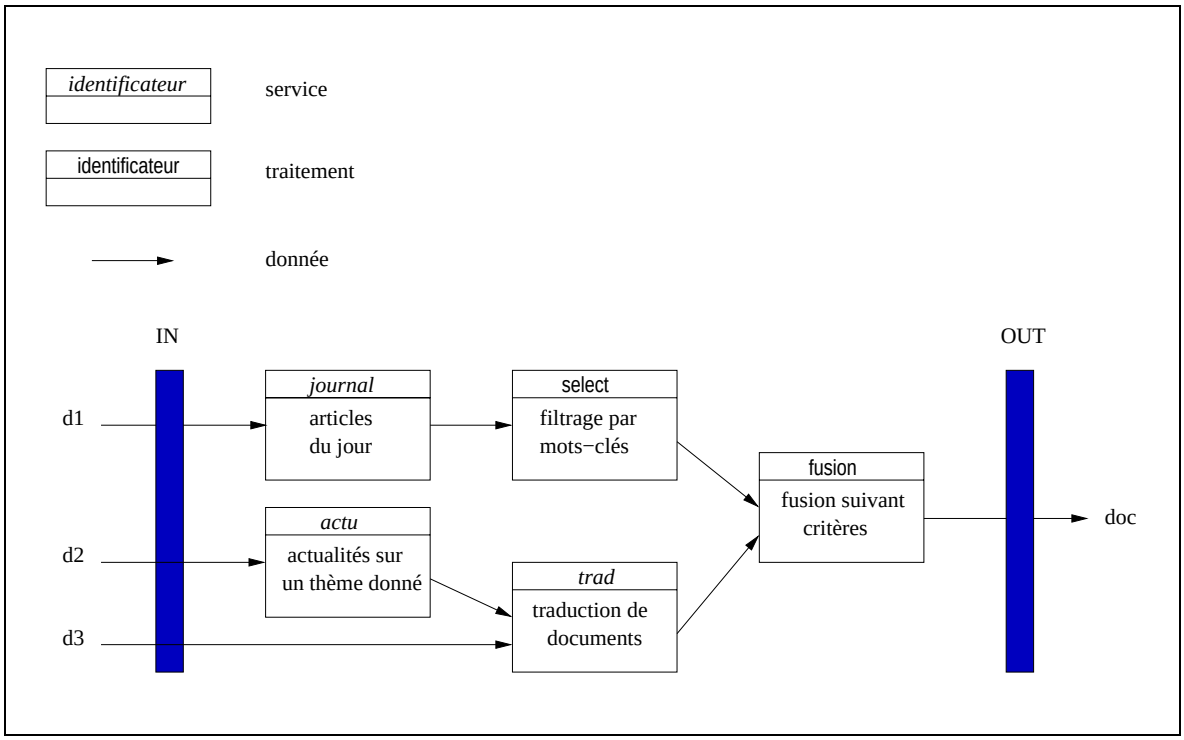

FIG. 3.3 – *Exemple 1 : un service de presse personnalisé*

De manière intuitive, un tel service peut être représenté à l'aide d'un diagramme de flot de données, comme celui présenté dans la Figure 3.3. Les boîtes représentent les services ou traitements primitifs et les flèches les flots de données entre ces fonctions. Pour une boîte, les flèches entrantes correspondent aux arguments de la fonction et les sortantes à ses résultats. Les données d_1, d_2 et d_3 sont les arguments du service complexe, et *doc* en est le résultat.

Ce service complexe pourrait être facilement étendu avec des accès à d'autres services de documents (p.ex. autres prestataires, autres thèmes, éditorial, petites annonces), à des services de rendu d'images (p.ex. carte météo, couverture de magazine, illustrations) ou encore à des services de rendu d'informations (p.ex. cotation boursière de titres, messages d'un forum de discussion).

La Figure 3.4 généralise les différentes constructions issues de la grammaire du langage abstrait, à savoir le n-uplet de données en entrée et la composition de fonctions, où f, g et h sont des identificateurs appartenant à l'ensemble *Primitif* (p.ex. $f \equiv trad$, $g \equiv journal$ et $h \equiv$select).

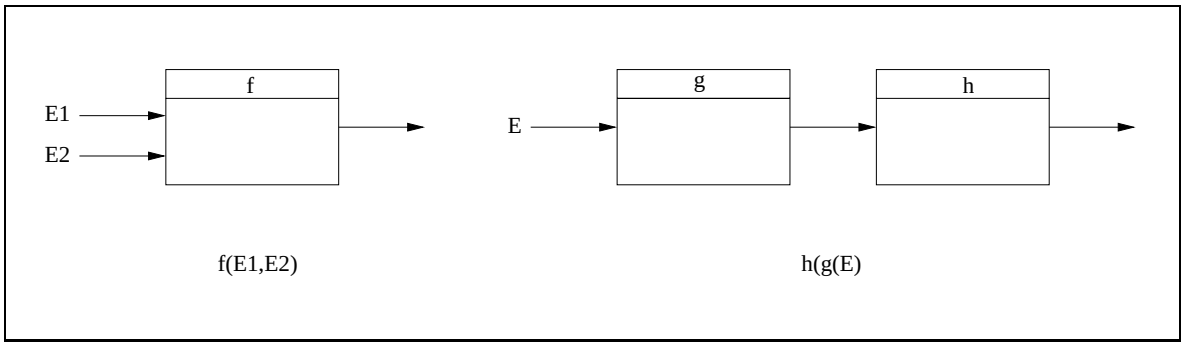

FIG. 3.4 – *Généralisation schématique*

Exemple 2

Dans le cadre d'un service de voyage, nous avons traité dans [FIR00] un service complexe de recherche combinée d'avion/train/hôtel. L'expression associée est (cf. Figure 3.5) :

$$\mathsf{correspond}(\mathit{avion}(d_1, \mathit{desti}\ d_2), \mathsf{filtre}(\mathit{train}\ d_3))$$

Elle représente un service de réservation d'avion et de tickets de train, dépendant de plusieurs critères. Par exemple :

1. le service *desti* retourne une liste de destinations possibles en fonction du critère d_2 (p.ex. une catégorie d'hôtels). Le service *avion* prend une liste de dates possibles (d_1) et une liste de destinations (*desti* d_2) pour retourner une liste de vols.

2. indépendamment, la sous-expression filtre(*train* d_3) comprend une requête (d_3) à un service « compagnie de trains » (*train*) qui fournit des horaires, filtrées (par filtre) pour ne retenir que les trains d'un jour précis se rendant à l'aéroport.

3. finalement, le traitement correspond fait correspondre les vols et les trains.

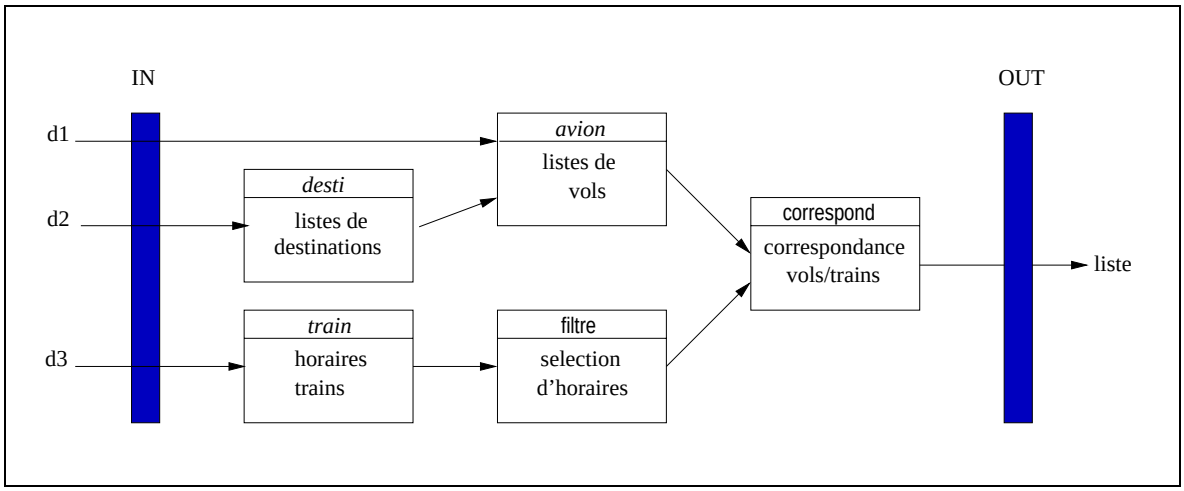

FIG. 3.5 – *Exemple 2 : un service de voyage*

3.2.4 Commentaires

Le langage abstrait est volontairement simple. Il permet d'exprimer une large gamme de composition de services et a des analyses statiques (cf. chapitre suivant). La composition de fonctions spécifie la séquence des tâches alors que les opérations indépendantes sont spécifiées dans des n-uplets. Ainsi, dans l'exemple 1, le service *actu* doit être rendu avant le service *trad* (composition), alors qu'il n'y a pas d'ordre imposé entre les services *actu* et *journal*.

3.3 Langage concret

Le but du langage concret est d'expliciter les mécanismes d'interactions utilisés dans la mise en œuvre de services complexes. En ce sens, il peut être vu comme une version compilée du langage abstrait. Il permet d'exprimer les migrations de processus afin de traiter les schémas d'interaction à base de code mobile. Il doit donc fournir un cadre unifié afin de spécifier des interactions locales ou distantes. Nous avons choisi de reposer sur les continuations [Rey93] qui sont largement utilisées dans la communauté des langages fonctionnels pour expliciter l'ordre d'évaluation des expressions. Les continuations nous permettent d'encoder « par étape» les migrations d'agents tout en restant dans le cadre fonctionnel. L'étude d'exemples (cf. section 3.3.4) qui donneront une intuition du codage des services complexes.

3.3.1 Utilisation des continuations

L'utilisation des continuations permet d'encoder dans les expressions l'ordre d'application des services et traitements. Elle facilite l'expression des analyses ultérieures car toutes les opérations se trouvent ainsi mises en séquence. Une fonction du niveau concret, à la différence des fonctions du niveau abstrait, utilise en plus un argument continuation A (la séquence des fonctions qui restent à être appliquées) qui est appliqué au résultat de son évaluation sur les données. Ces fonctions CPS (*Continuation Passing Style*) appartiennent à l'ensemble $\overline{Primitif}$. Dans la suite, afin d'éviter toute confusion, nous annotons ces fonctions avec un sur-lignage.

La version CPS d'une fonction abstraite f, telle que	$f\ d = d'$	
est une fonction	$\overline{f}$, telle que	$\overline{f}\ A\ d = A\ (f\ d)$
	soit	$\overline{f}\ A\ d = A\ d'$

Le type d'une fonction CPS est $\overline{f}$: $(exp \to ans) \to exp \to ans$, où ans est un type prédéfini, dénotant le résultat final. Une fonction particulière (notée end par la suite), de type $exp \to ans$, termine l'évaluation.

Par exemple, dans le cas où la fonction de terminaison est l'identité, une expression du type $\overline{f_2}\ (\overline{f_1}\ end)\ d$ se réécrit en :

$$\begin{aligned} \overline{f_2}\ (\overline{f_1}\ end)\ d &= \overline{f_1}\ end\ (f_2\ d) \\ &= end\ (f_1(f_2\ d)) \\ &= f_1(f_2\ d) \end{aligned}$$

3.3.2 Syntaxe concrète

La grammaire qui engendre le langage concret est donnée dans la Figure 3.6.

- une expression concrète Exp_c est une séquence de constructions **let**, chacune représentant abstraitement une interaction ($A\ D$), ou simplement un n-uplet d'identificateurs de données (D).

- une interaction est représentée par une fonction particulière définie par le non-terminal A, appliquée à des données D. Le mode d'interaction peut être une invocation distance (p.ex. RPC ou RMI), une évaluation à distance ou un agent mobile. Une représentation uniforme est utilisée pour les interactions. Les invocations distantes et évaluations distantes sont représentées comme des versions simplifiées d'agents (ce point sera plus clair sur les exemples proposés ensuite). La fonction en CPS A représente la séquence de services et de traitements qui sont à exécuter ($\overline{f}\ A$). La primitive $go_{i,j}$ dénote la migration d'un site i à un site j et permet de représenter le parcours d'un agent. Il est donc nécessaire, à ce niveau de spécification, de connaître la localisation des différents services et du client sur les sites. Pour cela, un ensemble Id_{site} contient les identificateurs des sites du réseau. Bien que les itinéraires d'agents puissent être déduits à partir de la localisation des services, les fonctions go sont indispensables pour spécifier les sites où les traitements sont exécutés. Finalement, la fonction end termine l'évaluation en rendant les résultats de l'interaction.

- les données sont des n-uplets, composés de données primaires ou de résultats intermédiaires. Les résultats intermédiaires sont représentés par les variables $(r_1, ..., r_n)$ et peuvent être réutilisés dans les expressions suivantes au même titre que des identificateurs de données.

$$
\begin{array}{rcl}
E & ::= & \textbf{let}\ (r_1, ..., r_n) = A\ D\ \textbf{in}\ E \mid D \\
A & ::= & \overline{f}\ A \mid go_{i,j}\ A \mid end \\
D & ::= & (D_1, ..., D_n) \mid d \mid r
\end{array}
$$

où $\overline{f} \in \overline{Primitif}$, $i, j \in Id_{site}$, et $d \in Id_d$

FIG. 3.6 – *Langage concret*

Notons qu'à la différence du langage abstrait, les expressions du langage concret ne comportent pas de construction n-uplet du type $(E_1, ..., E_n)$, hormis pour les données. Ainsi, fonctionnellement, tous les choix d'ordonnancement sont explicites dans une interaction.

3.3.3 Sémantique

La sémantique du langage concret (cf. Figure 3.7), sur le type $Data$, est relative à l'environnement (e) qui associe à un identificateur (c.-à-d. donnée, service ou traitement) sa définition fonctionnelle (sous forme CPS pour les éléments de $\overline{Primitif}$). Les données gardent leurs

interprétations abstraites. Les fonctions de valuation sémantiques $\mathcal{E}_c$, $\mathcal{A}_c$ et $\mathcal{D}_c$, sont liées aux trois non-terminaux de la grammaire engendrant le langage concret.

1. l'interprétation sémantique $\mathcal{E}_c$ d'une interaction (construction **let**) est usuelle et se rattache à l'équivalence :

$$\textbf{let } r = E_1 \textbf{ in } E_2 \quad \equiv \quad (\lambda r.E_2)E_1$$

   Par exemple,

$$\begin{array}{rcl} \textbf{let } r = add\ (3,2) \textbf{ in } sqr\ r & \equiv & (\lambda r.sqr\ r)\ add\ (3,2) \\ & \equiv & (\lambda r.sqr\ r)\ 5 \\ & \equiv & sqr\ 5 \quad \equiv \quad 25 \end{array}$$

2. pour chaque séquence de fonctions d'une interaction $(\overline{f}\ A)$, l'interprétation sémantique $\mathcal{A}_c$ correspond à l'application de la fonction CPS (dénotée par $\overline{f}$) à la continuation dénotée par A. L'application d'une fonction *go* est neutre du point de vue de l'interprétation sémantique. En effet, les migrations ne changent pas les résultats d'un service complexe. L'identificateur de terminaison d'une interaction (*end*) est interprété comme l'identité, où la notation $\lambda\ d.d$ correspond à l'identité, c.-à-d. une fonction qui prend un argument et le fournit en résultat.

3. l'interprétation sémantique $\mathcal{D}_c$ pour les données est standard.

$$\begin{array}{lcl}
\mathcal{E}_c : Exp_c \rightarrow Env \rightarrow Data & & \\
\mathcal{E}_c[\![\textbf{let } (r_1,...,r_n) = A\ D \textbf{ in } E]\!]\ e & = & (\lambda(r_1,\ldots,r_n).\mathcal{E}_c[\![E]\!]\ e)(\mathcal{A}_c[\![A]\!]\ e\ (\mathcal{D}_c[\![D]\!]\ e)) \\
\mathcal{E}_c[\![D]\!]\ e & = & \mathcal{D}_c[\![D]\!]\ e \\
 & & \\
\mathcal{A}_c : Exp_A \rightarrow Env \rightarrow (Data \rightarrow Data) & & \\
\mathcal{A}_c[\![\overline{f}\ A]\!]\ e & = & e\ \overline{f}\ (\mathcal{A}_c[\![A]\!]\ e) \\
\mathcal{A}_c[\![go_{i,j}\ A]\!]\ e & = & \mathcal{A}_c[\![A]\!]\ e \\
\mathcal{A}_c[\![end]\!]\ e & = & \lambda d.d \\
 & & \\
\mathcal{D}_c : Id_d \rightarrow Env \rightarrow Data & & \\
\mathcal{D}_c[\![(D_1,...,D_n)]\!]\ e & = & (\mathcal{D}_c[\![D_1]\!]\ e,...,\mathcal{D}_c[\![D_n]\!]\ e) \\
\mathcal{D}_c[\![d]\!]\ e & = & e\ d
\end{array}$$

FIG. 3.7 – *Sémantique du langage concret*

Les primitives de migration trouveront tout leur sens dans les analyses de propriétés non fonctionnelles. En effet, il est possible de déterminer, à l'aide de cette construction et au sein d'une interaction, quels sont les éléments susceptibles de migrer (c.-à-d. code et données) sur le réseau entre deux sites. Des interprétations non fonctionnelles d'expressions concrètes ouvriront la voie à différentes analyses qui porteront sur les volumes de données/code ou encore leurs niveaux de sécurité.

3.3.4 Exemples

Afin de donner l'intuition de l'interprétation des expressions concrètes, nous proposons ici quelques exemples d'interactions. Pour ne pas surcharger les exemples, nous supprimons les parenthèses et les remplaçons par des points-virgules, tout en gardant la notation avec surlignage pour éviter toutes confusions entre les fonctions du niveau abstrait et celle du niveau concret (les arités ne sont pas toujours les mêmes). Ainsi, une expression concrète du type :

$$go(\overline{f_1}(go(\overline{f_2}(\overline{f_3}(go\ end))))) \quad d$$

est écrite sous la forme (transformation syntaxique) :

$$(go; \overline{f_1}; go; \overline{f_2}; \overline{f_3}; go; end) \quad d$$

Le cadre de description de la mise en œuvre permet de traiter les trois catégories d'interactions qui nous intéressent, à savoir, invocation distante, évaluation distante et agent mobile. Le cas du code à la demande dans ce cadre est discuté plus tard (section 4.4.2, page 96). Les invocations et évaluations distantes sont vues comme des versions simplifiées d'agents, donc des migrations avec un itinéraire réduit à un site où une seule fonction est à appliquer. Cette fonction est soit un service (c.-à-d. invocation distante), soit un traitement (c.-à-d. évaluation distante). Ce cadre formel unifié facilite les analyses de qualité (cf. chapitre suivant). Nous décrivons ci-après les trois schémas d'expression concrète correspondant à l'invocation distante, l'évaluation distante et l'agent mobile avant de donner deux concrétisations de l'exemple 1 du service de presse.

3.3.4.1 Invocation distante

L'invocation distante (cf. section 1.2), initiée par un client (site c), consiste à émettre une requête vers un service distant. Une fois cette requête traitée par le service, le client récupère les résultats. La liste des fonctions à appliquer pour l'interaction se restreint donc à un service primitif s, que nous représentons par $[s]$. Le schéma général d'une invocation distante dans le langage concret est alors :

$$(go; \overline{s}; go; end) \quad D$$

Considérons le service *journal*, implanté sur un site j. Il fournit un ensemble d'articles sous la forme d'un fichier (cf. exemple 1). L'expression concrète qui correspond à une invocation distante avec le service est de la forme :

$$(go_{c,j}; \overline{journal}; go_{j,c}; end) \quad d_{req}$$

La donnée d_{req} est l'argument initial de la continuation. Dans un premier temps, cette donnée est transmise au service *journal* après migration entre le site du client et le site du prestataire. Le résultat du service est retourné au client ($go_{j,c}$) et l'interaction se termine (end). Le client récupère donc un fichier correspondant au journal électronique « du jour ». La fonction $\overline{journal}$ applique sa continuation au résultat rendu par le service *journal*. Elle est définie dans l'environnement (e) comme :

$$\lambda c.\lambda x.c\ (journal\ x)$$

L'expression concrète $(go_{c,j}; \overline{journal}; go_{j,c}; end) \quad d_{req}$ a la même interprétation sémantique (fonctionnalité) que l'expression abstraite $(journal\ d_{req})$.

3.3.4.2 Évaluation distante

L'évaluation distante (cf. section 1.3.1) consiste à transmettre et exécuter un traitement sur un site distant pour ensuite récupérer les résultats. La liste des fonctions à appliquer pour l'interaction se restreint donc à un traitement t, que nous représentons par $[t]$. Le schéma général d'une évaluation distante dans le langage concret est alors :

$$(go; \overline{\mathsf{t}}; go; end) \quad D$$

Supposons que le client souhaite réaliser le traitement select (cf. exemple 1) sur un site autre que le sien, suite à des limitations de ressources locales par exemple. Il envoie ce traitement sur un site distant, noté m pour *mainframe*, avec les données argument. Une fois le traitement réalisé, il récupère ses résultats. L'expression concrète correspondante s'écrit :

$$(go_{c,m}; \overline{\mathsf{select}}; go_{m,c}; end) \quad d$$

Cette expression concrète a la même interprétation sémantique (fonctionnalité) que l'expression abstraite $(\mathsf{select\ d})$.

3.3.4.3 Agent mobile

Les agents mobiles (cf. section 1.3.3) permettent à la fois des traitements déportés et des migrations multiples. La liste des fonctions à appliquer pour une interaction par agent mobile est composée de services et/ou de traitements. Cette liste contient au moins deux fonctions, c.-à-d. $[f_1; ...; f_n],\ n > 1,\ f_i \in Primitif$. Le schéma général d'un agent mobile dans le langage concret est alors :

$$(\overline{F}; go; \overline{F}; go; \overline{F}; end) \quad D$$

où $\overline{F}$ représente une ou plusieurs fonctions CPS.

Par exemple, les services *actu* et *trad* de l'exemple 1 peuvent être composés en une même interaction agent. Si le client souhaite personnaliser à distance le fichier reçu par le service de traduction, il peut intégrer un traitement de sélection dans son agent, $\mathsf{select'}$ (différent au niveau du type de l'argument du traitement select de l'exemple 1). L'expression correspondante à cet agent mobile, représentée dans la Figure 3.8, est de la forme :

$$(go_{c,a}; \overline{actu}; go_{a,t}; \overline{trad}; \overline{\mathsf{select'}}; go_{t,c}; end) \quad (d_{req}, d_{lang})$$

L'agent, initié chez le client migre dans un premier temps vers le site s, avec à la fois son code, ses données (la requête et le type de traduction) et le traitement $\mathsf{select'}$ (cf. réduction 1). La suite des réductions de l'expression concrète correspond à :

(1)	$(go_{c,a}; \overline{actu}; go_{a,t}; \overline{trad}; \overline{\mathsf{select'}}; go_{t,c}; end) \quad (d_{req}, d_{lang})$	
(2)	$(\overline{actu}; go_{a,t}; \overline{trad}; \overline{\mathsf{select'}}; go_{t,c}; end) \quad (d_{req}, d_{lang})$	
(3)	$(go_{a,t}; \overline{trad}; \overline{\mathsf{select'}}; go_{t,c}; end) \quad (r_1, d_{lang}),$	avec $r_1 = actu\ d_{req}$
(4)	$(\overline{trad}; \overline{\mathsf{select'}}; go_{t,c}; end) \quad (r_1, d_{lang})$	
(5)	$(\overline{\mathsf{select'}}; go_{t,c}; end) \quad r_2,$	avec $r_2 = trad\ (r_1, d_{lang})$
(6)	$(go_{t,c}; end) \quad r_3,$	avec $r_3 = \mathsf{select'}\ r_2$
(7)	$(end) \quad r_3$	
(8)	r_3	

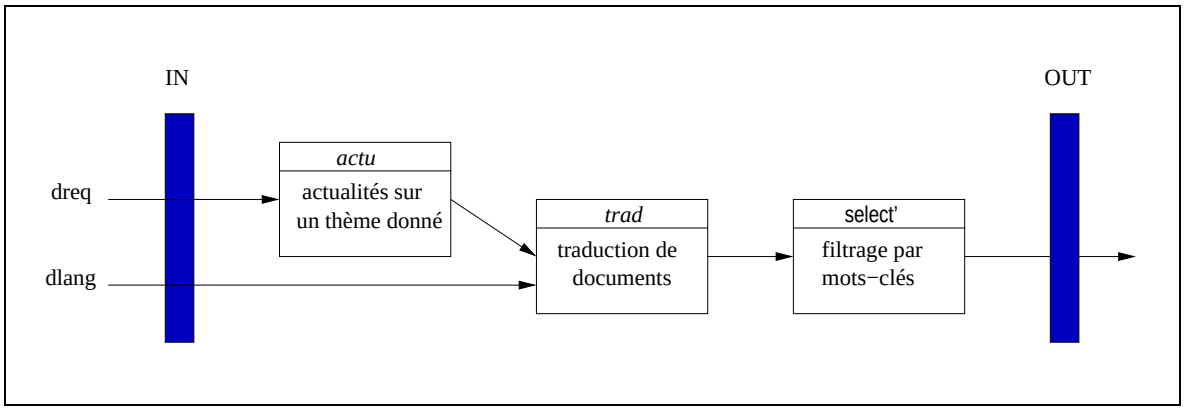

FIG. 3.8 – *Exemple 1bis : un service de traduction personnalisé*

Une fois sur le site de dépêches (s) (cf. réduction 2), il interagit avec le service *actu* et conserve les résultats avant de migrer de manière autonome vers le site de traduction (t) (cf. réduction 3). Après avoir récupéré le fichier traduit (cf. réduction 4), cet agent profite des ressources disponibles sur le site t afin d'appliquer le traitement de sélection (cf. réduction 5) pour ne conserver que certaines informations dans le fichier. Une fois le traitement appliqué, il retourne chez le client (cf. réduction 6) et termine son exécution (cf. réduction 7). Les données de l'agent sont transformées par chacun des services appliqués. Les fonctions en CPS d'une expression concrète ont en charge de transmettre les données de l'agent. Dans cet agent et pour cet exemple, la fonction $\overline{actu}$ a en charge de retransmettre la donnée d_{lang} à sa continuation. Cette fonction est ici définie dans l'environnement par :

$$\lambda c.\lambda(x,y).c\ (actu\ x,y)$$

L'interprétation de l'expression concrète correspond ici à l'expression abstraite suite à la réduction 8.

3.3.4.4 Concrétisation de l'exemple 1 du service de presse

Le service de presse, présenté dans la section 3.2.3 page 51, peut être mis en œuvre de multiples façons. Nous pouvons par exemple choisir de n'utiliser que des invocations distantes à partir du client (site c). Il est aussi possible d'utiliser un seul agent mobile pour réaliser tout le service complexe. Nous proposons ici deux expressions concrètes réalisant le service complexe.

La première utilise à la fois des invocations distantes, un traitement local et un agent mobile. Les interactions avec les services *actu* et *trad* se font par invocation distante et le traitement fusion est réalisé en local, sur le site du client. L'interaction avec le journal, personnalisée par le traitement select, est réalisée par un agent mobile. Cette mise en œuvre du service complexe consiste donc en quatre interactions, le traitement local étant réalisé en dernier lieu (les résultats des autres interactions lui servent d'entrée). Le client envoie donc un agent vers le site du journal, contenant sa requête d_1 et le traitement select. Ensuite, ou en parallèle[3], il invoque à distance le service *actu* avec une requête d_2 afin de récupérer un fichier

3. Notre sémantique dénotationnelle ne fait aucunes suppositions sur l'ordre d'exécution d'interactions indépendantes (besoin dans ce cas d'une sémantique opérationnelle). Ces choix de mise en œuvre n'ont pas d'impacts sur nos analyses.

de dépêches, retransmis ensuite au service de traduction avec la donnée d_3 (langage cible de traduction). Une fois les résultats de l'agent et du service de traduction rendus au client, le traitement de fusion est effectué en local.

L'expression concrète correspondant à cet agencement des interactions est :

$$\begin{array}{l}\textbf{let } r_1 = (go_{c,j}; \overline{journal}; \overline{\mathsf{select}}; go_{j,c}; end) \;\; d_1 \textbf{ in} \\ \textbf{let } r_2 = (go_{c,a}; \overline{actu}; go_{a,c}; end) \;\; d_2 \textbf{ in} \\ \textbf{let } r_3 = (go_{c,t}; \overline{trad}; go_{t,c}; end) \;\; (r_2, d_3) \textbf{ in} \\ \textbf{let } r_4 = (\overline{\mathsf{fusion}}; end) \;\; (r_1, r_3) \textbf{ in } r_4\end{array}$$

Les interactions peuvent être indépendantes (comme celles dénotées par r_1 et r_2) ou dépendantes (p.ex. r_3 utilise le résultat de l'interaction précédente).

Cette mise en œuvre est présentée graphiquement sur la Figure 3.9, qui fait apparaître une version possible des flots de données et de code entre les différents sites. Les services sont supposés disponibles sur des sites distincts. Une boîte représente le client ou un site proposant un service. Les flèches pleines déterminent les flots (données et code). Elles sont annotées par les identificateurs de données et de traitements. Ces transferts seront étudiés plus en détail dans le chapitre suivant. Les flèches pointillées représentent l'application des traitements, qui n'induisent pas directement de flots.

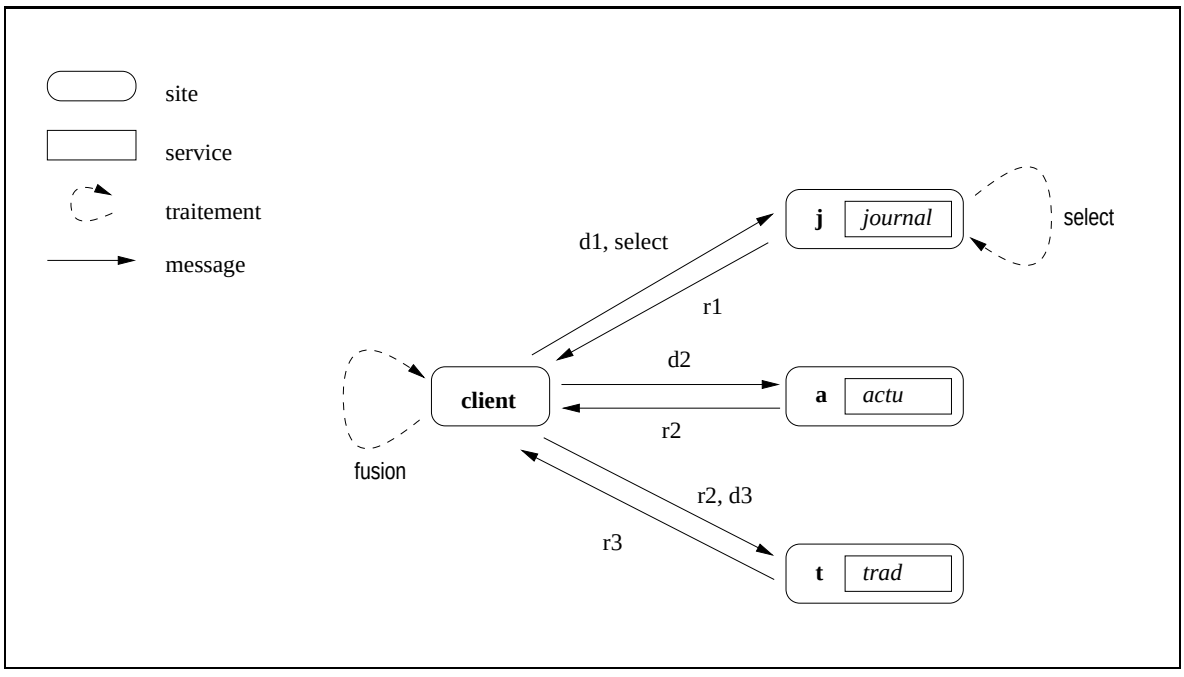

FIG. 3.9 – *Une concrétisation du service de presse*

La seconde expression concrète qui réalise le service complexe que nous présentons est :

$$\begin{array}{l}\textbf{let } r_1 = (go_{c,j}; \overline{journal}; \overline{\mathsf{select}}; go_{j,c}; end) \;\; d_1 \textbf{ in} \\ \textbf{let } r_2 = (go_{c,a}; \overline{actu}; go_{a,t}; \overline{trad}; \overline{\mathsf{fusion}}; go_{t,c}; end) \;\; (r_1, d_2, d_3) \textbf{ in } r_2\end{array}$$

Cette mise en œuvre est présentée graphiquement sur la Figure 3.10. Le traitement select est envoyé vers le site hôte de *journal via* un agent mobile. Ce traitement, une fois appliqué, peut rester dans le sac à dos de l'agent ou bien être supprimé automatiquement à la manière d'un

ramasse miettes de code. Le résultat de cette interaction (r_1) est réutilisé par un second agent qui migre vers les sites hôtes a et t avant de retourner à son propriétaire.

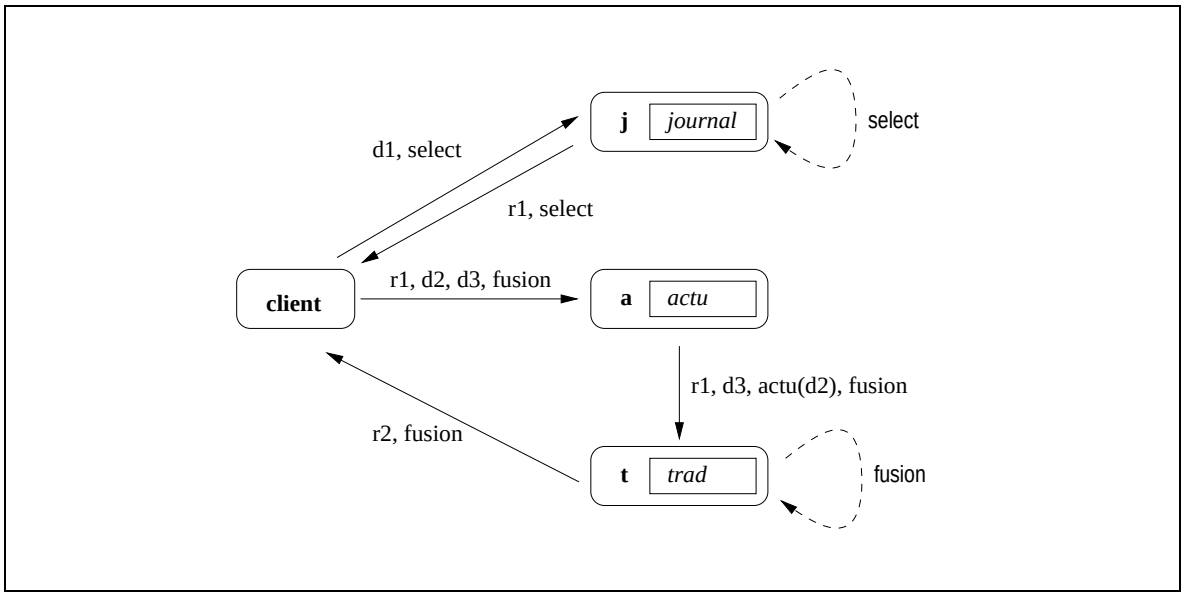

FIG. 3.10 – *Une seconde concrétisation du service de presse*

3.3.5 Commentaires

Le langage abstrait est simple et permet de spécifier une large classe de services complexes à client unique. Le langage concret sert à expliciter les interactions (c.-à-d. exécution locale, invocation distante, évaluation distante et migration d'agents) utilisées dans la mise en œuvre de ces services complexes. Un même service complexe peut s'exprimer dans chacun des deux langages par des expressions distinctes mais fonctionnellement équivalentes. Une expression concrète peut donc être vue comme un raffinement d'une expression abstraite. Nous étudions dans la section suivante comment compiler une expression abstraite en une expression concrète.

3.4 Raffinement d'expressions abstraites

Un des objectifs de la spécification de services distribués complexes à deux niveaux est de permettre la comparaison de différentes mises en œuvre utilisant des interactions distinctes. Ainsi, à une expression dans le langage de spécification abstrait correspondent des expressions concrètes qui décrivent des mises en œuvre. Ces mises en œuvre peuvent utiliser différentes interactions qui influent notamment sur les critères de qualité (p.ex. performance, sécurité, sûreté de fonctionnement). Il est possible de compiler automatiquement [FIR00] une expression abstraite en une expression concrète en choisissant le nombre, les sites d'exécution des traitements et l'itinéraire des agents (cf. les invocations et évaluations distantes étant vues comme des versions simplifiées d'agents). Ce mécanisme est présenté dans la section 3.4.1. Il est également possible (mais coûteux) de générer automatiquement toutes les expressions concrètes qui mettent en œuvre une expression abstraite donnée (cf. section 3.4.2).

3.4.1 Réécriture d'une expression abstraite

À partir d'une expression abstraite, représentative d'un service complexe, la réécriture en une expression concrète nécessite de connaître le nombre et le type des différentes interactions, l'ordre d'applications des différentes fonctions dans le cas des interactions par agents (itinéraire) et les sites d'exécution des services primitifs ainsi que des traitements. Ce dernier choix est représenté à l'aide d'un l'environnement (e_s) qui associe à chaque fonction primitive sa localisation. Les choix de mise en œuvre peuvent être représentés sous la forme d'une liste ll d'interactions. Chaque interaction de cette liste est une liste l de fonctions primitives représentant la séquence des fonctions appliquées par un agent. Par exemple, les choix correspondant à l'expression concrète, proposée dans la Figure 3.10 page 61, sont décrits par :

$$[[journal; \mathsf{select}]; [actu; trad; \mathsf{fusion}]]$$

Cette liste de listes représente deux interactions de type agent. Une invocation distante est associée à une liste avec un seul service, alors qu'une évaluation distante ou locale l'est à une liste avec un seul traitement. Une liste d'interactions valide doit inclure les différents services et traitements de l'expression abstraite. Les interdépendances existantes dans le terme abstrait doivent être respectées. Pour toute liste l, représentant la séquence des fonctions à appliquer, il faut donc s'assurer que chacune des fonctions primitives respecte les dépendances des autres fonctions primitives.

À partir d'une liste ll d'interactions valides, il est facile de compiler une expression abstraite en une concrète. Ce processus est décrit par la fonction *Abs2Conc*, donnée dans la Figure 3.11. Dans cette figure, les nouvelles notations utilisées sont :

- [] pour la liste vide,
- $[tete; queue]$ pour une liste à au moins un élément,
- $E[r_i/E_i]$ pour la substitution du terme E_i par le terme r_i dans l'expression E, par exemple $f_5(f_1(d_1, f_2\ d_2), f_4(f_3\ d_3))[r_2/f_2\ d_2] = f_5(f_1(d_1, r_2), f_4(f_3\ d_3))$,
- $e[id_i \leftarrow \lambda\text{-terme}]$ pour la mise à jour (ou la création d'une entrée id_i si l'élément n'existait pas) de l'environnement (e) avec l'élément id_i.

Abs2Conc utilise, en plus de l'expression abstraite E, une liste d'interactions ll et produit une expression **let** pour chacune des interactions (c.-à-d. construction du langage concret). Elle modifie l'environnement (e) car les fonctions sont transformées en version CPS. Le processus suppose que toutes les fonctions présentes dans ll aient été renommées afin que chacune dénote de manière non ambiguë une application dans l'expression.

- la fonction *Abs2Conc* procède comme suit :
  1. si ll est vide, l'expression E est rendue telle quelle car elle se restreint nécessairement à un n-uplet d'identificateurs de donnée (c.-à-d. d ou r).
  2. pour chaque interaction (c.-à-d. une sous-liste l), *Abs2Conc* extrait, à l'aide de la fonction *SubExp*, les sous-expressions $(E_1, \ldots, E_n)$ de E qui utilisent les fonctions primitives de l. Par exemple :

     $$SubExp\ [f_4; f_3]\ f_5(f_1(d_1, f_2\ d_2), f_4(f_3\ d_3)) = f_4(f_3\ d_3)$$
     et
     $$SubExp\ [f_4; f_3; f_2]\ f_5(f_1(d_1, f_2\ d_2), f_4(f_3\ d_3)) = (f_2\ d_2, f_4(f_3\ d_3))$$

Dans un terme E, *SubExp* recherche donc les racines de la plus petite forêt qui comprend les éléments de la liste l. Une sous-expression $(E_1, \ldots, E_n)$ est transformée en une interaction (X_1) au sens du langage concret en utilisant la fonction *Agent* décrite plus loin. *Agent* associe également un nouvel environnement (e_1) qui est réutilisé par *Abs2Conc* pour produire les parties d'expressions concrètes restantes (X_2) de ll. Pour cela, les sous-expressions $(E_1, \ldots, E_n)$ déjà traitées sont substituées par des variables r_i dans l'expression abstraite E courante.

$$
\begin{array}{l}
Abs2Conc : \ Exp_a \ \rightarrow Liste(Liste(Primitif)) \rightarrow \ Env \ \rightarrow \ (Exp_c, Env) \\
\\
Abs2Conc\ E\ []\ e \ = \ (E, e) \\
Abs2Conc\ E\ [l; ll]\ e \ = \\
\quad \underline{let}\ (E_1, ..., E_n) = SubExp\ l\ E\ \ \underline{in} \\
\quad \underline{let}\ (X_1, e_1) = \ Agent[\![(E_1, \ldots, E_n)]\!]\ l\ client\ end\ e\ \underline{in} \\
\quad \underline{let}\ (X_2, e_2) = Abs2Conc\ E[r_i/E_i]\ ll\ e_1\ \underline{in} \\
\quad (\textbf{let}\ (r_1, \ldots, r_n) = X_1\ \textbf{in}\ X_2\ ,\ e_2) \\
\\
Agent : \ Exp_a \ \rightarrow Liste(Primitif) \rightarrow \ Id_{site} \ \rightarrow \ Cont \rightarrow \ Env \ \rightarrow \ (Exp, Env) \\
\\
Agent[\![E]\!]\ []\ i\ k\ e \ = \ (go_{client,i}\ k\ E\ ,\ e) \\
Agent[\![(E_1, \ldots, f E_i, \ldots, E_n)]\!]\ [f; l]\ i\ k\ e \ = \\
\quad Agent[\![(E_1, \ldots, E_i, \ldots, E_n)]\!]\ l\ (e_s\ f)\ (\overline{f}(go_{(e_s\ f),i} k)) \\
\qquad\qquad\qquad\qquad e[\overline{f} \ \leftarrow \ \lambda c.\lambda(x_1, \ldots, x_n).c\ (x_1, \ldots, (e\ f)\ x_i, \ldots, x_n)]
\end{array}
$$

FIG. 3.11 – *Algorithme de réécriture*

– la fonction *Agent* a en charge de produire une expression du type $A\ D$ (cf. grammaire du langage concret, figure page 55). Elle utilise les sous-expressions à compiler, une séquence de fonctions (c.-à-d. liste l), le site courant de l'agent (initialement le site client), la continuation à construire et l'environnement des fonctions primitives. Cet environnement sera modifié afin d'y intégrer les nouvelles versions CPS des fonctions primitives. La construction d'une expression se fait « par la fin » et est donc initiée par la continuation *end*.

1. si la liste qui représente la séquence de fonctions est vide, l'interaction a été intégralement traitée (l'expression E est un n-uplet de données). Il reste donc à placer la migration initiale $(go_{client,i})$, où i est le site où a été traitée la dernière fonction de la liste l, en tête de la continuation k.
2. dans le cas général, la fonction f de la liste l à intégrer dans l'expression apparaît dans les sous-expressions $(E_1, \ldots, E_n)$ passées en paramètre (générées par *SubExp* qui utilise l). La fonction *Agent* est appliquée récursivement sur ces sous-expressions où la fonction f a été supprimée (c.-à-d. elle a été associée dans sa version CPS à la continuation k). L'application de l'environnement e_s (sites) aux fonctions f permet de déterminer les identificateurs de sites à associer aux primitives de migration. L'environnement e est mis à jour en insérant la version CPS $\overline{f}$ de la fonction f. La

version CPS d'une fonction f permet de transmettre dans une expression concrète les données non utilisées par f au sein de la continuation (cas de la construction n-uplet du langage abstrait).

Pour chaque liste d'interactions qui respectent les dépendances fonctionnelles, la correction de l'étape de réécriture est exprimée par la propriété suivante :

$$(Exp_c, e') = Abs2Conc\ Exp_a\ ll\ e\ \Rightarrow\ \mathcal{E}_a[\![Exp_a]\!]\ e = \mathcal{E}_c[\![Exp_c]\!]\ e'$$

Cette propriété indique que la compilation d'une expression abstraite Exp_a donne une expression concrète Exp_c, dont l'interprétation sémantique est équivalente.

3.4.2 Complexité théorique des raffinements

En théorie, la fonction *Abs2Conc*, couplée à la génération de toutes les listes d'interactions valides, peut être utilisée afin de produire automatiquement toutes les mises en œuvre possibles d'un terme abstrait. En général, le nombre de mises en œuvre différentes pour un service complexe abstrait est très grand. Nous étudions tout d'abord le cas où un seul agent est utilisé pour réaliser le service complexe. Ensuite, nous généralisons au cas où plusieurs interactions sont utilisées (mixité).

3.4.2.1 Un seul agent

Considérons le cas où l'on veuille implanter un service complexe par un unique agent. Différents itinéraires peuvent réaliser un même service complexe. L'itinéraire d'un agent est déterminé en fonction des services primitifs à appliquer. Un agent mettant en œuvre une expression abstraite de la forme $s_1(s_2(\ldots s_n)\ldots)$ a un seul itinéraire possible, celui-ci étant « câblé » dans l'expression. L'itinéraire est conditionné par l'ordre d'application des fonctions dans l'expression abstraite. Si les services primitifs de l'agent sont indépendants (c.-à-d. cas de la construction n-uplet), de nombreux choix d'itinéraires sont possibles. Une expression abstraite du type $(s_1, \ldots, s_n)$ a $n!$ mises en œuvre possibles (permutations/itinéraires) pour un unique agent.

Le choix des sites d'exécution des traitements de l'agent apporte de nouveaux choix. Il est possible d'exécuter un traitement soit sur le site courant de l'itinéraire, soit sur le site suivant, c.-à-d. sur le site du prochain service primitif à appliquer. De manière plus globale, il est possible d'effectuer ce traitement sur n'importe quel site du réseau, à savoir le site client, un serveur hôte de service primitif ou bien même un serveur dédié non lié à un service (p.ex. *mainframe*).

3.4.2.2 Plusieurs agents

Il est également possible d'entrelacer plusieurs interactions pour réaliser un service complexe (c.-à-d. invocations distantes, évaluations distantes, traitements locaux et agents mobiles). Pour chaque service primitif de l'expression abstraite, il faut envisager soit une interaction distante soit une interaction locale à l'aide d'un agent. Ainsi, une expression de la forme $s_1(s_2(\ldots s_n)\ldots)$ (itinéraire fixé) peut être découpée en 2^{n-1} mises en œuvre (arrangements). Globalement, une expression abstraite du type $(s_1, \ldots, s_n)$ va de plus nécessiter de considérer

les différents itinéraires d'agents ($n!$). Au pire cas, une expression abstraite constituée exclusivement d'un n-uplet a donc de l'ordre de $2^{n-1}n!$ mises en œuvre possibles, où n est le nombre de services présents.

Au même titre que pour un seul agent, les sites d'exécution des traitements apportent de nouveaux choix. Les traitements peuvent être appliqués en local sur le site du client, sur un site distant (c.-à-d. évaluation distante) ou au sein d'un agent mobile.

3.4.3 Commentaires

Il existe une adéquation sémantique entre le langage abstrait et son homologue concret. Pour un service complexe donné, l'évaluation d'une de ses expressions concrètes avec l'environnement associé est équivalente à l'évaluation de l'expression abstraite. Il est trivial de retrouver une expression abstraite à partir d'une concrète. Toutefois, à une expression abstraite correspondent de nombreuses expressions concrètes, dépendant des choix d'interactions et d'itinéraires. À partir d'une expression abstraite, il est théoriquement possible de générer (arrangements et permutations) toutes les listes *ll* valides des services/traitements primitifs, afin de pouvoir comparer ses différentes mises en œuvre. Devant une telle complexité (au pire cas), il n'est pas envisageable de générer toutes les expressions concrètes correspondant à un service abstrait si le nombre de fonctions présentes est supérieur à une demi-douzaine. Toutefois, certains choix de mises en œuvre d'un service complexe partagent des interactions communes. Pratiquement, ce partage peut être considéré (cf. section 5.3.3.1) pour limiter le nombre des expressions concrètes à générer. De plus, dans un souci de choix de « meilleures » mises en œuvre, l'utilisation de critères non fonctionnels (cf. chapitre suivant) peut permettre de limiter la compilation à des expressions concrètes pertinentes au regard de propriétés de qualité.

3.5 Discussion

Nous avons proposé une approche fonctionnelle simple dont le but est de permettre l'analyse de propriétés de qualité. Ces analyses nécessitent de spécifier formellement les services distribués complexes. Notre approche impose certaines restrictions qu'il aurait été difficile de pallier par l'utilisation d'autres formalismes plus complet.

3.5.1 Justification des restrictions de notre cadre fonctionnel

À partir d'une expression abstraite, nous recherchons à mettre en œuvre, grâce à des analyses automatisables, un service complexe « optimal» au regard des critères de qualité. Les limitations de notre cadre de spécification sont liées d'une part à cet objectif et d'autre part au domaine d'application traité. Le langage abstrait est minimal mais permet toutefois de décrire une majorité de services distribués complexes pour des architectures de services. Au regard des analyses qui seront présentées dans le chapitre suivant, les hypothèses de travail qui ont été posées (cf. section 3.1) restent d'actualité pour la suite du document. Vu la complexité théorique de la réécriture d'une expression abstraite en toutes les expressions concrètes qui la mette en œuvre, lever une hypothèse, non directement liée au domaine d'application, influe sur la viabilité des analyses proposées par la suite :

1. dans le cas où des sites prestataires permettraient une utilisation « abusive » des ressources locales dans une architecture de services, il devient possible à un agent mobile

en transit de réaliser des invocations distantes vers d'autres sites de prestataires. Ces interactions ne sont que rarement liées au service primitif offert par le prestataire et ne sont favorables qu'à la réalisation du service complexe pour le compte du client. Toutefois, si un site peut être utilisé comme centre d'invocation par des agents (blocage de ports de communication du prestataire), la sémantique du langage concret doit permettre d'invoquer un service primitif s_i après une primitive de migration *go*, quand la localisation de ce service ne correspond pas à celle du site cible de la primitive de migration. Dans ce cas, l'interprétation fonctionnelle d'une expression concrète reste la même. Cependant, avec une telle largesse de spécifications concrètes, le nombre d'expressions concrètes pouvant mettre en œuvre un service complexe tend à croître considérablement. Chaque site, même les sites qui ne seraient pas liés aux services primitifs de l'expression abstraite, devient un point de choix supplémentaire d'expressions concrètes. La viabilité des analyses proposées par la suite en serait limitée pour des expressions abstraites sans contraintes.

2. le cadre fonctionnel ne considère pas de primitives de communication entre les agents du service complexe, hormis sur le site du client pour transmettre des résultats. La prise en compte de rendez-vous d'agents sur des sites distants pour effectuer des échanges de messages implique d'ajouter une nouvelle construction au langage concret. Comme précédemment, le nombre d'expressions concrètes pouvant alors mettre en œuvre un service complexe tendrait à s'accroître considérablement. Dans ce cas, l'approche aurait tendance à porter sur la recherche d'une mise en œuvre optimale utilisant uniquement des agents mobiles, plutôt que de porter sur des analyses uniformes permettant de comparer les invocations distantes et les interactions avec des services primitifs par agents.

3. la spécification d'un service complexe est mono-entrée. Si des requêtes différentes sont utilisées après production de l'expression concrète, il est nécessaire de réétudier le service complexe. De plus, les données d'entrée du client potentiel sont toutes issues d'un même site. Nous ne traitons pas le cas d'un client qui réalise le service complexe à partir de plusieurs sites comme points initiaux d'interaction. Ce schéma nécessite un moyen d'échanger des résultats d'interaction entre les sites initiateurs, or nous avons pris le choix de ne pas traiter les communications distantes entre sites, autres que les invocations de services primitifs.

3.5.2 Autres formalismes

D'autres formalismes, à différents niveaux de description (abstrait/concret) peuvent être considérés. Des travaux se placent dans le domaine du *workflow* pour la spécification de dépendances entre tâches (activités) qui ouvre la voie à certaines analyses. D'autres s'appuient sur les calculs de processus pour la spécification des agents mobiles. Nous ne cherchons pas ici à donner une énumération exhaustive ni une étude en profondeur des formalismes existants vu qu'ils ne sont pas directement liés à la spécification de services distribués complexes et/ou à l'analyse statique de propriétés de qualité. Nous présentons simplement les diverses orientations de ces travaux.

3.5.2.1 Formalismes pour *workflow*

Un *workflow* consiste en l'exécution coordonnée d'un ensemble de tâches ou activités, sur différentes entités d'un système [Hol94]. Le plus souvent, à l'exécution, un processus particulier (le gestionnaire) a en charge d'activer les tâches, les arrêter, les coordonner et gérer les erreurs. Classiquement, les champs d'application de ce schéma portent sur le traitement de documents dans le domaine administratif (p.ex. traitement des missions), le traitement de commande dans les processus industriels ou encore le développement de logiciels. Le gestionnaire s'appuie sur la définition d'un *script* qui décrit les aspects nécessaires au contrôle et à la coordination entre les tâches. La coordination spécifie la séquence des tâches ainsi que les données échangées entre ces tâches. Les tâches sont généralement décrites par :

- leurs interfaces,
- leurs préconditions pour être exécutée,
- leurs actions.

Sur de nombreuses plate-formes le gestionnaire utilise un schéma d'organisation client-serveur pour gérer/coordonner les tâches à l'exécution.

Il existe de nombreux travaux qui permettent la spécification formelle des *scripts* où de multiples acteurs (utilisateurs ou logiciels) peuvent être sollicités selon une séquence donnée.. Ces travaux s'appuient sur des extensions des langages formels classiques [CCPP95]. Par exemple, les dépendances peuvent reposer sur une algèbre d'événements. Les tâches sont spécifiées en terme d'événements et leurs dépendances sont représentées à l'aide de contraintes sur les événements pour les coordonner. Certains travaux utilisent les logiques temporelles pour exprimer les dépendances (p.ex. [AMSR93]). Basés sur CTL ou LTL, des automates à états finis sont synthétisés pour décrire les différentes dépendances. Pour ordonnancer les événements, soit il faut générer le produit des automates, soit il suffit de rechercher un ensemble de chemins consistants dans les différents automates. Ces automates peuvent être de taille relativement importante. De manière similaire, des travaux utilisent des algèbres fondées sur les logiques d'actions, basées par exemple sur les langages réguliers qui correspondent à un historique linéaire des événements [Pra90]. À l'aide de ces spécifications, il devient possible de vérifier la consistance des différentes dépendances. D'autres travaux utilisent un calcul de processus [KGMW99] ou des réseaux de Pétri [SM96, vdA00], afin de produire un modèle de comportement fondé sur un système de transitions. Il est ensuite possible de vérifier des propriétés de sûreté ou de vivacité. À notre connaissance, il n'existe pas de travaux qui portent sur une approche purement fonctionnelle comme celle que nous avons suivie. Notre langage abstrait pour la spécification des dépendances dans un service complexe est plus limitée que celles permises dans les langages de *script*, mais elle reste suffisante par rapport à nos objectifs d'analyses globales.

Dans le cadre des services Web (cf. section 1.2.3), IBM a proposé en mai 2001 le langage WSFL (Web Services Flow Language) pour la description des compositions de services [Ley01], en voie de standardisation[4]. Ce langage, basé sur XML, utilise un modèle de flots pour décrire la composition de services où apparaissent les flots de données et de contrôles. Il permet de définir les liaisons entre les interfaces de services Web et est destiné à être utilisé par le gestionnaire orchestrant les interactions entre les différents services. Le modèle de flots proposé

4. Harmonisé avec le langage XLANG de Microsoft et ceux d'autres partenaires du consortium W3C.

dans WSFL est le fruit de nombreux travaux dans le domaine des *business process*. Il est plus général que ce que nous proposons dans notre langage abstrait. Il permet, par exemple, de combiner plusieurs acteurs dans un système ou de spécifier des conditions de transition entre les services (p.ex. expressions booléennes sur les valeurs des données). Toutefois, en dehors des vérifications sur les liaisons, les outils d'analyse de propriétés de qualité sur de telles spécifications n'existent pas encore et risquent d'être limitée au regard de ce que nous proposons.

3.5.2.2 Calculs pour agents mobiles

Nous avons présenté dans la section 2.4 (page 37), les travaux notoires qui portent sur la description de systèmes à agents mobiles à partir du développement des idées des calculs de processus, orientées vers la distribution [BGL00], ou du λ-calcul. Notre but au niveau langage n'est pas de se porter en concurrent des modèles à base de π-calcul qui sont relativement puissants. Nous avons choisi de suivre une approche purement fonctionnelle, dans un cadre applicatif restreint et qui supporte de manière unifiée des analyses de qualité. À notre connaissance, cette voie a été inexplorée jusqu'ici.

Dans les extensions du π-calcul pour la mobilité, les sites et les agents sont représentés par des processus qui communiquent à travers des canaux. Ces calculs ne s'intéressent pas directement au schéma client-serveur qui repose sur la seule mobilité des données. Ils sont très généraux et puissants et sont motivés par le contrôle dynamique d'un environnement où les agents et leurs actions changent constamment. À la différence de notre cadre, ces calculs ne se placent pas du point de vue du client mais plutôt du côté des domaines (contrôle des exécutions, accès aux ressources, communications et migrations des agents). Nous ne proposons pas de notion de domaine aussi fine que dans les Join, Ambient ou Seal calculs. Notre structure est plate, où un site est simplement un domaine contenant des agents et non pas d'autres domaines pouvant migrer. Dans notre langage concret, nos agents communiquent en local et non pas par envoi de messages ou invocations distantes vers d'autres sites. C'est également l'approche suivie par le calcul des Ambients, où les domaines ne peuvent communiquer à distance. La sémantique opérationnelle de modèles à base de π-calcul peut reposer sur des systèmes de transitions ouvrant la voie à des analyses. Toutefois, en pratique, il est difficile d'utiliser ces modèles puissants pour analyser statiquement des propriétés globales de qualité de service (p.ex. performance, sécurité, sûreté de fonctionnement). Ces modèles sont plus particulièrement utilisés pour un contrôle dynamique des exécutions.

En ce qui concerne le langage concret, le calcul λdist de Sekiguchi et Yonezawa [SY97] est une proposition plus proche de notre cadre (point de vue modèle d'exécution). Au même titre qu'en λdist, notre structure de domaines est plate (seule spécification des sites) et l'approche est fonctionnelle. Cependant, le langage λdist est plus fin en terme d'expressivité des mécanismes de migration de code (sémantique opérationnelle à l'aide de contextes), bien qu'il ne propose pas d'analyses de type non fonctionnelles. Notre langage concret est suffisant pour traiter les mécanismes d'interaction que nous cherchons à comparer en termes non fonctionnels.

Sur le cadre fonctionnel proposé, le chapitre suivant s'attache à définir les différentes analyses de qualité de services distribués complexes qui peuvent être produites.

Chapitre 4

Analyses de qualité de services distribués complexes

Lors de la conception d'un service complexe, il est important de pouvoir garantir/optimiser des propriétés non fonctionnelles. Pour le compte de clients, il convient donc de s'intéresser aux propriétés de qualité (cf. introduction) que sont la performance, la sécurité et la sûreté de fonctionnement. Garantir au mieux des critères de qualité n'est pas une tâche triviale. Il faut chercher à minimiser les données en transit à travers le réseau (performance), protéger les données du client (sécurité) ou encore supporter les défaillances (fiabilité) des sites et liens du réseau. Les mécanismes d'interaction, les traitements personnalisés ainsi que la distribution des données, du code et des ressources (p.ex. services, client) sur les différents sites du réseau influent directement sur la qualité finale d'un service complexe.

Ce chapitre propose des analyses de qualité vouées à assister un concepteur de services complexes. Le concepteur peut être le client, qui compose simplement de l'existant, ou un expert (prestataire de service) qui construit un service complexe à partir de :

- services existants (internes ou externes à son réseau),
- de services primitifs qu'il (fait) développe(r).

Les analyses statiques proposées s'appuient sur notre cadre de spécification fonctionnel, qui fournit une abstraction de haut niveau destinée à réduire la complexité de l'étape de conception. Elles visent à étudier les différents choix de mises en œuvre d'un service complexe. Le but est de sélectionner les configurations qui conviennent le mieux aux besoins en qualité.

Les analyses s'appuient sur le langage concret, qui est un modèle d'exécution explicitant les interactions (mécanisme, migrations, etc.). Les propriétés de qualité traitées concernent :

1. la performance *via* l'évaluation du trafic total (c.-à-d. volume de données) engendré par un service complexe dans le réseau,
2. la sécurité *via* la vérification de la confidentialité et de l'intégrité des données/code étant donnés des niveaux de sécurité,
3. la fiabilité *via* la vérification de la tolérance aux défaillances des interactions.

Les différentes analyses statiques basées sur cette représentation des propriétés de qualité nous permettent de :

1. vérifier par inférence sur la grammaire du langage concret, étant données des hypothèses sur la plate-forme d'exécution, si une mise en œuvre d'un service complexe respecte un seuil de performance, assure la confidentialité (resp. intégrité) des données et codes du client, ou est tolérante aux défaillances.

2. comparer, en terme non fonctionnel, différentes mises en œuvre (à base d'invocation distante, d'évaluation distante et d'agents mobiles) d'un même service complexe.

3. optimiser ou garantir, en combinant plusieurs mécanismes d'interaction, les contraintes de qualité au plus tôt lors de la conception d'un service complexe. En effet, imposer un certain mécanisme d'interaction (p.ex. invocation distante) peut fortement compromettre le déploiement effectif en fonction de critères de qualité (p.ex. performance). Plutôt que de laisser les services systèmes ou intergiciels de la plate-forme (cf. sections 1.2.2 et 1.4.2.1) tenter de traiter les problèmes de qualité, il peut être favorable de combiner différents types d'interactions pour tenter de garantir/optimiser des propriétés de qualité plus en avant dans les étapes de l'ingénierie. « Quel est le meilleur mécanisme (p.ex. RPC, migration d'agent, etc.) à mettre en place pour couvrir les besoins en qualité ? ». La réponse à cette question est jusqu'ici laissée aux seules expériences et expertises du concepteur. Notre approche permet de donner une première réponse à cette question dans les architectures de services.

4. synthétiser des éléments d'architectures pour l'exécution de services complexes. Ces éléments (p.ex. sites, liens), une fois déployés, utilisent des mécanismes connus pour garantir/optimiser des propriétés de qualité.

5. guider l'utilisation de traitements personnalisés en-capsulés dans des agents mobiles et aptes à améliorer ou garantir certaines propriétés de qualité.

Dans ce chapitre, la section 4.1 étudie l'interprétation sémantique non fonctionnelle des expressions concrètes au regard de chacune des propriétés de qualité traitées, à savoir la performance, la sécurité et la fiabilité. Dans la section 4.2, nous donnons l'ensemble des analyses qui peuvent en découler. La section 4.2.1 présente une analyse qui s'appuie sur une spécification du réseau où tous les services primitifs sont placés et décrits au regard des propriétés traitées. La section 4.2.2 se focalise sur les analyses possibles lorsque des éléments du réseau (p.ex. localisation/propriétés de sites ou liens) ne sont pas complètement définis. La section 4.3 considère une analyse permettant d'ajouter des traitements personnalisés aux agents mobiles. Dans la section suivante, nous envisageons des extensions possibles au langage abstrait. Ce langage étant particulièrement simple pour supporter de manière effective les analyses proposées, des spécifications plus fines sont parfois nécessaires. Nous y discutons de la faisabilité des analyses présentées pour chacune des extensions.

4.1 Analyse de la qualité de mise en œuvre de services composites

Au regard des services complexes, nous présentons ici précisément les trois propriétés non fonctionnelles qui nous intéressent, à savoir la performance, la sécurité et la fiabilité.

Ces propriétés de qualité, perçue par le client d'un service complexe, sont importantes à considérer, particulièrement dans le cas des réseaux grande échelle ouverts. Elles sont de plus représentatives quant aux avantages et inconvénients des agents mobiles (cf. chapitre 2). Nous traitons ces propriétés en associant des propriétés élémentaires à chacun des objets de notre cadre (c.-à-d. données, services, traitements, sites et liens) (cf. section 3.1.2). Chacune d'elle est relative à un ensemble de valeurs ordonnées (c.-à-d. le domaine) et à un opérateur permettant de calculer, pour une propriété, la valuation sémantique d'une expression en fonction des propriétés de ses sous-expressions. Ces informations additionnelles sont contenues dans des environnements qui associent à un identificateur donné sa valeur dans le domaine.

L'analyse de base porte sur l'**inférence de la qualité de mise en œuvre**. Son interprétation sémantique sert de corps aux analyses qui seront présentées ensuite. Elle consiste à déterminer, à partir d'une expression concrète et des environnements, la valeur d'une propriété pour chacune des interactions de l'expression. Cette analyse [FIR00], de complexité linéaire en fonction des services/traitements présents dans une expression, se définit récursivement sur la grammaire du langage (induction structurelle). Nous proposons ci-après cette analyse pour les trois propriétés non fonctionnelles considérées.

4.1.1 Performance

La performance d'un service complexe dépend d'une part des capacités de calcul et de communication des sites et d'autre part des liens (canaux de communication). La performance de calcul des sites d'un réseau est liée aux matériels employés. Les techniques de communication entre les sites (p.ex. protocoles) sont liées à la plate-forme utilisée (p.ex. CORBA, .NET). Les performances des canaux de communication dépendent du réseau physique. Les débits peuvent varier considérablement entre deux sites, particulièrement dans les réseaux grande échelle. Dans ce type de réseau, la localisation des sites influe donc sur les temps de réponse.

Un premier moyen d'optimiser les communications entre deux sites dans le cas des invocations distantes consiste à rapprocher, autant que faire se peut, les entités communicantes. Dans un environnement de services distribués, l'usage des agents mobiles peut également contribuer à améliorer les performances d'un système (cf. section 2.1, page 23). Les agents permettent de déporter des unités d'exécution directement vers les prestataires de services. Pour le compte d'un client, ils ouvrent la voie à la réduction de la consommation en bande passante en déportant les calculs vers les données plutôt que de rapatrier ces données pour les traiter localement. Il devient alors possible de chercher à améliorer les performances d'un système en s'intéressant aux mécanismes d'interaction, en plus de la localisation.

Nous ne cherchons pas à traiter directement le problème des temps d'exécution, qui est particulièrement délicat dans les systèmes ouverts du fait de la nature non prédicative des entités participantes. En effet, les temps de réponses sur les réseaux grande échelle sont variables et imprévisibles statiquement car ils sont liés à l'instabilité et à l'encombrement du réseau. Les travaux dans ce domaine reposent largement sur des approches probabilistes. La notion de performance que nous considérons est en terme de volume de données échangées sur le réseau. La prise en compte de la bande passante des canaux dans notre cadre, qui détermine le volume de données transmissible en un temps donné, peut toutefois nous amener à considérer grossièrement la performance d'un point de vue temporel.

4.1.1.1 Domaine

Les volumes de données engendrés par les interactions d'une expression concrète sont représentés dans notre cadre par des entiers positifs (domaine de l'interprétation sémantique). Les services/traitements abstraits sont ainsi caractérisés par des fonctions de $Int^n \rightarrow Int^n$. Une représentation symbolique de ces tailles aurait été plus générique mais beaucoup plus difficile à normaliser et à comparer. L'opérateur qui combine les différents transferts de données entre sites dans une interaction est tout naturellement l'addition. Une valeur de performance est associée à chaque objet de nos langages de la façon suivante :

- à une donnée d est associée une valeur dans le domaine, c.-à-d. sa taille (p.ex. un fichier de 1024Ko).
- à un service primitif s (ou traitement t) est associé une fonction donnant la taille des résultats en fonction de la taille des arguments. Ainsi, à un traitement de compression compress peut être associée la fonction $\lambda x.x/3$, qui indique que la compression fait gagner un facteur 3. Les tailles des résultats d'une fonction peuvent dépendre ou non des tailles de ses arguments. Bien qu'il ne soit pas toujours possible de connaître statiquement la taille de données ou de résultats, notre approche considère des approximations dans ces cas (p.ex. taille moyenne ou maximale des résultats).
- il est nécessaire de connaître la taille du code des agents susceptibles de transiter sur le réseau dans le cas des interactions par agents ou évaluation distante. Cette taille correspond à la taille du *byte code* des traitements, associée à la taille du code des interactions agent.
- si considérée, la bande passante (en bits/sec) est représentée à l'aide d'un coefficient $\alpha_{i,j}$ associé à l'opérateur de migration $go_{i,j}$. Ce coefficient représente l'inverse de la bande passante sur le lien (c.-à-d. connexion) entre les sites i et j. Lors de l'analyse, ce coefficient est considéré comme statique. Il est nul si la bande passante est infinie (p.ex. sites i et j identiques). Par exemple, le coût d'une migration entre les sites i et j d'un agent de taille T (c.-à-d. code + données + état), est $\alpha_{i,j} * T$.

Ces informations sur les objets sont définies dans des environnements :

1. l'environnement e (cf. section 3.1.2) contient les définitions de performance relatives aux fonctions,
2. l'environnement e_c (code) donne la taille du code source des fonctions primitives. Dans le cas des services, cette taille est définie comme nulle, les services étant liés aux serveurs et ne migrant pas sur le réseau,
3. l'environnement e_s (site ou place) donne, en plus de la localisation des services, la bande passante entre deux sites.

4.1.1.2 Analyse

Suivant les interactions mises en œuvre pour réaliser un service complexe, les données, codes et résultats en circulation sur le réseau diffèrent et empruntent des itinéraires différents. Le calcul du volume de données généré par les interactions d'une expression concrète se définit

de manière récursive (induction structurelle) sur la grammaire du langage en utilisant les environnements e, e_c et e_s, relativement au critère de performance :

1. l'environnement e, en plus des définitions des fonctions abstraites f, comprend celles des fonctions concrètes $\overline{f}$ (c.-à-d. version CPS). Ces définitions de fonctions concrètes sont à fournir ou alors sont issues de la compilation automatique d'une expression concrète à partir d'une expression abstraite (cf. fonction *Agent*, section 2, page 63). Par exemple, la fonction abstraite compress, définie dans l'environnement e par $\lambda x.x/3$, peut être [1] définie dans sa version CPS par $\lambda c.\lambda x.c\ (x/3)$. L'environnement e contient donc les signatures des objets, sous forme de continuation pour les fonctions primitives.

2. les environnements e_c et e_s sont des constantes globales.

La fonction de valuation sémantique $Perf_E$, qui calcule le volume de données induit par une expression concrète, est donnée dans la Figure 4.1. La définition proposée reste abstraite, en ce sens qu'elle n'est pas directement liée à une plate-forme d'exécution donnée. En effet, diverses techniques peuvent être employées pour réaliser la migration d'agents. L'analyse peut être facilement adaptée pour une plate-forme particulière. Nous supposons ici que le code d'un agent est la concaténation du code de ses traitements et qu'une fois qu'un traitement est exécuté, il reste au sein de l'agent. Les données transportées par un agent le sont jusqu'à ce qu'elles soient utilisées.

$$
\begin{array}{lcl}
Perf_E \ : \ Exp_c \to Env \to Int & & \\
Perf_E[\![\textbf{let } (r_1,\ldots,r_n) = A\ D \textbf{ in } E]\!]\ e & = & Perf_A[\![A]\!]\ e\ (Source[\![A]\!])\ (\mathcal{D}_c[\![D]\!]\ e) + \\
 & & Perf_E[\![E]\!] \\
 & & \qquad (e[(r_1,\ldots,r_n) \leftarrow \mathcal{A}_c[\![A]\!]e\ (\mathcal{D}_c[\![D]\!]e)]) \\
Perf_E[\![D]\!]\ e & = & 0 \\
 & & \\
Perf_A \ : \ Exp_A \to Env \to Int \to Data_{Int} \to Int & & \\
Perf_A[\![\overline{f}\ A]\!]\ e\ c\ D & = & e\ \overline{f}\ (Perf_A[\![A]\!]\ e\ c)\ D \\
Perf_A[\![go_{i,j}\ A]\!]\ e\ c\ D & = & Perf_A[\![A]\!]\ e\ c\ D + \\
 & & \underline{si}\ i = j\ \underline{alors}\ 0 \\
 & & \qquad \underline{sinon}\ (c\ +\ Somme\ D) * (e_s\ \alpha_{i,j}) \\
Perf_A[\![end]\!]\ e\ c\ D & = & 0 \\
 & & \\
Source : \ Exp_A \to Int & & \\
Source[\![\overline{f}\ A]\!] & = & e_c\ f\ +\ Source[\![A]\!] \\
Source[\![go_{i,j}\ A]\!] & = & e_s\ \alpha_{go} + Source[\![A]\!] \\
Source[\![end]\!] & = & 0
\end{array}
$$

FIG. 4.1 – *Évaluation de la performance*

La fonction de valuation sémantique $Perf_E$ est définie comme suit :

1. le coût d'une expression **let** est la somme des coûts de ses sous-expressions. Pour cette

1. La version CPS peut utiliser des arguments supplémentaires afin de représenter la transmission de données non utilisées par la fonction, cf. section 2. Elle est donc directement dépendante de l'interaction où elle apparaît.

construction, l'analyse se décompose en l'étude de l'agent courant $A\ D$ et l'étude des interactions suivantes E. Pour le deuxième point, l'interaction $A\ D$ est évaluée par $\mathcal{A}_c[\![A]\!]e\ (\mathcal{D}_c[\![D]\!]e)$ (cf. section 3.3.3) pour obtenir les tailles des résultats intermédiaires $(r_1, \ldots, r_n)$. $\mathcal{D}_c[\![D]\!]$ dénote la taille des données arguments D de la continuation A. L'application des fonctions CPS de la continuation A sur ces tailles fournissent les tailles des données $(r_1, \ldots, r_n)$. Les valeurs associées à ces identificateurs r_i sont ajoutées à l'environnement e. Elles seront utilisées par la fonction de valuation sémantique $Perf_E$ pour l'étude des interactions suivantes qui utilisent ces identificateurs.

2. l'analyse d'un agent $A\ D$ (les invocations distantes et les évaluations locales ou distantes étant vues comme des versions simplifiées d'agents) se fait par la fonction de valuation sémantique $Perf_A$. Cette fonction prend en paramètre l'environnement courant et la taille du « sac à dos de l'agent » (c.-à-d. son code c et ses données D). Le coût $Perf_A$ d'une interaction se définit par application successive des différentes fonctions de la continuation. Ainsi, l'application d'une fonction primitive $\overline{f}$ n'induit pas de trafic et revient à calculer le coût induit par le reste de la continuation avec le nouveau sac à dos qui contient les résultats de f (c.-à-d. application de f sur le n-uplet de données d). Par exemple, à partir des versions CPS, $e\ \overline{f}\ (Perf_A[\![A]\!]\ e\ c)\ D$ s'évalue en $Perf_A[\![A]\!]\ e\ c\ (f\ D)$ quand les arités en version CPS sont gardées. Une migration $go_{i,j}$ n'est pas neutre au regard des critères non fonctionnels. Le coût d'une migration est le coût de la suite de l'interaction, auquel nous ajoutons le coût induit par la transmission du sac à dos sur le canal (c.-à-d. code c et somme des tailles des données intermédiaires D). Les données intermédiaires sont accumulées par la fonction *Somme*. À ce stade, le coefficient $\alpha_{i,j}$, utilisé pour représenter l'inverse de la bande passante, peut être employé. Si les sites i et j sont identiques, la migration est neutre. Finalement, la fonction *end* termine l'analyse d'une interaction.

3. la taille du code est déterminée par la fonction de valuation sémantique *Source* qui somme la taille des traitements présents dans l'agent. En fonction de la plate-forme utilisée, cette taille peut être fixée pour toute l'interaction, ou bien variable si l'on dispose d'un mécanisme de récupération de code une fois les traitements appliqués. La taille du code associé aux instructions de migration $go_{i,j}$ et aux invocations locales de services peut être prise en compte (p.ex. $e_s\ \alpha_{go}$).

Il est envisageable d'intégrer des coefficients à même de prendre en compte le coût induit par des mécanismes particuliers liés à la plate-forme cible (p.ex. assemblage, sérialisation). Nous pouvons également modifier l'analyse en fonction de mécanismes d'interaction par agents particuliers (p.ex. migration faible/forte, pas de retour de l'agent chez le client mais juste émission des données après applications du dernier service).

4.1.1.3 Exemple

Afin de donner l'intuition de cette analyse (qui prend une forme assez proche pour les autres propriétés de qualité et qui est à la base des analyses ultérieures), nous détaillons l'analyse de l'agent à migrations multiples avec traitement déporté, proposé dans la section 3.3.4.3 :

$$(go_{c,s}; \overline{actu}; go_{s,t}; \overline{trad}; \overline{\mathsf{select}'}; go_{t,c}; end)\quad (d_{req}, d_{lang})$$

Dans l'environnement e, supposons que les données/fonctions associées au service abstrait $\mathsf{select}'(trad(actu\ d_{req}, d_{lang}))$ soient décrites pour la performance en octets :

$$\begin{aligned} d_{req} &: 8 \\ d_{lang} &: 1 \\ actu &: \lambda x.1024 \\ trad &: \lambda(x,y).x \\ \mathsf{select}' &: \lambda x.(x/2) \end{aligned}$$

Le service primitif *actu* fournit un résultat estimé de 1024 octets quel que soit la taille de la requête. Le service de traduction prend en entrée le document source et le langage cible désiré. La taille de son résultat est de taille identique à celle de son premier argument. Le traitement de sélection divise en moyenne la taille de son argument par 2. On suppose que la taille du code source de ce traitement est de 128 octets (cf. environnement e_c). Pour simplifier, nous ne considérons pas les bande passantes des différents canaux, où les sites c, s et t sont distincts. De même, nous restreignons la taille des agents aux données paramètres et traitements (p.ex. pas de prise en compte de la taille des primitives de migration, des invocations locales de service, des paramètres inhérents à la plate-forme cible). Les versions CPS des fonctions primitives, pour l'expression concrète donnée, sont (génération automatique) :

$$\begin{aligned} \overline{actu} &: \lambda c.\lambda(x,y).c\ (1024, y) \\ \overline{trad} &: \lambda c.\lambda(x,y).c\ x \\ \overline{\mathsf{select}'} &: \lambda c.\lambda x.c\ (x/4) \end{aligned}$$

Notons qu'à la différence des deux autres fonctions, la version CPS de *actu* utilise un argument supplémentaire. Il sert à transmettre la requête d_{lang} au service de traduction au sein de la continuation à étudier.

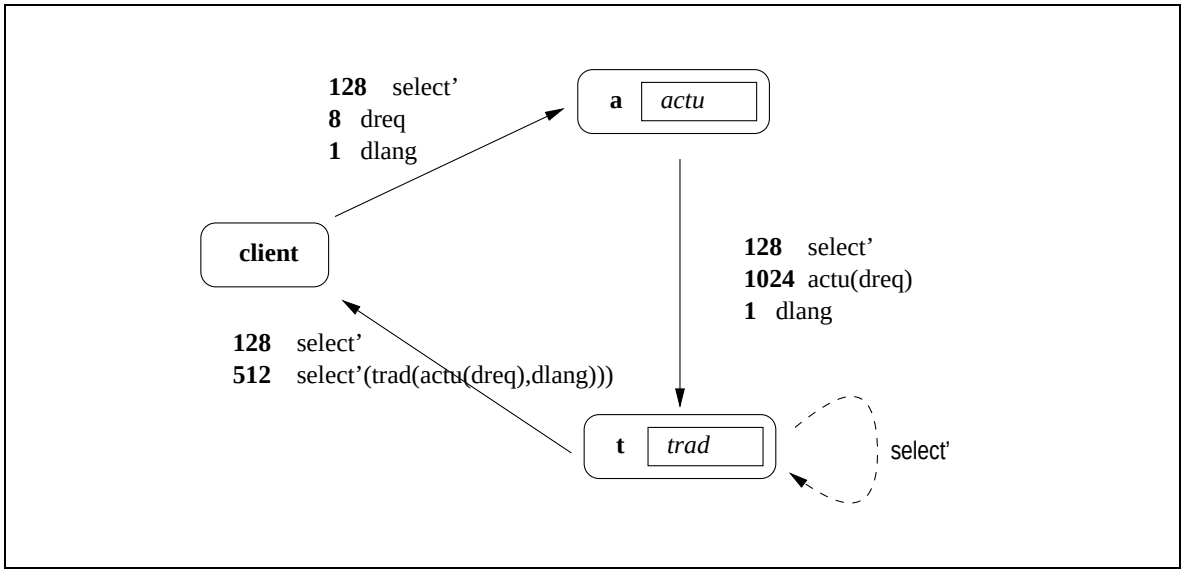

FIG. 4.2 – *Volume de données pour une interaction par agent*

L'application de la fonction de valuation sémantique $Perf_E$ à l'expression concrète, avec les environnements décrits, permet d'inférer le trafic total engendré par l'interaction. Les différents

volumes de données induits sur les canaux sont représentés sur la Figure 4.2. Les valeurs en gras représentent les tailles associées aux identificateurs et expressions.

La fonction de valuation sémantique $Perf_A$ dispose comme arguments de l'environnement (qui contient les versions CPS des fonctions), la taille du code source de l'agent calculé par la fonction *Source* (ici 128) et la taille des données initiales, c.-à-d. le doublet $(8, 1)$.

1. à la rencontre du premier go, le processus consiste à déterminer la taille du sac à dos devant transiter sur le lien (c, s), où les sites c et s sont distincts. Cette taille est la somme de la taille de select' et la taille du n-uplet argument, soit $128 + 8 + 1$. L'application de *actu* sur le n-uplet permet de déterminer le volume du n-uplet résultat, soit $(1024, 1)$.

2. le coût de la migration sur le lien (s, t) comprend la taille du traitement, du fichier résultat $(actu(d_{req}))$ et la donnée d_{lang} qui est retransmise telle quelle, soit $128+1024+1$. L'application du service de traduction porte sur les deux données arguments fournies par l'agent. Le résultat est de taille égale au fichier en entrée, 1024 octets. Le reste de la continuation (c.-à-d. $\overline{\mathsf{select}'}; go_{t,c}; end$) est ensuite appliquée. Le traitement select' est fait en local sur le site de traduction. Il fournit une version du fichier deux fois plus petite que le fichier traduit en entrée, soit 512 octets.

3. selon l'analyse proposée (la valuation sémantique utilisée), le coût de retour de l'agent (c.-à-d. $go_{t,c}; end$) porte (somme) sur le traitement et ce dernier fichier. Au final, avec les hypothèses données, le coût de cette interaction est de $(128 + 8 + 1) + (128 + 1024 + 1) + (128 + 512)$.

L'application du traitement en déporté aura permis de décroître la taille des données à rapporter au client, mais aura nécessité de transmettre trois fois la taille du code du traitement.

4.1.2 Sécurité

D'un point de vue « besoin client », la sécurité d'un service complexe concerne la protection des données clientes sensibles en circulation sur le réseau (p.ex. applications de commerce électronique). Le client d'un service complexe, dans un système ouvert, peut ne pas vouloir laisser circuler en clair des informations personnelles sur certains liens ou sites et désirer s'assurer que les résultats qu'il reçoit et les exécutions de ses traitements n'ont pas été falsifiés. La confidentialité porte sur la non-divulgation des informations (cf. section 2.2). L'intégrité porte sur la non-modification des informations. Notre analyse doit permettre de s'assurer que des données ou traitements transitent sans être directement lisibles ou modifiables par des intrus.

Dans le domaine des interactions bidirectionnelles par invocation distante, l'utilisation du chiffrement est à même d'assurer la protection des données en transit. Dans certains cas (p.ex. numéro de carte bancaire), il faut s'assurer que le serveur contacté dispose d'un niveau de confiance suffisant. Les données transmises par un client à un serveur peuvent parfois être retransmises par le serveur ou accédées par des tiers. En général, un client a confiance envers des serveurs de son Intranet ou bien ceux d'institutions (p.ex. gouvernementales ou financières). Pour les interactions à l'aide d'agents mobiles, les aspects de sécurité sont plus critiques (cf. section 2.2). La protection des agents (p.ex. code, données et état) envers des sites malintentionnés (p.ex. prestataires malhonnêtes) est encore loin d'avoir trouvé des solutions standardisées. Il ne suffit plus de chiffrer les communications entre un client et un serveur pour

protéger les données en transit. Les données arguments ou résultats confidentielles d'un client doivent rester inconnues des sites malveillants.

Les codes confidentiels (c.-à-d. traitements) ne doivent pas être lisibles par certains sites qui pourraient en retirer de l'information (cf. section 2.2.2). De même, il n'est pas souhaitable qu'un site modifie des données qui seront utilisées par un site suivant dans l'itinéraire (atteinte à l'intégrité). Il faut s'assurer de la bonne exécution des agents (c.-à-d. itinéraires et traitements). Les traitements d'un client doivent pouvoir être exécutés de manière intègre, comme s'ils avaient été exécutés sur le site du client. Les données d'un agent peuvent transiter sur certains sites sans y être utilisées (elles servent d'entrée pour des services ultérieurs de l'itinéraire). L'agent doit avoir l'assurance que ni son code ni ses données ne seront consultés ou modifié par les sites de son itinéraire. Ce point est particulièrement important dans le cas des services de commerce électronique. Si un agent réalise un service de comparaison de tarifs, il faut s'assurer qu'un prestataire ne va pas voir ou modifier les tarifs d'autres prestataires récoltés par l'agent.

4.1.2.1 Domaine

Pour traiter le délicat problème de la sécurité (confidentialité et intégrité) du point de vue du client d'un service complexe, nous choisissons d'évaluer le niveau de confiance que l'on peut porter à une interaction donnée. Pour ce faire, nous associons aux données et traitements des clients un niveau dans un treillis de sécurité, qui représente le niveau de sensibilité. Des niveaux de confiance sont associés aux sites et canaux de communication, en fonction d'identité ou de mécanismes/matériels/recours déployés. Le domaine de valeur et l'analyse proposés pour la confidentialité sont quasi-identiques avec ceux de l'intégrité. Par la suite, nous décrivons principalement la confidentialité, l'intégrité s'adapte trivialement au cadre proposé pour la confidentialité.

Les données arguments ou résultats confidentielles d'un client doivent rester inconnues des sites malveillants. Les codes confidentiels (c.-à-d. traitements) ne doivent pas être lisibles par certains sites qui pourraient en retirer de l'information. Pour représenter la confidentialité des données, nous considérons ici simplement deux niveaux associés aux objets : public et secret, avec l'ordre partiel public $\sqsubseteq$ secret (resp. nous utilisons les termes non-sûr et sûr pour l'intégrité). L'approche s'étend facilement à tout treillis de sécurité (p.ex. non-classifié, public, confidentiel, secret, top-secret). Pour combiner deux niveaux, l'opérateur utilisé $\sqcup$ est le plus petit majorant (p.ex. public $\sqcup$ secret = secret). Au même titre que pour la performance, les environnements contiennent les niveaux des objets de base :

- à une donnée fournie initialement par un client est associé son niveau de confidentialité.
- à un service primitif ou traitement est associée une fonction donnant le niveau de confidentialité des résultats en fonction du niveau des arguments. Ainsi, à un identificateur f pourrait être associé la fonction $\lambda x.secret$. Cette signature signifie que pour tout niveau de la donnée en entrée, l'application de f fournit toujours un résultat confidentiel. C'est par exemple le cas d'un service bancaire qui fournit le solde du compte d'un client. Les niveaux du résultat d'une fonction peuvent dépendre ou non des niveaux de l'argument.
- à un traitement est associé le niveau de confidentialité de son code (resp. d'intégrité de son code). Dans l'environnement e_c, il est définit comme secret si le client ne souhaite pas qu'un intrus ou un site malhonnête lise son code (resp. sûr si le traitement nécessite d'être non modifié et exécuté de manière intègre).

- à un *go* est associé un niveau qui reflète la sécurité de la liaison (p.ex. chiffrée ou non).

- à un site est associé son niveau de confiance dans l'environnement e_s. Un serveur de niveau secret est un site de confiance, c.-à-d. il est considéré comme honnête envers les données et code qui transitent (p.ex. site gouvernemental, site Intranet). Il ne divulguera pas les informations reçues. Un serveur de niveau public ne devra pas être considéré comme de confiance et est à même de divulguer, volontairement ou non, les informations reçues. Pour l'intégrité, un niveau est associé à un site qui reflète l'exécution intègre du code des agents et leur non-modification. Si un site est classé comme non sûr, il est susceptible d'exécuter de manière falsifiée un traitement ou encore de l'exécuter plusieurs fois (p.ex. ordre d'achat de produits). Les niveaux de confidentialité et d'intégrité sont retenus en fonction d'une confiance commune entre les acteurs ou bien sur des critères matériels ou logiciels (p.ex. *tamperproof hardware*, sites intermédiaires de confiance). Par exemple, dans le cas des sites marchands, le client peut établir un niveau de confiance en fonction de certifications ou de labels, qui dans le pire des cas, permet un recours juridique. La certification fait l'objet d'un dispositif réglementaire encadré par la loi (cf. le service Fimatex de bourse en ligne qui détient le certificat WebCert, délivré par l'AFAQ). Le label fait l'objet d'un contrôle par un auditeur indépendant, lui-même certifié (cf. Labelsite de la fédération des entreprises de vente à distance). S'il n'y a pas de connaissance sur la sécurité d'un site, le niveau minimal est utilisé.

4.1.2.2 Analyses

L'inférence de la qualité de mise en œuvre sur le critère de sécurité se définit de la même manière que pour le critère de performance. Toutefois, plutôt que de synthétiser une valeur, le processus vérifie le respect de la propriété de sécurité demandée (c.-à-d. confidentialité ou intégrité). Le domaine de l'interprétation sémantique est donc ici un booléen. Pour les deux instances de cette analyse, nous considérons que le site du client est de confiance maximale. En effet, le client est l'initiateur des interactions et ses données et traitements sont déjà présents et accessibles sur son site. L'application d'un traitement en local sur son site, ou encore d'un service localisé sur ce même site est donc considérée comme protégée.

La vérification de la propriété de confidentialité prend la forme d'une fonction de valuation sémantique $Conf_E$. Elle utilise l'environnement e, qui contient les niveaux de confidentialité des objets (sous forme de continuation pour les fonctions primitives), et les environnements e_c et e_s qui sont utilisés seulement en lecture (constantes globales). La fonction de valuation sémantique $Conf_E$ est décrite dans la Figure 4.3.

$Conf_E$ prend en arguments l'expression concrète, représentative d'une mise en œuvre de service complexe, et l'environnement e. En fonction des hypothèses des environnements, son résultat est un booléen qui détermine si oui ou non les données et traitements secrets du client du service complexe ne seront pas divulgués.

1. la construction **let** est traitée comme précédemment (cf. section 4.1.1.2). Si l'expression concrète est simplement une donnée D, elle respecte la confidentialité (pas de migrations de code ou de données). Le service complexe est confidentiel si toutes les interactions le sont ($Conf_A$ rend la valeur $true$).

2. chacune des interactions $A\ D$ est traitée par la fonction $Conf_A$. La fonction de valuation sémantique $Conf_A$ prend en argument l'environnement des fonctions, le site courant de

$$
\begin{array}{lcl}
Conf_E \ : \ Exp_c \to Env \to Bool & & \\
Conf_E[\![\textbf{let } (r_1,\ldots,r_n) = A\ D \textbf{ in } E]\!]\ e & = & Conf_A[\![A]\!]\ e\ client\ (Lub\ [\![A]\!])\ (\mathcal{D}_c[\![D]\!]\ e) \wedge \\
 & & Conf_E[\![E]\!] \\
 & & \quad (e[(r_1,\ldots,r_n) \leftarrow \mathcal{A}_c[\![A]\!]\ e\ (\mathcal{D}_c[\![D]\!]\ e)]) \\
Conf_E[\![D]\!]\ e & = & true \\
 & & \\
Conf_A \ : \ Exp_A \to Env \to Id_{site} \to Level \to Data_{Level} \to Bool & & \\
Conf_A[\![X]\!]\ e\ p\ c\ D & = & (\bigsqcup d \sqcup c) \sqsubseteq e_s\ p \wedge \\
 & & \underline{Cas}\ \ X\ \ \underline{dans} \\
 & & \overline{f}\ A \quad : e\ \overline{f}\ (Conf_A[\![A]\!]\ e\ p\ c)\ D \\
 & & go_{i,j}\ A\ : (\bigsqcup d \sqcup c) \sqsubseteq e_s\ \alpha_{i,j} \wedge \\
 & & \quad Conf_A[\![A]\!]\ e\ j\ c\ D \\
 & & end \quad : true \\
 & & \\
Lub \ : \ Exp_A \to Level & & \\
Lub\ [\![\overline{t}\ A]\!] & = & (e_c\ t) \sqcup Lub\ [\![A]\!] \\
Lub\ [\![\overline{s}\ A]\!] & = & Lub\ [\![A]\!] \\
Lub\ [\![go_{i,j}\ A]\!] & = & Lub\ [\![A]\!] \\
Lub\ [\![end]\!] & = & \bot
\end{array}
$$

FIG. 4.3 – *Vérification de la confidentialité*

l'agent (initialement le client), le niveau de confidentialité du code source de l'agent (la borne supérieure des niveaux des traitements) et bien sûr les niveaux des données arguments.

Vérifier la confidentialité d'une interaction par agent revient à s'assurer qu'à chaque étape de réduction, le niveau requis par l'agent (c.-à-d. de ses données courantes et de son code source) est inférieur ou égal au niveau proposé par le site sur lequel l'agent est en transit ($e_s\ p$). L'identificateur p est utilisé à défaut de s pour site, afin d'éviter toute confusion avec un identificateur de service primitif. Les données courantes d'un agent pouvant être un n-uplet, le symbole $\bigsqcup$ permet de définir le niveau de confidentialité du sac à dos de données. Il est défini comme suit :

$$
\begin{cases}
\bigsqcup(d_1, ..., d_n) = \bigsqcup(d_1) \sqcup ... \sqcup \bigsqcup(d_1) \\
\bigsqcup public = public \\
\bigsqcup secret = secret \\
\text{avec } public \sqcup secret = secret, secret \sqcup public = secret, a \sqcup a = a
\end{cases}
$$

Lors d'une application de fonction par un agent, les niveaux de confidentialité du résultat de la fonction sont calculés et utilisés par l'analyse du reste de la continuation. Pour chaque primitive de migration, le niveau de confidentialité requis par les données et le code doit être inférieur ou égal au niveau assuré par le canal de communication entre les sites i et j ($e_s\ \alpha_{i,j}$). La suite de la continuation est analysée, où le site courant de l'agent est mis à jour.

3. le niveau de confidentialité du code source est calculé à l'aide de la fonction *Lub*. Cette

fonction traite chaque traitements de la continuation de l'agent et en extrait le plus grand niveau. Le symbole $\bot$ dénote le plus petit élément du treillis (p.ex. public).

Il est possible de définir de la même manière une fonction Int_E, duale de $Conf_E$ (cf. Figure 4.4). L'environnement e contient alors les niveaux d'intégrité (c.-à-d. sûr et non sûr) des données et fonctions du service complexe. Dans ce cas, une fois qu'un traitement a été exécuté sur un site, il n'est pas nécessaire de vérifier son intégrité (garantie de non modification) sur la suite de l'itinéraire. Le niveau du code source de l'agent ($Lub\ [\![A]\!]$) est donc à réexaminer à chaque fois qu'un traitement de la continuation est appliqué. À chaque étape de l'étude d'une continuation, il faut s'assurer que le niveau d'intégrité du sac à dos courant de l'agent est inférieur ou égal à celui du site courant ($((\bigsqcup d \sqcup c) \sqsubseteq e_s\ p)$. Un site p est sûr s'il garantit l'exécution intègre des agents (données et code) et la non-modification des agents.

$$
\begin{array}{lcl}
Int_E\ :\ Exp_c \to Env \to Bool & & \\
Int_E[\![\textbf{let}\ (r_1,\ldots,r_n) = A\ D\ \textbf{in}\ E]\!]\ e & = & Int_A[\![A]\!]\ e\ client\ (Lub\ [\![A]\!])\ (\mathcal{D}_c[\![D]\!]\ e) \wedge \\
 & & Int_E[\![E]\!] \\
 & & \quad (e[(r_1,\ldots,r_n) \leftarrow \mathcal{A}_c[\![A]\!]\ e\ (\mathcal{D}_c[\![D]\!]\ e)]) \\
Int_E[\![D]\!]\ e & = & true \\
 & & \\
Int_A\ :\ Exp_A \to Env \to Id_{site} \to Level \to Data_{Level} \to Bool & & \\
Int_A[\![X]\!]\ e\ p\ c\ D & = & (\bigsqcup D \sqcup c) \sqsubseteq e_s\ p \wedge \\
 & & \underline{Cas}\ \ X\ \ \underline{dans} \\
 & & \overline{f}\ A \quad : e\ \overline{f}\ (Int_A[\![A]\!]\ e\ p\ (Lub\ [\![A]\!]))\ d \\
 & & go_{i,j}\ A\ :\ (\bigsqcup D \sqcup c) \sqsubseteq e_s\ \alpha_{i,j} \wedge \\
 & & \quad Int_A[\![A]\!]\ e\ j\ (Lub\ [\![A]\!])\ D \\
 & & end \quad : true
\end{array}
$$

FIG. 4.4 – *Vérification de l'intégrité*

4.1.2.3 Exemple

Les exemples de presse électronique ou de recherche de documents proposés précédemment (cf. chapitre 3) n'ont pas nécessairement un souci de sécurité au niveau du client (p.ex. récupération d'articles sportifs, publications scientifiques). Afin de donner l'intuition de la vérification de la qualité de mise en œuvre, nous étudions une mise en œuvre par un agent d'un service de cotations boursières dans un souci de confidentialité. Un client souhaite établir la valeur de son portefeuille d'actions en utilisant deux services.

1. un service *cotation* permet de récupérer la liste des cotations courantes de toutes les actions en bourse (à partir de la requête d_1).

2. un service bancaire *bank*, accessible à partir d'un identifiant et code d'accès client (contenus dans d_2), fournit la liste (et le nombre) des actions d'un client.

3. le client souhaite utiliser un traitement personnalisé valeur qui utilise une liste des cotations et une liste d'actions afin de fournir la valeur de son portefeuille à l'instant donné.

L'expression abstraite liée à ce service complexe est $\mathsf{valeur}(cotation\ d_1, bank\ d_2)$.

Les données confidentielles vont être le code d'accès du client (d_2), la liste de ses actions (résultat de *bank*) et la valeur de son portefeuille (résultat final). Les cotations sont publiques, alors que le service bancaire rend des données personnelles. Le code du traitement peut être public, vu qu'il se limite à un algorithme de correspondance et de sommation. Toutefois, son résultat est du niveau de son deuxième argument (c.-à-d. liste d'actions). Selon le critère de confidentialité, les niveaux associés aux fonctions peuvent être définis comme suit :

$$\begin{array}{l} cotation : \lambda x.public \\ bank : \lambda x.secret \\ \mathsf{valeur} : \lambda(x,y).y \end{array}$$

La réalisation du service complexe par un unique agent peut être spécifié par :

$$(go_{c,cot}; \overline{cotation}; go_{cot,b}; \overline{bank}; \overline{\mathsf{valeur}}; go_{b,c}; end)\ \ (d_1, d_2)$$

Afin que cette interaction $A\ D$ respecte la propriété, il est nécessaire que les sites (c, cot, b) et liens garantissent la confidentialité de l'agent en transit. Initialement, le niveau du code de l'agent ($Lub\ [\![A]\!]$) correspond à public, le code de valeur n'étant pas un souci de confidentialité. Les données initiales de l'agent correspondent à $(public, secret)$. L'analyse porte sur $Conf_A[\![A]\!]\ e\ client\ public\ (public, secret)$:

1. le sac à dos de l'agent, au départ du client (code et données), a un niveau *secret* (c.-à-d. $\bigsqcup(public, secret) \sqcup public$). Par hypothèse, ce sac à dos est toujours compatible avec le niveau du site du client ($\sqsubseteq\ e_s\ p$). Au $go_{c,cot}$, le niveau offert par la liaison entre les sites c et cot doit être de niveau secret. Le site cot doit également assurer la confidentialité de l'agent en transit ($d_2 \sqsubseteq\ e_s\ cot$). Ce service de cotation est le plus souvent public. L'application du service de cotation se fait à partir de la définition CPS. Le niveau du nouveau sac à dos reste à secret, vu que la donnée d_2 n'a pas été consommée. Le lien entre les sites cot et b doit donc être également de niveau secret.

2. le site b de la banque doit assurer le secret car il exécute un agent à données confidentielles. Il est vraisemblable que le site bancaire garantisse la confidentialité. Une fois le service *bank* appliqué, la donnée confidentielle d_2 est consommée. Toutefois, le résultat reste de niveau secret. Ce résultat est utilisé par le traitement valeur. Le traitement peut être lu par le site de la banque, vu qu'il n'est pas confidentiel.

3. après application du traitement valeur sur le site de la banque, le sac à dos de l'agent pour son retour chez le client ($go_{b,c}$) est toujours de niveau secret, le résultat de valeur étant du niveau de celui du résultat de *bank*.

Sur les trois liens du réseau vont donc circuler une donnée secrète, les niveaux correspondant (α) devront être maximaux pour garantir la propriété (c.-à-d. canaux protégés). L'analyse proposée donne *false* si un des sites ou liens viole la relation d'ordre partiel au regard du sac à dos de l'agent traité (qui est toujours à secret avec les hypothèses données). Dans ce cas, il est nécessaire d'interagir avec d'autres services fonctionnellement identiques, ou d'étudier la possibilité d'utiliser un traitement pour pallier à ce problème de confidentialité des données (p.ex. chiffrement à clé publique), ou encore de considérer un autre mécanisme d'interaction (p.ex. invocation distante, autre itinéraire).

4.1.3 Fiabilité

Les risques de défaillances des sites ou des canaux de communication peuvent considérablement altérer la fonctionnalité des services complexes (cf. section 2.3). Si un service primitif ne peut être rendu suite à une défaillance, c'est tout le service complexe qui n'est pas rendu. Autant que faire se peut, un service complexe fiable doit pouvoir continuer de fonctionner en présence de fautes ou de pannes, et ce sur toutes les interactions de sa mise en œuvre. Pour cela, il est possible de s'appuyer sur des services de persistance (c.-à-d. résistance aux pannes de sites), implantés chez les prestataires de services, ou sur des mécanismes de tolérance aux fautes logicielles. Les services de persistance s'appuient sur de la sauvegarde d'états. La tolérance aux fautes permet de masquer les défaillances logicielles en reposant sur de la réplication active, semi active ou encore passive au sein des serveurs (cf. section 2.3.2). Augmenter la fiabilité d'une interaction par invocation distante revient à sélectionner l'un de ces trois mécanismes de réplication. Pour améliorer la continuité du service complexe réalisé par un agent (suite d'actions), il est nécessaire de répliquer ses actions sur plusieurs sites distincts. Des mécanismes spécifiques construits sur les sites serveurs (invisibles aux agents et programmeurs d'agents) ont été proposés afin de supporter des défaillances (cf. section 2.3.3). La tolérance aux fautes pour agents impose un même mécanisme de réplication sur tout l'itinéraire de l'agent. En effet, il n'est pas raisonnable de chercher à combiner un mécanisme de réplication active (c.-à-d. Schneider, section 2.3.3.1) avec un mécanisme de réplication primaire/secondaire (p.ex. Straßer et Rothermel, section 2.3.3.2) sur l'itinéraire d'un agent.

4.1.3.1 Domaine

Un site, ou groupe de sites, ne peut pas garantir une fiabilité à 100% de ses exécutions (p.ex. incendies sur le groupe de sites). L'utilisation de mécanismes de persistance ou de tolérances aux fautes permet d'augmenter la fiabilité des exécutions. La fiabilité d'un site dépend donc des qualités matérielles et logicielles et du nombre de sites utilisés pour la réplication. Comme domaine de valeurs, il est possible de considérer des probabilités approximatives (estimation quantitative) de défaillances de sites. Toutefois, cette approche requière des connaissances précises sur la plate-forme, le déploiement et sur les différents mécanismes complexes de réplication existants (pour invocation distante et agents mobiles). Ces mécanismes sont eux-mêmes exposés aux défaillances et il est difficile d'y associer des valeurs. À défaut de définir des niveaux de fiabilité, nous préférons restreindre le domaine à un ensemble d'identificateurs de mécanismes. Nous considérons simplement si un site met en œuvre un mécanisme de tolérance aux fautes donné. L'élément minimal du domaine de valeurs est l'absence de mécanisme mis en œuvre (p.ex. null), et les maximaux, correspondant à la propriété *fiable*, sont les identificateurs des différents mécanismes identifiés (p.ex. Schneider, Straßer, Silva). L'opérateur de combinaison se restreint donc à l'équivalence.

Les environnements contiennent simplement les identificateurs liés aux mécanismes mis en œuvre sur les sites (c.-à-d. environnement e_s). Les environnement e et e_c ne sont pas utilisés pour la propriété de fiabilité. Dans notre cadre, les données et fonctions n'ont pas à être associés à des mécanismes attachés aux arguments ou résultats, vu leur indépendance envers ceux-ci. Seules les garanties d'exécution des sites (rendu de services et exécution des traitements) sont à traiter. Nous ne considérons pas les défaillances des connections entre les sites, celles-ci étant le plus souvent prises en charge par les couches systèmes ou indirectement par les mécanismes de réplication (c.-à-d. mécanisme de Schneider où un agent est diffusé à

tous les sites qui constituent le groupe de l'étape suivante).

4.1.3.2 Analyse

L'inférence de la qualité de mise en œuvre sur le critère de fiabilité est plus simple que les analyses précédentes. L'analyse permet de vérifier que la mise en œuvre d'un service complexe à un minimum de garanties de fiabilité. Un service complexe est considéré comme fiable si l'ensemble des interactions qui le mettent en œuvre respectent la propriété. Vérifier la fiabilité sur une interaction par invocation distante revient à s'assurer de la présence d'un mécanisme de tolérance aux fautes sur le site du prestataire. Dans les cas d'une interaction par agent, vérifier la fiabilité repose sur l'équivalence des mécanismes mis en œuvre sur les différents sites de l'itinéraire. La fonction de valuation sémantique $Fiab_E$ qui met en œuvre cette vérification est donnée dans la Figure 4.5.

$$
\begin{array}{lcl}
Fiab_E \ : \ Exp_c \rightarrow Bool & & \\
Fiab_E[\![\textbf{let } (r_1, \ldots, r_n) = A \ D \textbf{ in } E]\!] & = & Fiab_E[\![E]\!] \wedge Fiab_A[\![A]\!] \\
Fiab_E[\![D]\!] & = & true \\
 & & \\
Fiab_A \ : \ Exp_A \rightarrow Bool & & \\
Fiab_A[\![\overline{f} \ A]\!] & = & Fiab_A[\![A]\!] \\
Fiab_A[\![go_{i,j} \ A]\!] & = & Fiab_A[\![A]\!] \wedge \\
 & & \underline{si} \ i = client \ \underline{alors} \ e_s \ j \ \neq \ null \wedge \\
 & & \underline{si} \ i \neq client \ \wedge \ j \neq client \ \underline{alors} \ e_s \ i \ = \ e_s \ j \\
Fiab_A[\![end]\!] & = & true
\end{array}
$$

FIG. 4.5 – *Vérification de la fiabilité*

Pour la vérification, chaque interaction $A \ D$ est traitée par $Fiab_A$. Dès qu'une primitive de migration apparaît dans la continuation et qu'elle est initiée à partir du client, le site destination doit proposer un mécanisme voué à augmenter la fiabilité ($e_s \ j \ \neq \ null$). Si la migration n'implique pas le site client, les mécanismes mis en œuvre sur les deux sites en jeux doivent être équivalent ($e_s \ i \ = \ e_s \ j$). Nous ne déroulons pas ici d'exemple, vu la simplicité de l'analyse par rapport aux précédentes.

4.1.4 Commentaires

Nos analyses se définissent récursivement sur la grammaire du langage concret, qui est un modèle d'exécution laissant apparaître clairement les flots de données et de code. Elles permettent de comparer différentes mises en œuvres de services distribués complexes selon différentes qualités. À chaque objet de nos langages est associé une valeur ou une fonction signée sur un domaine ordonné. Cette valeur permet de proposer une mesure pour une propriété de qualité donnée. Ces spécifications sont indépendantes des mécanismes d'interaction utilisés. Le fait qu'un service soit accédé à distance par une invocation ou bien en local par un agent, ne changera pas ses spécifications. Cette approche nous permet d'étudier les flots d'informations (p.ex. données et traitements) induits par des interactions, au regard des propriétés traitées. Pour chacune des propriétés, un opérateur est utilisé pour calculer/vérifier les propriétés des

services complexes en fonction de leurs constituants. Pour les domaines de valeurs, les données sont définies statiquement, ce qui impose une certaine abstraction sur les spécifications non fonctionnelles. Il est possible de les associer à des valeurs symboliques. Toute propriété qui peut être déterminée par un domaine de valeur ordonné et associé à un opérateur de combinaison peut être analysée. En dehors de la qualité de service, des propriétés telles que la capacité de croissance ou le temps de développement sont difficilement quantifiables (cf. introduction). Elles sont de plus peu pertinentes d'un point de vue client de service complexe. Pour un concepteur de service complexe, bien qu'elles puissent servir de mécanisme de sélection ou de comparaison entre diverses interactions, ces propriétés paraissent difficilement intégrables dans le cadre choisi, très lié aux fonctionnalités et à leurs dépendances.

Pour les analyses de sécurité (resp. fiabilité), nous avons considéré que le site du client était de confiance maximale en terme de sécurité (resp. sûr de fonctionnement). Ainsi, placer tous les services primitifs à appliquer chez le client (que des interactions locales) est la mise en œuvre la plus performante (en terme de volume de données), sécurisé et sûre de fonctionnement. Cette solution triviale n'est évidement pas celle qui serait choisie par un concepteur de service complexe. Pour guider la conception, les analyses proposées peuvent s'adapter au cas où des éléments ou propriétés du réseau ne sont pas définis dans les environnements. Elles ouvrent ainsi la voie à des choix de placement ou de mécanismes à mettre en œuvre aptes à garantir au mieux les propriétés de qualité d'un service complexe. Un tel service complexe peut être spécifié uniquement au niveau abstrait pour sélectionner sa « meilleure » mise en œuvre.

4.2 Synthèses d'éléments de mise en œuvre

Nous avons présenté des analyses simples qui portent sur l'**inférence de la qualité de mise en œuvre** de services complexes. Elles supposent une vue complète du réseau, des propriétés de chacune des entités (p.ex. sites, liens, données, fonctions) et une description de la mise en œuvre (mécanismes d'interactions). Ces analyses peuvent prendre la forme d'une **vérification de la qualité de mise en œuvre** quand il est nécessaire de vérifier qu'une expression concrète respecte un seuil de performance, la confidentialité (resp. intégrité) des données et traitements du client ou encore qu'elle est relativement fiable. Si le service complexe représenté par une expression concrète ne vérifie pas la qualité de mise en œuvre, l'analyse indique au concepteur les objets qui violent la propriété pour la sécurité et la fiabilité.

Le concepteur dispose dans tous les cas de l'expression abstraite Exp_a (spécifiée à l'aide de notre langage abstrait) et de l'environnement e associé. Elle représente le service complexe à étudier ou construire. Les deux paramètres utilisés pour la vérification/inférence de la qualité de mise en œuvre sont :

1. l'expression concrète (associée aux environnement e et e_c),
2. l'environnement e_s qui correspond au réseau (placement des sites, propriétés des sites et liens).

Quels sont les autres problèmes que peut se poser le concepteur de services distribués complexes lorsqu'il dispose d'une expression abstraite Exp_a ? Ces problèmes dépendent des connaissances dont il dispose sur l'expression concrète Exp_c et sur le réseau (voir Figure 4.6). Nous considérons dans ce chapitre qu'un ou ces deux paramètres peuvent être partiellement définis et cherchons à synthétiser la meilleure mise en œuvre possible. Au service abstrait correspondent des expressions concrètes Exp_c (c.-à-d. les mises en œuvre possibles), qui sont un raffinement

de Exp_a. L'expression concrète peut par exemple ne pas être connue quand un architecte souhaite construire un service complexe. Ainsi, les paramètres peuvent être soit complètement définis (noté mode + dans la figure), soit à synthétiser s'il manque des informations (mode −). Nous disposons donc théoriquement de quatre (2^2) analyses possibles. Toutes ces analyses peuvent se fonder sur l'interprétation des expressions concrètes, envers les critères non fonctionnels. Toutefois, certaines de ces analyses sont trop coûteuses en pratique pour des services complexes non triviaux. C'est plus particulièrement le cas quand l'expression concrète n'est pas connue et que le réseau est à «trous» (incomplet) en terme de spécification des placements des services, des liens et des propriétés de qualité qui s'y attachent.

ANALYSE	CONNAISSANCES	
	Exp_c (interactions)	Réseau (localisation, propriétés)
inférence de la qualité de mise en œuvre	+	+
synthèse d'une description de mise en œuvre	-	+
synthèse d'architecture	+	-
synthèse d'architecture et d'une description de meo	-	-

FIG. 4.6 – *Différentes analyses possibles*

La première analyse de la Figure 4.6 a été traitée (cf. section 4.1). Nous présentons dans la suite les trois autres. Nous traitons la **synthèse d'une description de mise en œuvre** qui permet de produire une expression concrète « optimale » en fonction de la propriété de qualité. La **synthèse d'architecture**, quand le réseau est partiellement défini, est également étudiée. Elle permet de placer ou imposer des propriétés aux éléments du réseau à construire. La fusion des deux analyses précédentes est également discutée (**synthèse d'architecture et d'une description de mise en œuvre**).

4.2.1 Synthèse d'une description de mise en œuvre

Quand le réseau est totalement défini (mode +) et que l'expression concrète Exp_c n'est pas intégralement fournie (mode −), il est théoriquement possible de synthétiser la meilleure mise en œuvre du service complexe, au sens d'une ou des propriétés de qualité traitées. Dans le cas extrême, l'expression concrète est inexistante, le concepteur ne dispose que de l'expression abstraite. Nous pouvons également imaginer des cas où l'expression concrète est partiellement définie. Par exemple, certains mécanismes/itinéraires peuvent être fixés pour interagir avec des services primitifs. Il convient alors de s'intéresser aux services restants en cherchant à étendre les interactions déjà existantes ou à en proposer de nouvelles. Le concepteur peut également vouloir utiliser un seul agent pour réaliser intégralement le service complexe, il faut alors trouver l'itinéraire optimal correspondant, au sens d'une ou des propriétés de qualité traitées.

Pour la recherche d'une expression concrète, le critère d'optimalité s'appuie sur la relation d'ordre qui lie les valeurs du domaine considéré. Pour la performance, nous chercherons ainsi à fournir l'expression concrète qui génère le moins de trafic, pour la sécurité une expression qui assure la confidentialité (ou l'intégrité) des données et traitements, pour la fiabilité une expression qui utilise des sites avec mécanismes liés à la sûreté de fonctionnement. Dans le cas extrême et pour une propriété donnée, la complexité de la synthèse d'une expression concrète «optimale» est directement dépendante de celle de l'expression abstraite. Nous avons vu (cf.

section 3.4.1, page 62) qu'il est possible de générer toutes les expressions concrètes correspondantes à une expression abstraite donnée. Selon cette approche, il suffit ensuite d'inférer la qualité de mise en œuvre de toutes ces expressions, pour en sélectionner la meilleure, au sens de la relation d'ordre. Toutefois, pour un service complexe non trivial, il existe un nombre important d'expressions concrètes qui en sont un raffinement. Sans informations sur l'expression concrète, théoriquement et au pire cas, à une expression abstraite constituée exclusivement d'un n-uplet de premier niveau, correspondent de l'ordre de $2^{n-1}n!$ mises en œuvre possibles, où n est le nombre de fonctions présentes. Les types d'interactions (p.ex. invocation distante, agent) et les itinéraires influent directement sur les propriétés de qualité. Par exemple, en reprenant le service de cotations boursières (cf. section 4.1.2.3), à l'expression abstraite $\mathsf{valeur}(cotation\ d_1, bank\ d_2)$ vont correspondre deux expressions concrètes qui utilisent un unique agent :

$$(go_{c,cot}; \overline{cotation}; go_{cot,b}; \overline{bank}; \overline{\mathsf{valeur}}; go_{b,c}; end) \quad (d_1, d_2)$$
$$(go_{c,b}; \overline{bank'}; go_{b,cot}; \overline{cotation'}; \overline{\mathsf{valeur}}; go_{cot,c}; end) \quad (d_1, d_2)$$

Pour la performance, la taille du code de valeur et les données d_1 et d_2 ne transitent pas par les mêmes canaux, tout comme le résultat final (c.-à-d. $go_{b,c}$ ou $go_{cot,c}$). Sur ces deux expressions, les valeurs résultantes de l'inférence de la qualité sont donc vraisemblablement différentes. Il en est de même pour la sécurité où la donnée d_2 est de niveau secret. Pour le premier agent, elle transite sur le canal (c, cot), sur le site du service de cotation et sur le canal (cot, b), alors que pour le deuxième, elle transite seulement sur le canal (c, b). Pour la fiabilité, les sites b et cot doivent proposer un même mécanisme voué à améliorer la fiabilité des exécutions d'agents.

En pratique, nous verrons en section 5.3.3.1 une mise en œuvre possible de cette analyse qui partage des calculs afin de déterminer la meilleure expression concrète à partir d'une abstraite. Au vu de la complexité théorique, il est quand même possible de traiter des services complexes composés de plusieurs dizaines de fonctions. En effet, en analysant au fur et à mesure une expression abstraite, il devient possible de partager des interactions et de ne développer que celles qui sont susceptibles d'être optimales pour un même état accessible. De plus, afin de déterminer une expression concrète « optimale », il est judicieux de traiter l'ensemble des critères non fonctionnels en même temps. Ainsi, si une interaction avec un service ne respecte pas les critères de sécurité ou de fiabilité (booléen), il n'est pas nécessaire de développer cette interaction ou les interactions suivantes pour construire une mise œuvre du service complexe. Le critère de performance est plus délicat à utiliser car le problème ne peut se décomposer sans perdre en précision ; il est intrinsèquement global (ce point sera étudié plus amplement dans le dernier chapitre). Quand un début d'interaction est moins coûteux qu'un autre, l'interaction finale ne conduit pas toujours à un meilleur coût global.

L'analyse de synthèse d'une description de mise en œuvre permet de traiter, lors de la conception d'un service complexe, les contraintes de qualité au plus tôt dans l'ingénierie en combinant plusieurs mécanismes d'interaction. Elle permet de répondre à la question posée dans l'introduction de ce chapitre, à savoir, quelle est la meilleure combinaison de mécanismes/itinéraires (p.ex. invocation distante, migration d'agent, etc.) à mettre en place pour couvrir les besoins en qualité.

4.2.2 Synthèse d'architecture

L'utilisateur des analyses précédentes est un client désireux de se construire un service complexe sur un réseau fixé ou bien un architecte/concepteur agrégeant des services existants.

Toutefois, le concepteur peut chercher à construire des services distribués complexes composés de services primitifs qu'il a construits mais qui ne sont pas encore installés. Nous considérons dans un premier temps le cas où les mécanismes d'interactions sont fixés par l'expression concrète qui est fournie. Plus précisément, dans l'expression concrète, le découpage en interactions et les parcours d'agents sont définis ; seuls les sites correspondants aux services sont éventuellement indéfinis (c.-à-d. les identificateurs des go_{id_1,id_2}). Le placement des services peut être à la charge du concepteur. Les garanties non fonctionnelles nécessaires sur ces sites peuvent imposer des placements ou l'utilisation de mécanismes de qualité sur les serveurs associés. Dans ce cas, l'environnement e_s associé au réseau n'est pas entièrement défini. Pour la performance, certaines bandes passantes, si considérées, ne sont pas définies. Pour la sécurité, les niveaux des sites et liens peuvent ne pas être fixés. Pour la fiabilité, tous les identificateurs de mécanismes associés aux sites ne sont peut être pas donnés.

L'analyse qui porte sur la synthèse d'architecture permet d'assister le concepteur pour proposer une spécification du réseau où tous les éléments sont décrits dans l'environnement. Cette spécification servira à développer les parties de l'environnement d'exécution (p.ex. configuration d'un site, installation de nouveaux composants pour garantir les exigences de l'application, spécialisation du système de communication). Celle-ci doit respecter la (les) propriété(s) de qualité traitée(s) pour le service complexe considéré. La spécification du réseau produite doit correspondre, au sens de la relation d'ordre de la propriété considérée, à l'expression concrète minimale. Pour la performance, celle qui génère le moins de trafics, pour la sécurité, une qui respecte la confidentialité et l'intégrité des données et traitements clients et enfin pour la fiabilité, celle qui utilise des sites considérés comme fiable. L'analyse se base sur l'inférence de la qualité de mise en œuvre de l'expression concrète. Elle permet, autant que faire se peut, de donner une instance à tous les éléments sous-spécifiés. Lors de l'inférence de la qualité d'une interaction, il est possible de déterminer le placement optimal de services primitifs non localisés ou de retrouver les propriétés nécessaires à un site ou un lien pour garantir/optimiser la propriété de qualité à la fin de l'interaction. Pour la synthèse d'architecture, nous distinguons ainsi deux cas suivant les informations dont dispose le concepteur sur les éléments implantés : (i) le placement des services (ii) les propriétés que doivent garantir les sites et les liens.

4.2.2.1 Placement de services

Nous considérons ici que l'expression concrète est donnée. Certains des identificateurs de sites n'y sont pas fixés. Dans la description du réseau, à tous les sites et toutes les liaisons sont associées les propriétés de qualité (p.ex. p, α). La Figure 4.7 propose graphiquement une recherche de placement d'un service primitif, basée sur l'exemple 1 du chapitre précédent (cf. section 3.2.3, page 51).

Elle laisse apparaître les différents flots de données et traitements, où nous supposons, pour simplifier, que les traitements sont « perdus » après application, c.-à-d. qu'ils n'appartiennent plus au code de l'agent (critère propre à la plate-forme de mise en œuvre). Un concepteur souhaite proposer le service complexe de presse à ses clients, défini par :

$$\mathsf{fusion}\,(\mathsf{select}\,(\mathit{journal}\ d_1), \mathit{trad}(\mathit{actu}\ d_2, d_3))$$

Il impose une interaction unique par agent avec les différents services afin de réaliser les traitements select et fusion à distance, par souci de performance. Il a construit lui-même le service *actu* et cherche à lui donner une localisation sur le réseau. L'expression concrète à

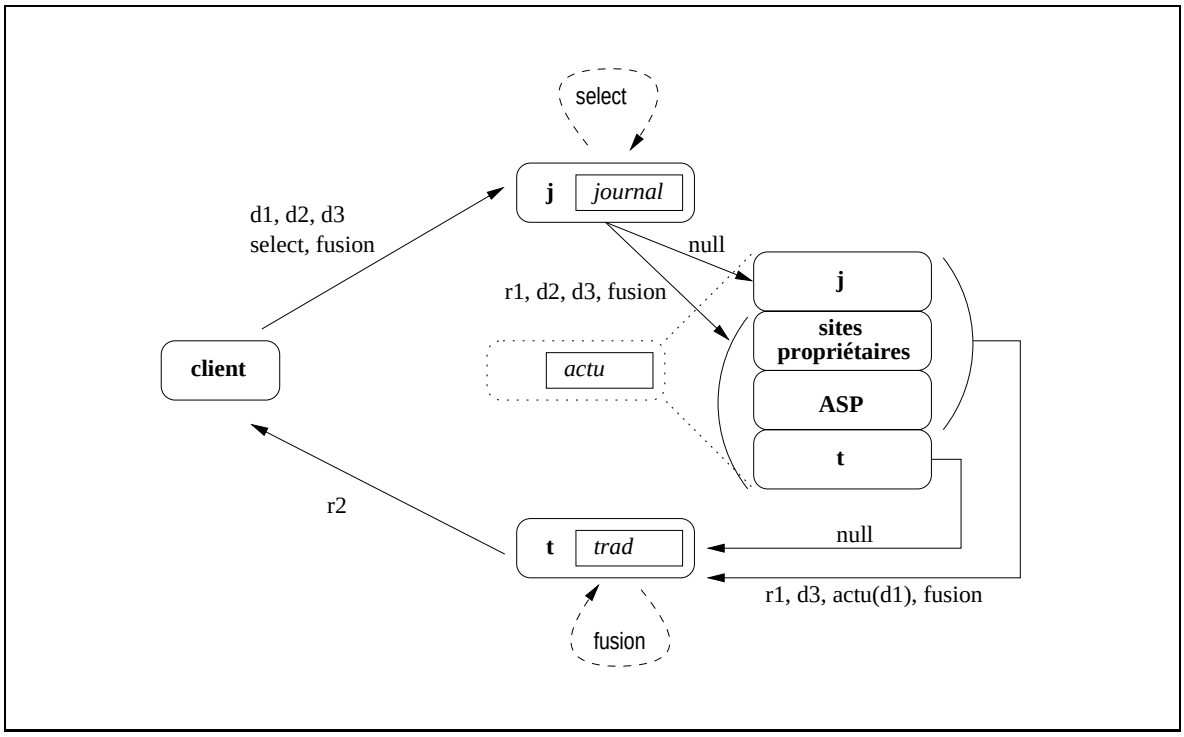

FIG. 4.7 – *Exemple de placement d'un service primitif*

analyser prend donc la forme suivante :

$$(go_{c,j}; \overline{journal}; \overline{\mathsf{select}}; go_{j,id_x}; \overline{actu}; go_{id_x,t}; \overline{trad}; \overline{\mathsf{fusion}}; go_{t,c}; end) \quad (d_1, d_2, d_3)$$

Pour placer son service *actu* (id_x), il doit considérer les sites déjà utilisés (c.-à-d. j et t), s'il a la capacité d'y installer son service primitif, les serveurs de son réseau (c.-à-d. sites propriétaires) ou des serveurs distants que son organisation utilise (c.-à-d. serveurs locatifs ou *Application Service Provider*, cf. section 2.5). Sur l'exemple et pour la performance, sans considérer les bandes passantes, la localisation du service *actu* sur le site t est vraisemblablement optimale (c.-à-d. si $d_2 < actu(d_1)$).

L'analyse s'appuie sur l'inférence de la qualité de mise en œuvre d'un service complexe concret, par définition récursive (induction structurelle) sur la grammaire du langage concret pour chaque interaction. Pour une propriété de qualité donnée (p.ex. performance), à chaque étude d'un go_{id_1,id_2}, il faut vérifier la présence dans les environnements des identificateurs. Initialement, la première primitive *go* de l'expression est initiée à partir du client. Ainsi, si le site id_2 n'a pas de correspondance dans les environnements, cela signifie que le ou les services primitifs suivants (c.-à-d. jusque la prochaine primitive de migration) dans la continuation ne sont pas localisés sur le réseau. Il faut alors chercher à placer ce(s) service(s) primitif(s) sur un site qui peut supporter les définitions non fonctionnelles, qui propose un mécanisme garantissant (resp. optimisant) la confidentialité/intégrité (resp. fiabilité) sur l'itinéraire et qui minimise le coût de transfert ($\alpha_{i,j}$ de la performance). La recherche de tels sites se fait à partir de la liste des sites disponibles (p.ex. définie dans l'environnement e_s). Dans le cas des interactions par agent, au plus simple, l'analyse cherche à placer le service primitif en fonction de l'itinéraire de l'agent, à savoir sur le site courant ou sur le premier site successeur.

Chaque point de choix doit être conservé dans l'éventualité où d'autres services ne seraient pas localisés dans le reste de la continuation.

Au niveau de la complexité de l'analyse, il faut considérer les différentes alternatives en fonction de la (les) propriété(s), pour une interaction donnée, et ce à chaque fois qu'un choix de placement apparaît dans l'expression concrète. La complexité dépend donc du nombre d'éléments non définis et des différentes alternatives envisageables. Dans le cas extrême, si aucun service primitif n'est placé il convient de les localiser sur le site client. Si c'est un architecte qui propose un service complexe comme interface à ses clients potentiels, il peut placer ses services sur le serveur qui sert de point d'accès aux clients pour son service complexe. Dans les cas intermédiaires, comme le graphe est par hypothèse fini, le nombre de choix est borné. Si plusieurs propriétés de qualité sont prises en compte, dès qu'une proposition garantit les propriétés de sécurité et de fiabilité pour une interaction, l'analyse sur ces critères peut se terminer. Le cas de la performance impose de comparer les différentes propositions d'expressions concrètes où les id_x ont été définis, pour déterminer les parcours optimaux.

4.2.2.2 Propriétés des sites et des liaisons

Dans le deuxième cas de la synthèse d'architecture, nous considérons que les services sont tous placés sur le réseau. L'expression concrète est donc complètement spécifiée, elle ne comporte que des identificateurs de sites définis. Cependant, dans l'environnement e_s, des sites ou des liens n'ont pas de valeur associée pour la (les) propriété(s) de qualité. Des sites ou des liaisons entre sites n'ont donc pas encore été déployés sur le réseau. Des mécanismes particuliers pourront leur être associés afin de garantir/optimiser une (des) propriété(s) de qualité. Pour la sécurité et la fiabilité, ces mécanismes sont liés à des squelettes d'architectures (enrichissement de la plate-forme ou spécialisation du système de communication) à intégrer au déploiement (cf. chapitre suivant), qui correspondent à une valeur dans un domaine de qualité donné.

À partir d'une expression concrète, il faut donc rechercher à dimensionner des sites ou des liaisons au regard des propriétés de performance (α des liens), de sécurité (niveaux des sites et liens) ou de fiabilité (identificateurs pour sites). Pour cela, l'analyse s'appuie sur l'inférence de la qualité de mise en œuvre d'un service complexe concret. Comme exemple, nous pouvons traiter celui de la section précédente (cf. Figure 4.7), où le service *actu* est localisé sur un site p propriétaire, différents des sites j et t. Envers le critère de confidentialité, nous supposons que les sites j et t sont déjà déployés et ont respectivement un niveau maximal et minimal. Le site du journal respecte donc la confidentialité des agents en transit, tandis que le site de traduction ne garantit pas la non-divulgation des informations (traitements et données). Les liaisons entre le client et les sites j et t sont supposées protégées. L'analyse va permettre de déterminer quel est le niveau minimal de confidentialité à imposer sur le site p et sur les liens (j, p) et (p, t), afin de garantir la confidentialité du service complexe. Au niveau du site p, l'étude va donc porter sur le sac à dos entrant (c.-à-d. résultat r_1, données d_2 et d_3, traitement fusion) et le sac à dos sortant (c.-à-d. résultats r_1 et $actu(d_1)$, donnée d_3, traitement fusion).

L'analyse suit l'interprétation sémantique de l'inférence de la qualité de mise en œuvre. Si pour un identificateur de service d'une continuation, l'environnement e_s n'est pas défini suivant la propriété traitée, le niveau minimal à proposer dans l'environnement est déterminé :

- pour la sécurité, le niveau maximal des éléments du sac à dos est sélectionné afin de respecter la relation $(\bigsqcup d \sqcup c) \sqsubseteq e_s\ p$. Si la sécurité est à garantir et que le sac à dos a le niveau (public,non-sûr), le site pourra être déployé sans mécanismes de sécurité particuliers.

- pour la fiabilité, si l'interaction est une invocation distante, un identificateur correspondant à un mécanisme de tolérance aux fautes est sélectionné. Si l'interaction est un agent mobile, l'identificateur est sélectionné en fonction de ceux présents dans les autres sites de services primitifs de la sous-expression.

Si, pour un identificateur de lien (canal de communication $\alpha_{i,j}$), l'environnement e_s n'est pas défini suivant la propriété traitée (connexion non déployée), le niveau minimal à proposer dans l'environnement est déterminé :

- pour la performance, la bande passante maximale est à sélectionner (inverse de la bande passante $\alpha_{i,j}$ minimal). La sélection se fait à partir d'une base de liaisons (fournie en plus des environnements). Cette base contient une liste de connecteurs physiques avec les bandes passantes associées.

- pour la sécurité, il faut proposer le niveau maximal des éléments du sac à dos afin de respecter la relation $(\bigsqcup d \sqcup c) \sqsubseteq e_s\ \alpha_{i,j}$. Ce niveau pourra imposer un chiffrement sur la liaison au déploiement.

Pour une propriété, la complexité de cette analyse correspond à celle de la synthèse de la qualité de mise en œuvre. Pour un élément non défini dans une expression correspond seulement une valeur. Pour la performance, une bande passante maximale ou une valeur de sécurité minimale est définie pour un lien. Si un lien (resp. site) apparaît plusieurs fois dans l'expression, la valeur maximale des différentes propositions de l'analyse prime.

4.2.2.3 Synthèse d'architecture et d'une description de mise en œuvre

Cette analyse doit à la fois compléter un réseau partiellement défini (localisation et/ou propriété(s)) et synthétiser une expression concrète qui garantit/optimise une (des) propriété(s) de qualité. Le concepteur dispose dans ce cas simplement de l'expression abstraite, auquel est éventuellement associée une ébauche d'expression concrète. Par principe, il est possible de synthétiser une expression concrète minimale, au sens d'une relation d'ordre, avec son réseau associé. Toutefois, le nombre d'alternatives à considérer/comparer est considérable dans un cas général. Si l'expression concrète n'est pas fournie, dans un premier temps, l'analyse a en charge de produire toutes les expressions concrètes qui se rapportent au service complexe (c.-à-d. synthèse exhaustive des descriptions de mise en œuvre). Il est possible de générer une expression concrète sans connaître le placement de certains services en apposant des primitives de migrations qui contiennent des identificateurs de sites indéfinis. Ces expressions concrètes ne sont pas directement comparables à l'aide de l'inférence de la qualité de mise en œuvre. En effet, le réseau associé à une expression concrète est encore incomplet. Chacune de ces expressions concrètes doit donc être traitée par la synthèse d'architecture afin de déterminer le réseau « optimal » qui lui correspond, tant en terme de placement des services que des propriétés associées aux éléments.

Au même titre que pour les analyses précédentes, pour la sécurité et la fiabilité, le domaine de valeur étant fini, il est possible d'atteindre une valeur minimale pour des petits services complexes sur un réseau où la plupart des éléments sont placés. Dès qu'une expression concrète et son réseau complet garantit la sécurité et/ou optimise la fiabilité, il n'est pas nécessaire de s'intéresser aux autres alternatives d'expressions concrètes. Le cas de la performance est nettement plus exhaustif et inaccessible dans un cadre général. La complexité correspond dans ce cas à celle de l'étude de tous les raffinements possibles du service complexe (c.-à-d. synthèse

d'une description de mise en œuvre), auxquels vont correspondre des alternatives de placement de sites. L'écriture d'un algorithme réaliste qui traite de manière combinée les deux analyses n'est pas trivial. Les techniques de partage proposées dans l'analyse de synthèse d'une description de mise en œuvre se trouvent réduites à cause du réseau incomplet et donc plus difficilement applicables. La synthèse d'architecture et d'une description de mise en œuvre est donc vraisemblablement peu viable hormis dans les cas où le réseau est presque complet (principalement en terme de placement) et la propriété de performance n'est pas prise en compte.

4.2.3 Commentaires

Certaines des analyses présentées ont une complexité qui n'est pas négligeable, particulièrement quand la propriété de performance est prise en compte (c.-à-d. synthèse d'une description de mise en œuvre et synthèse d'architecture/description de mise en œuvre). Des techniques particulières (p.ex. partage, heuristiques) peuvent permettre de les rendre effectives pour des services complexes de taille raisonnable en nombre de fonctions. Dans un cadre général, le concepteur peut participer à différents choix afin de guider l'analyse. Par exemple, pour les deux cas correspondants à la synthèse d'architecture (c.-à-d. placement, propriétés), des critères supplémentaires sur le réseau peuvent être fournis par le concepteur. Le concepteur peut limiter le placement des services à des sites déjà identifiés dans l'expression concrète afin de limiter les migrations d'agents. Il peut également contraindre l'utilisation de mécanismes voué à garantir une propriété de qualité ou encore se satisfaire d'une valeur pour une propriété (p.ex. mise en œuvre d'un service complexe qui ne génère pas un volume de trafic supérieur à un seuil plutôt que recherche de la minimale).

L'analyse la plus complexe est la synthèse d'architecture/description de mise en œuvre dans le cas de la performance. Elle requiert de produire toutes les expressions concrètes d'un service complexe. Pour chacune d'elle, la synthèse d'architecture doit traiter du placement des services, et pour chaque proposition, de la recherche des propriétés des sites et liens.

4.3 Synthèse de traitements

Grâce au mécanisme de migration de code, les agents mobiles ont la capacité d'exécuter des traitements sur les sites de services. Les agents permettent d'étendre les fonctionnalités d'un service primitif, pour le compte d'un client. Par hypothèse pour cette analyse, les prestataires de services permettent l'exécution de traitements (autres que les invocations de services et les migrations) d'agents provenant de l'extérieur. Nous pouvons guider la conception d'un agent en considérant l'introduction de traitements spécifiques, de nature non fonctionnelle, dans le service complexe qu'il représente et ce afin de chercher à garantir « au mieux » une propriété. Nous étudions dans cette section comment choisir les traitements appropriés afin d'étendre un service complexe donné pour les trois propriétés de qualité.

4.3.1 Extension d'expressions

Une expression abstraite ou concrète représente un service complexe. En général, ajouter un traitement dans une telle expression modifie la sémantique du service complexe. Toutefois, si un traitement t a un traitement inverse t^{-1}, il est possible de les composer au service complexe tout en conservant la sémantique. Par exemple, un traitement compress, correspondant à une

compression sans perte, possède un inverse (uncompress). On suppose ici que les traitements particuliers, voués à l'extension de services sont proposés au sein d'une bibliothèque ou bien directement fournis par un client. Ils doivent posséder un traitement inverse. Un traitement est déterminé par ses signatures fonctionnelles et non fonctionnelles (environnement e). Le code de ce traitement doit être à la disposition du client ou de l'architecte qui crée le service complexe afin de pouvoir créer la version exécutable de l'agent étendu.

Pour la synthèse de traitements, l'expression concrète est supposée connue, tout autant que le réseau (localisations et propriétés), bien que des versions adaptées aux synthèses de descriptions de mise en œuvre et d'architecture puissent être envisagées. Le concepteur utilise cette analyse, soit pour chercher à étendre une expression concrète qui ne garantirait pas la propriété de qualité, soit pour « accroître » le niveau d'une propriété pour un service complexe. Chercher à étendre un service complexe revient à déterminer les sites où des traitements prédéfinis peuvent s'insérer. Des règles de typage (correspondance entre le type des arguments et le type des résultats) permettent de déterminer, pour chaque fonction primitive f de l'expression abstraite, les traitements éligibles. Si un traitement, et son inverse, sont applicables au sens de la composition des interfaces, il est nécessaire de vérifier si l'extension fonctionnelle respecte les contraintes non fonctionnelles (gain) au niveau de l'expression concrète étendue correspondante. La vérification s'appuie sur l'inférence de la qualité de mise en œuvre en étudiant les différents choix de placement des traitements dans l'expression concrète (p.ex. traitement à distance, en local chez le client). Les différentes expressions concrètes étendues sont comparées entre elles et par rapport à l'expression initiale.

4.3.2 Exemples

4.3.2.1 Performance

Pour le critère de performance, l'utilisation de la compression de données peut permettre de réduire la taille des données devant transiter sur un canal de communication. Ainsi, il est possible d'appliquer sur des données (clientes ou résultats d'un service) un traitement de type compression sans pertes (p.ex. codage de Huffman), voir dans des cas particuliers une compression avec pertes (p.ex. à l'extrême JPEG, MPEG) qui, à strictement parler, ne possède pas d'inverse. La compression porte sur le site où la donnée est crée, à savoir sur le site du client pour les données initiales, ou sur un site de l'itinéraire pour les résultats de service (ou éventuellement de traitement). L'analyse se fait à partir d'une interaction concrète, p.ex. :

$$\textbf{let } r_1 = (go_{c,1}; \overline{s_1}; go_{1,2}; \overline{s_2}; go_{2,c}; end)\ d \textbf{ in}$$

Si la compression de données a lieu chez le client, l'agent correspondant à l'interaction devra contenir dans son code le traitement inverse de décompression. Ce traitement inverse décode les données sur un autre site de l'itinéraire (cf. Exp_{c1+}). Si la compression est réalisée sur un résultat de service qui est réutilisé par les autres fonctions de l'itinéraire, la décompression a lieu sur l'itinéraire (cf. Exp_{c2+}). Si la compression a été réalisée sur un résultat de service qui n'est pas réutilisé sur l'itinéraire, la décompression a lieu chez le client (cf. Exp_{c3+}). Dans ce cas, la décompression se fait à l'aide d'un traitement local, c.-à-d. à l'aide d'une autre interaction. Ainsi, l'agent étendu n'a pas à intégrer le traitement de décompression dans son code, qui transite à travers plusieurs sites.

(Exp_{c1+}) **let** $r'_1 = (\overline{t}; end)\ d$ **in**
let $r_1 = (go_{c,1}; \overline{s_1}; go_{1,2}; \overline{t^{-1}}; \overline{s_2}; go_{2,c}; end)\ r'_1$ **in**

(Exp_{c2+}) **let** $r_1 = (go_{c,1}; \overline{s_1}; \overline{t}; go_{1,2}; \overline{t^{-1}}; \overline{s_2}; go_{2,c}; end)\ d$ **in**

(Exp_{c3+}) **let** $r'_1 = (go_{c,1}; \overline{s_1}; go_{1,2}; \overline{s_2}; \overline{t}; go_{2,c}; end)\ d$ **in**
let $r_1 = (\overline{t^{-1}}; end)\ r'_1$ **in**

Plus concrètement, supposons un service de rendu de fichiers PostScript ps, qui fournit le fichier correspondant à une requête d. Le service Exp_{a1} spécifié ci-dessous peut être étendu en un service Exp_{a1+} en comprimant les résultats du service de rendu de documents.

(Exp_{a1}) $ps\ d$
(Exp_{a1+}) uncompress(compress($ps\ d$))

Une vérification de la qualité de mise en œuvre permet de déterminer si le coût d'envoi du traitement avec un agent reste favorable au regard des données compressées. Ainsi, l'expression concrète étendue qui minimise le volume de données garde le fichier compressé jusqu'au retour chez le client, c.-à-d. :

let $r_1 = (go_{c,r}; \overline{ps}; \overline{\mathsf{compress}}; go_{r,c}; end)\ \ d$ **in**
let $r_2 = (\overline{\mathsf{uncompress}}; end)\ \ r_1$ **in** r_2

4.3.2.2 Sécurité

Dans le cadre de la sécurité, nous pouvons utiliser des techniques de chiffrement sous la forme de traitements. Par exemple, une interaction par agent peut conduire à récupérer des données confidentielles sur un site. Si la suite de l'itinéraire de l'agent contient un site malhonnête, il est raisonnable de chercher à chiffrer ces données avant de migrer vers le site problématique. Le code de l'agent peut alors être étendu avec un traitement cryptographique (algorithme de chiffrement à clé publique), appliqué après le service qui aura fourni les données sensibles. Les données seront simplement déchiffrées sur le site du client, au retour de l'agent. En chiffrant avec sa clé publique, l'agent s'assure qu'un site malveillant ne pourra déchiffrer ses données lors du passage de l'agent. Pour garantir l'approche, il convient de s'assurer que le traitement cryptographique ne peut pas être utilisé par le site malhonnête pour retrouver les données en clair. Les données chiffrées ne doivent donc pas être à déchiffrer sur le reste de l'itinéraire (cas du résultat point d'entrée d'un autre service). En effet, dans ce cas, le traitement inverse ferait partie intégrante de l'agent et les sites de l'itinéraire ont la visibilité sur ce code.

Un agent a la possibilité de convertir des données résultats pour les rendre non-visibles et/ou non-modifiables. La protection du code de l'agent (cf. section 2.2.2, page 32) est difficile à mettre en œuvre par un traitement de l'agent. Les approches de Hohl [Hoh98b] ou de Sander et Tschudin [ST98] cherchent à protéger l'agent contre l'analyse d'un site attaquant. Elles imposent une conversion du code de l'agent (p.ex. brouillage ou chiffrement). La conversion doit s'effectuer sur le site client. Pour le chiffrement d'agents de Sander et Tschudin, la déchiffrement (de-conversion) a lieu au retour de l'agent, sur le site du client. Un tel traitement ne doit pas se contenter de modifier les données, il doit modifier le code de l'agent, donc le reste de la continuation (représentative d'un agent).

4.3.2.3 Fiabilité

Les différents travaux qui portent sur la fiabilité des agents mobiles (cf. section 2.3.3, page 35) reposent sur la réplication des services. Au sens large, les agents mobiles sont susceptibles d'améliorer la fiabilité des interactions de manière autonome. Un agent qui arrive sur un site peu fiable peut quitter ce site avant l'arrivée d'une défaillance/panne, s'il peut l'anticiper. Ces approches restent difficilement intégrables dans notre cadre fonctionnel. Par ailleurs, il est possible de s'intéresser au clonage autonome des agents. Si un agent a la capacité de se cloner sur un site, afin d'atteindre plusieurs services avec une même interface, il peut mieux tolérer des défaillances de certains serveurs. Dans une telle approche, il est nécessaire de considérer le problème de plusieurs agents en migration qui cherchent à réaliser un même service complexe. Certains agents doivent s'arrêter afin de ne pas dupliquer les services rendus. Des travaux plus en avant sont ici à étudier afin de traiter les mécanismes de communication induits pour qu'un seul agent réalise effectivement l'interaction. Rendre un agent mobile tolérant aux fautes de façon autonome (c.-à-d. sans modifier les sites de son itinéraire), seulement par l'intermédiaire de traitements dans une expression concrète, n'est donc vraisemblablement pas possible avec le langage proposé (p.ex. pas de clonage).

4.4 Vers un langage de spécification plus complet

Le langage abstrait est relativement simple. La principale motivation de cette simplicité est de permettre des analyses viables qui gardent un impact au niveau des étapes de construction d'un service distribué complexe (particulièrement pour la synthèse d'une description de mise en œuvre). Bien que ce langage nous permette de spécifier de nombreux services distribués complexes, il peut arriver qu'il soit nécessaire d'utiliser d'autres constructions pour spécifier des services complexes plus fins. Nous proposons ici diverses extensions au langage abstrait telles que le partage de résultats, l'ordre supérieur, les contraintes globales et les conditions de transition. Nous discutons des impacts de ces extensions sur la synthèse d'une description de mise en œuvre. Ces extensions sont utilisées au niveau spécification (langage abstrait) plutôt qu'au niveau description des mises en œuvre (langage concret). Elles s'inspirent, pour les contraintes globales et les conditions de transition, de travaux proposés dans la communauté de recherche sur le *workflow* (cf. section 3.5.2). Les langages de *script workflow* permettent de décrire des interactions et coordinations entre activités afin de contrôler les exécutions concurrentes de ces activités.

4.4.1 Partage de résultats

Le langage abstrait ne permet pas de spécifier l'utilisation de résultats de fonctions comme entrées de plusieurs autres services ou traitements. Par exemple, dans une expression abstraite de la forme $f(f_1\ d, E_2)$, au niveau de la grammaire, il n'est pas possible de récupérer des résultats de f_1, pour les utiliser dans E_2, tout en les gardant comme paramètres de f. Il faut soit dupliquer $(f_1\ d)$ dans l'expression pour la composer avec E_2 (modification de la sémantique du service global), soit utiliser un traitement particulier *duplicate* dans le langage. Plutôt que de modifier la sémantique des services complexes, nous choisissons d'introduire la construction **let** dans le langage abstrait :

$$E ::= \textbf{let}\ (r_1, ..., r_n) = E\ \textbf{in}\ E \mid r$$

Les résultats d'une fonction ou d'un n-uplet de sous-expressions peuvent être nommés $(r_1, ..., r_n)$ et réutilisés. Un résultat r_i peut ainsi servir d'entrée pour plusieurs services indépendants. Nous ne permettons pas la définition récursive de services complexes. L'interprétation sémantique de la construction **let** est standard (voir Figure 4.8).

$$
\begin{array}{lcl}
\mathcal{E}_a \ : \ Exp_a \ \rightarrow \ Env \ \rightarrow \ Data & & \\
\mathcal{E}_a[\![\textbf{let } (r_1, ..., r_n) = \ E_1 \textbf{ in } E_2]\!] \ e & = & (\lambda(r_1, \ldots, r_n).\mathcal{E}_a[\![E_2]\!] \ e)(\mathcal{E}_a[\![E_1]\!] \ e) \\
\mathcal{E}_a[\![r]\!] \ e & = & r
\end{array}
$$

FIG. 4.8 – *Interprétation sémantique du partage de résultats*

L'exemple 3 suivant présente un service complexe qui utilise ce partage de résultats.

Exemple 3

Dans [Rou02], nous nous sommes intéressés à un service de recherche de publications scientifiques. Dans cet exemple, un client souhaite récupérer des rapports de recherche parus dans un laboratoire (p.ex. INRIA).

1. à l'aide des requêtes d_1 et d_2, il interagit tout d'abord avec deux services $biblio_1$ et $biblio_2$ (p.ex. bibliothèque), afin de récupérer deux listes. Les données d_1 et d_2 peuvent contenir des requêtes du type « donnez-moi la liste des dernières publications internes depuis telle date ». Par exemple, ces listes exhaustives récupérées contiennent respectivement les dernières parutions des thèmes génie logiciel (premier service) et réseau (deuxième service), classées par auteurs et titres.

2. dans un second temps, ces listes sont filtrées et fusionnées à l'aide du traitement filtre, défini par le client, pour sélectionner, par mots-clés, les documents qu'il souhaite télécharger. Ces documents sont sur le serveur distant du laboratoire, au format PostScript. Sur ce serveur, un premier service primitif *bibtex* fournit des fichiers BiBTEX en fonction des champs auteurs/titres fournis. Un second service *ps* fournit les fichiers PostScript. Un traitement compress, est utilisé dans l'intention de limiter le trafic engendré par le téléchargement.

Ce service complexe peut être représenté par l'expression abstraite suivante et par le diagramme de la Figure 4.9 :

$$\textbf{let } r_1 \ = \ \mathsf{filtre}(biblio_1 \ d_1, biblio_2 \ d_2) \textbf{ in } (bibtex \ r_1, \mathsf{compress}(ps \ r_1))$$

La construction **let** permet de partager le résultat de filtre comme donnée d'entrée aux services *bibtex* et *ps*. À partir d'un diagramme, il est possible de déduire l'expression abstraite constituée exclusivement de **let**. Une boîte du diagramme, correspondante à une fonction f, est décrite par ses entrées $(x_1, ..., x_n)$ et ses sorties $(y_1, ..., y_p)$:

$$
\begin{array}{l}
\textbf{let } (x_1, ..., x_n) \ = \ E \textbf{ in} \\
\textbf{let } (y_1, ..., y_p) \ = \ f(x_1, ..., x_n) \textbf{ in} \\
(y_1, ..., y_p)
\end{array}
$$

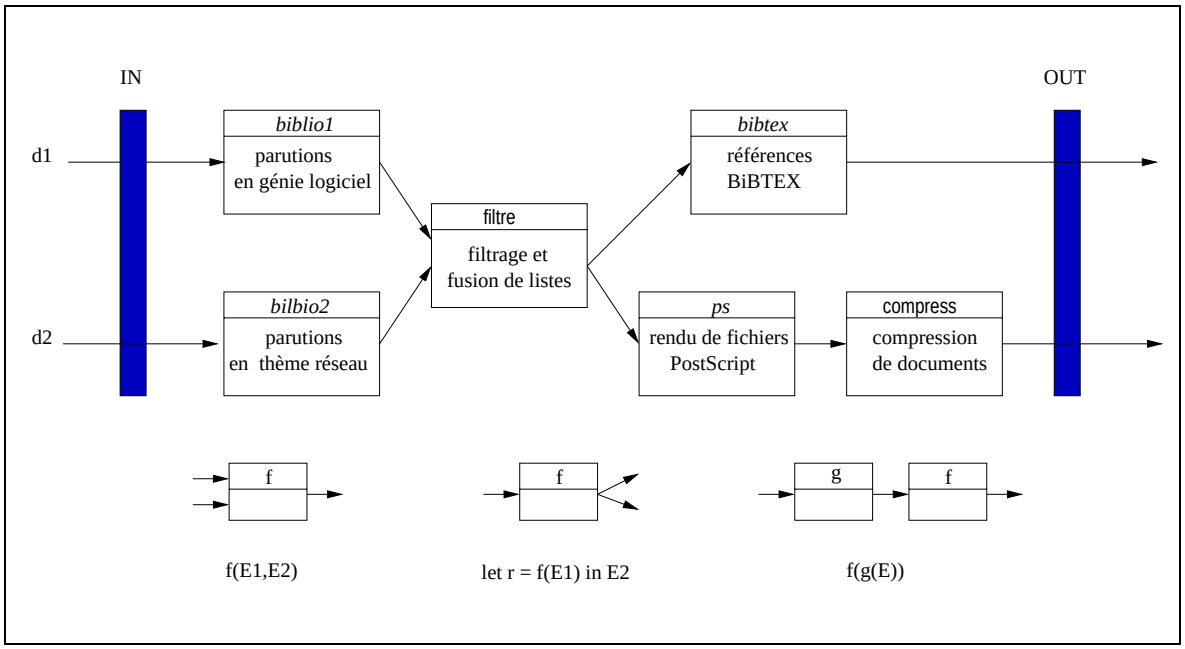

FIG. 4.9 – *Exemple 3 : un service de recherche de publications*

Les boîtes peuvent alors être facilement composées. Pour le diagramme de la Figure 4.9, l'expression déduite (sémantiquement équivalente à la précédente) est de la forme :

$$\begin{array}{l}\textbf{let } r_1 = \mathit{biblio}_1\ d_1 \textbf{ in}\\ \textbf{let } r_2 = \mathit{biblio}_2\ d_2 \textbf{ in}\\ \textbf{let } r_3 = \mathsf{filtre}(r_1, r_2) \textbf{ in}\\ \textbf{let } r_4 = \mathit{bibtex}\ r_3 \textbf{ in}\\ \textbf{let } r_5 = \mathit{ps}\ r_3 \textbf{ in}\\ \textbf{let } r_6 = \mathsf{compress}\ r_5 \textbf{ in } (r_4, r_6)\end{array}$$

L'utilisation de cette construction n'a pas directement d'impacts sur le nombre d'expressions concrètes. Par exemple, pour une expression abstraite constituée exclusivement de constructions **let** (cf. expression déduite précédente), il est toujours possible de compiler l'expression abstraite en ses concrètes en adaptant les algorithmes qui ont été fournis. Les dépendances de l'expression abstraite sont liées à la réutilisation des r_i. Si aucune dépendance n'apparaît dans l'expression abstraite, les différents **let** sont indépendants. Dans ce cas, nous retrouvons la complexité de la construction n-uplet qui va produire $n!$ itinéraires d'agents.

4.4.2 Ordre supérieur

L'extension du langage abstrait à l'ordre supérieur permettrait de décrire des interactions du type code à la demande (cf. section 1.3.2, page 14). De notre point de vue, le mécanisme de code à la demande représente un client qui contacte un service afin de récupérer un traitement à être appliqué en local (sur le site du client). La demande d'une appliquette peut se faire à la manière d'une invocation distante dans le langage concret. Suivant la finesse choisie, l'objet récupéré par le client est soit une fonction (ordre supérieur) soit une expression concrète quand l'appliquette reçue interagit avec des sites distants (le plus souvent le site qui a envoyé

l'appliquette). Le deuxième cas est particulièrement compliqué à intégrer dans nos analyses car, sans connaissances sur l'expression concrète d'une appliquette (on se limite aux interfaces), les choix d'interactions et d'itinéraires pour la synthèse d'une description de mise en œuvre restent indépendants.

Pour le premier cas, les appliquettes sont généralement utilisées pour éviter de trop nombreuses interactions à distance avec un serveur. Par exemple, un service qui requière des données en entrée valides peut être construit à l'aide d'une appliquette. L'appliquette est exécutée sur le site du client qui fournit ses entrées. La validité de celles-ci est vérifiée directement sur le site du client jusqu'à ce qu'il fournisse les bonnes entrées. Ensuite, le service distant est invoqué avec ces entrées. Seules les données valides transitent vers le serveur. Ainsi, si un service s nécessite un pré-traitement pt avant d'être invoqué, il est raisonnable de le laisser apparaître dans l'expression abstraite, sous un type différent des services ou traitements. Ce traitement pt (appliquette) doit nécessairement être exécuté sur le site client. Il est à représenter par la taille de son code, ses interfaces fonctionnelles et ses niveaux de sécurité. Ses données arguments sont les données arguments du service s et ses résultats également. Dans les choix d'expressions concrètes pour interagir avec pt vont rester l'invocation distante et les agents mobiles. Pour une invocation distante avec s, il est nécessaire auparavant de réaliser une invocation de pré-traitement (argument vide, résultat de la valeur du code de pt). Pour une interaction par agent mobile, pt va imposer un retour chez le client avant d'interagir avec s, réduisant ainsi les choix d'itinéraires.

Les analyses proposées dans ce chapitre peuvent donc s'adapter aux appliquettes simples moyennant quelques modifications. La complexité s'en trouve minimisée pour la synthèse d'une description de mise en œuvre, vu que les pré-traitements, plutôt que d'induire de nouveaux choix d'itinéraires, limitent leur nombre.

4.4.3 Contraintes globales

L'approche que nous avons suivie limite la spécification des dépendances aux résultats des fonctions. Elle est fortement liée à la composition de fonctions. Ainsi, dans une expression abstraite de la forme $f(E)$, l'application du service ou du traitement f se fait après la réalisation de l'expression E. Dans la construction n-uplet, si E est de la forme (E_1, E_2) (p.ex. $bibtex\ r_1, \mathsf{compress}(ps\ r_1)$), il n'est pas possible d'imposer un ordre d'application des fonctions de E_1 envers celles de E_2, tant que des données ou résultats ne sont pas partagés. Il est parfois souhaitable de spécifier des contraintes globales de dépendances entre les sous-expressions E_1 et E_2. Informellement, sur l'exemple 3 précédent, nous pourrions associer des contraintes du type :

- si *bibtex* est exécuté alors, compress a du être exécuté auparavant,
- si *ps* et compress sont tous deux exécutés, alors *bibtex* a du être exécuté auparavant.

De telles contraintes de précédence peuvent être décrites à part dans un langage *ad-hoc*. La formalisation de celles-ci peut s'appuyer sur une logique avec un opérateur de précédence ou sur des travaux issus de la communauté *workflow* (p.ex. [Kle91]).

Une expression concrète doit nécessairement respecter ces contraintes. Par construction, dans la réécriture d'une expression abstraite en ses concrètes, ces contraintes sont utilisables dans l'algorithme de vérification de la validité de l'agencement des interactions. Quand une liste d'interactions contenant les fonctions d'une contrainte est générée, la relation de précédence

est vérifiée. Les expressions concrètes produites respectent alors les dépendances plus fines du service complexe. Dans tous les cas, les contraintes globales permettent de minimiser le nombre d'expressions concrètes valides. En fonction de la portée d'une contrainte, la complexité de la synthèse d'une description de mise en œuvre peut se trouver réduite si beaucoup d'itinéraires d'agents sont invalidés.

4.4.4 Choix

Le langage abstrait ne permet que des spécifications sans choix dynamiques. Une fonction apparaissant dans une expression doit nécessairement être appliquée pour que le service complexe soit rendu. Au niveau d'une expression concrète, cette approche laisse clairement apparaître les choix d'interactions et les itinéraires d'agents sont «câblés» statiquement. Dans le domaine du *workflow* (cf. section 3.5.2), certains langages permettent d'apposer des conditions de transitions entre les tâches (p.ex. [Ley01]). Elles sont utilisées par le gestionnaire afin d'adapter l'exécution du *script*, le plus souvent en fonction des valeurs des données transmises entre les tâches. Par exemple, à la spécification, il n'est pas toujours possible de déterminer statiquement si un service primitif fournit un résultat vide ou non (p.ex. pas d'actualité sur un thème donné, pas de vols dans un service de réservation d'avion). Le type du résultat d'un service primitif peut alors conditionner la réalisation d'un service complexe.

Nous proposons de traiter de telles conditions de transitions sur les résultats de services complexes. Ces conditions, appliquées sur des données et à résultat booléen, peuvent porter sur des critères fonctionnels ou des propriétés non fonctionnelles. Les critères fonctionnels s'appuient sur le domaine de valeur du type de la donnée étudiée (p.ex. Int, $Bool$). Les critères non fonctionnels correspondent principalement aux propriétés de performance (c.-à-d. taille de la donnée) et éventuellement de sécurité (niveau d'une donnée s'il est possible de le déterminer, p.ex. chiffrée ou non). La propriété de fiabilité n'est pas prise en compte car elle ne s'associe pas directement aux données dans notre cadre.

Une façon de représenter les conditions dans le langage abstrait est d'introduire un choix (construction $\square$). Il est difficile de prendre en compte la condition concrète dans les analyses statiques. Pour la mise en œuvre du service complexe, ce choix est une expression conditionnelle concrète. Au niveau de nos langages de spécifications, un choix reste abstrait (c.-à-d. il ne peut pas être déterminé). La grammaire est étendue avec la construction suivante :

$$\textbf{let } (r_1, ..., r_n) = E_1 \textbf{ in } E_2 \square E_3$$

À l'exécution, les valeurs des r_i conditionnent le choix. La construction **let** est utilisée afin que les résultats $(r_1, ..., r_n)$ d'une expression E_1 puissent être réutilisés comme données arguments dans les expressions E_2 et E_3.

Exemple

Par exemple, pour le service de presse personnalisé (cf. exemple 1, Figure 3.3, page 52), il est possible de conditionner le service complexe en fonction du résultat fourni par le service *actu* (actualités sur un thème donné). Si ce service primitif ne dispose pas d'actualités sur le thème donné (requête d_2), il n'est pas nécessaire d'utiliser successivement le service de traduction *trad*. La spécification du service complexe va donc utiliser une condition sur le résultat du service *actu* :

$$\textbf{let } r_1 = actu\ d_2 \textbf{ in } \mathsf{fusion}(\mathsf{select}(journal\ d_1), trad(r_1, d_3)) \square \mathsf{select}(journal\ d_1)$$

Dans cet exemple, le client dispose initialement des données d_1, d_2 et d_3, ainsi que des traitements fusion et select. La mise en œuvre du service complexe va tout d'abord imposer de réaliser le sous-service complexe de la « garde », ici l'interaction avec le service primitif *actu*. Par exemple, si la taille du résultat de *actu* d_2 n'est pas nulle, le service complexe est à réaliser à la manière de l'exemple 1 (expression E_2). Sinon, il n'est pas nécessaire de poursuivre le service complexe avec le service de traduction, ni d'appliquer le traitement fusion (expression E_3). Il suffit de se limiter à l'utilisation du service *journal* et au filtrage par mots-clés de son résultat (traitement select). Deux services complexes sont donc à considérer. La Figure 4.10 propose graphiquement ces deux services complexes.

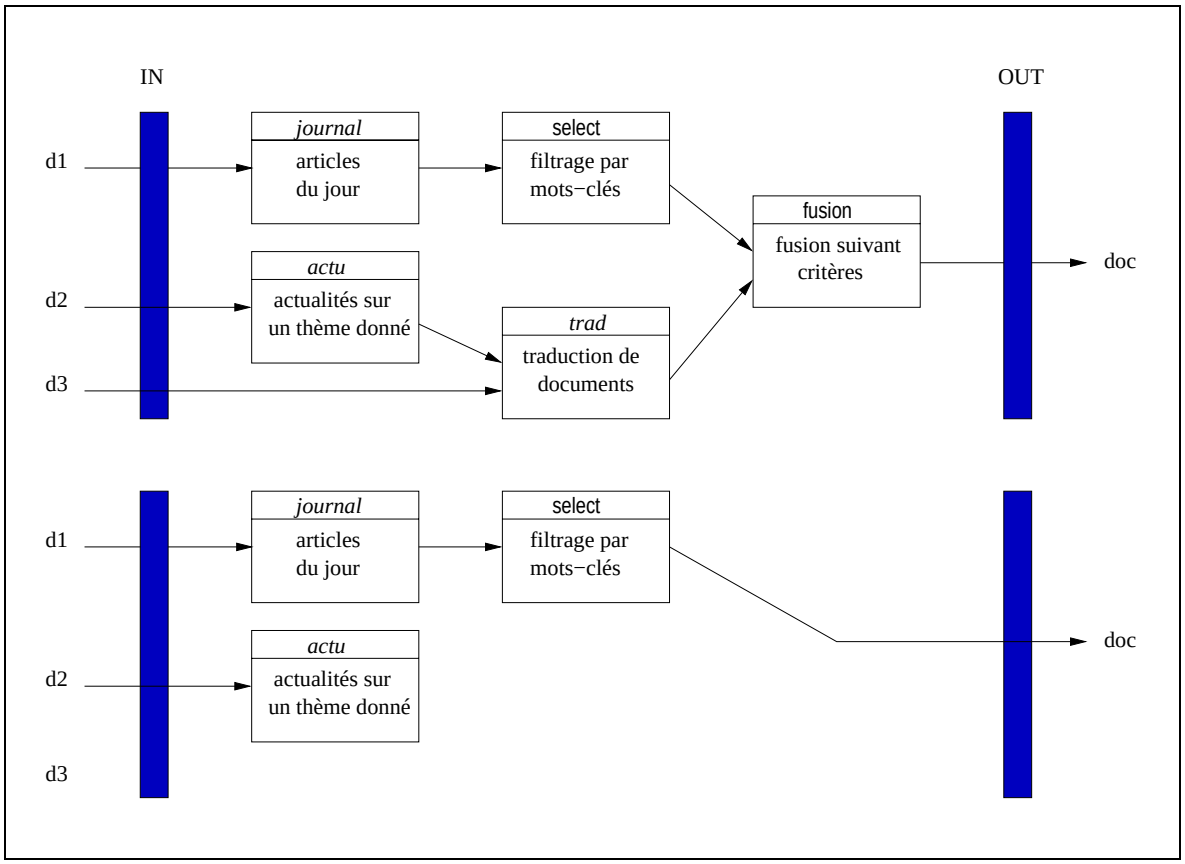

FIG. 4.10 – *Choix sur un service complexe de presse*

Extension sur le langage concret

Représenter une mise en œuvre d'un service complexe qui utilise des choix impose de les laisser apparaître dans le langage concret. En effet, le but du langage concret et de ses analyses est de permettre de comparer différentes mises en œuvre. Les analyses doivent donc traiter des différentes alternatives d'exécution. L'expression concrète produite doit en fait supporter les différentes mises en œuvre possibles, dont les interactions sont déterminées à l'exécution. Nous proposons donc d'étendre le langage concret en laissant apparaître également les choix :

$$\textbf{let } (r_1, ..., r_n) = A\ D \textbf{ in } E_1 \ \square\ E_2$$

Comme pour le langage abstrait, les résultats de l'interaction $A\ D$ peuvent être réutilisés comme données arguments dans les expressions E_i. Sur l'exemple précédent, une expression concrète qui intègre les deux choix peut être :

$$\begin{array}{l}\textbf{let } r_1 \ = \ (go; \overline{actu}; go; end)\ d_2 \textbf{ in} \\ \quad \textbf{let } r_{doc} \ = \ (go; \overline{trad}; go; \overline{journal}; \overline{\mathsf{select}}; \overline{\mathsf{fusion}}; go; end)\ (d_1, r_2, d_3) \textbf{ in } r_{doc} \\ \quad \square \textbf{ let } r_{doc} \ = \ (go; \overline{journal'}; \overline{\mathsf{select}'}; go; end)\ d_1 \textbf{ in } r_{doc}\end{array}$$

Il paraît difficile de laisser apparaître un choix directement dans une interaction $A\ D$ (c.-à-d. choix durant l'itinéraire). Une interaction par agent comprenant un choix à la manière d'une primitive de migration (go) ou d'une fonction ($\overline{f}$), impose de subdiviser le reste de la continuation, remettant en cause le sac à dos de l'agent (c.-à-d. données et traitements). Il paraît difficile d'introduire une garde dans les continuations sans nécessiter une modification importante des analyses proposées jusqu'ici. Des recherches approfondies pourraient être à menées afin de disposer d'une réelle adaptation des agents. Nous choisissons donc de limiter le propos à des choix effectués sur le site client, au retour de résultats suite à une interaction, comme cela se fait dans les invocations distantes.

Lors de la génération de code, nous proposons d'utiliser une méthode d'adaptation statique chez le client. Les choix sont exécutées chez le client. Selon la nature des résultats, les interactions suivantes adéquates vont être choisies à l'exécution. Certaines d'entre elles ne seront pas exécutées, bien qu'elles soient générées sur le site client.

Impacts sur les analyses

Au niveau des analyses de qualité proposées, l'inférence de la qualité de mise en œuvre va nécessiter de traiter les différentes alternatives des conditionnelles. La complexité, précédemment linéaire en fonction du nombre de fonctions de l'expression abstraite, va s'accroître afin de pouvoir traiter intégralement tous les choix. À une $\square$ correspondent deux branches. Ainsi, au pire cas, il est nécessaire d'étudier 2^n expressions complètes, où n est le nombre de gardes. Pour la performance, la valeur d'une expression concrète avec choix va correspondre à la valeur maximale des deux branches. L'inférence de la qualité fournit alors une valeur au pire cas, bien qu'il soit possible de considérer un cas moyen ou un meilleur cas. Pour la sécurité, le minimum des niveaux des deux branches est rendu. Pour la fiabilité, vu que les choix sont traités sur le site client, il suffit de chercher à vérifier que les deux branches disposent d'un identifiant pour mécanisme de tolérance aux fautes ou de résistance aux pannes.

Pour la synthèse d'une description de mise en œuvre, la recherche de l'expression concrète « optimale» va nécessiter de comparer/sélectionner les différentes branches alternatives. Dans le cas où les choix sont susceptibles d'être déterminés sur les itinéraires d'agents, l'analyse se complique sérieusement.

Pour la synthèse d'architecture, le placement des services à partir d'une expression concrète ne change pas, tant qu'un service non localisé n'apparaît pas dans les deux branches alternatives. Si c'est le cas, il convient de faire le choix en fonction d'une valeur médiane des deux branches. Pour la recherche de propriétés sur les éléments incomplets du réseau, l'analyse est identique tant que les branches alternatives n'imposent pas de compromis.

4.5 Discussion

Notre approche nous a permis de définir de manière uniforme des analyses simples calquées sur la sémantique du langage concret. Leur degré de précision est relatif à notre modèle d'exécution qui reste abstraite (interfaces de services, réseau, propriétés, etc.). Par exemple, pour la performance, la taille des résultats rendus par un service n'est pas toujours connue statiquement. Toutefois, nos mesures fournissent des guides pour l'analyse de la qualité d'un service complexe. Certaines des analyses présentées sont exploitables automatiquement pour des spécifications raisonnables. Elles peuvent être utilisées de manière systématique ou bien sous la forme d'une aide au concepteur. Il est facile de faire participer le concepteur à certains choix pour réduire la complexité d'une analyse trop coûteuse. Avec cette aide, les analyses les plus complexes peuvent alors être envisagées.

Les analyses des trois propriétés (c.-à-d. performance, sécurité et fiabilité) peuvent souvent être réalisées simultanément pour plus d'efficacité. Par exemple, pour la synthèse d'une description de mise en œuvre, les propriétés de sécurité et de fiabilité considérées au fur et à mesure des choix de mise en œuvre permettent de réduire considérablement le nombre d'expressions concrètes à comparer envers le critère de performance. D'un autre côté, des choix de conception pour une propriété donnée peuvent considérablement influer sur une autre propriété et conduire à des problèmes de cohérence et consistance [FLP99], particulièrement dans la synthèse d'architecture. Ce problème reste un axe de recherche à part entière. Le concepteur doit alors donner ses priorités. Par exemple, l'introduction d'un mécanisme pour traiter la sécurité (chiffrement) a un impact sur la performance (taille du traitement). Vouloir assurer des propriétés de fiabilité ou de sécurité dans un système génère la plupart du temps des dégradations au niveau de la performance du système final (volume de données en transit, authentification).

Afin de présenter plus concrètement la portée de notre cadre et certaines de ses analyses, nous proposons dans la partie suivante son intégration dans un environnement de construction de services distribués complexes. Dans le dernier chapitre de cette thèse, nous nous sommes restreints au langage abstrait donné dans la Figure 3.1. Le passage au langage étendu nécessite d'affiner les mises en œuvre proposées par la suite. L'objectif est de guider, le plus automatiquement possible, la conception et l'implantation du service complexe, ainsi que le développement de l'environnement d'exécution qui le supporte, tout en respectant les contraintes de qualité. Notre proposition utilise un langage de description d'architectures [MT00] pour la spécification et s'appuie sur l'environnement Aster [IB96] pour le développement du système.

Chapitre 5

Environnement de construction

Il existe deux modes de construction d'un service complexe : (i) le mode utilisateur où un client utilise simplement une spécification réseau complète qui correspond à un système d'exécution existant (ii) le mode prestataire où une compagnie (ou organisation) dispose de services primitifs et de sites qui lui sont propres et qui utilise des services extérieurs. Ces deux modes sont décrits schématiquement dans la Figure 5.1 qui laisse apparaître les sites clients et ceux des services primitifs. Les boîtes grisées représentent la connaissance des clients sur les objets (p.ex. données, services, traitements). Ainsi, dans le premier mode, un client se compose son service complexe à l'aide des différents services primitifs qu'il connaît sur le réseau. Il utilise ses données pour effectuer les requêtes et des traitements personnalisés pour étendre des services primitifs. Tous les éléments du réseau sont définis. L'environnement d'exécution du service complexe est donc fixé et est constitué de systèmes inter-connectés. La construction du service complexe se limite à la construction de l'application qui comprend les étapes de spécification, de conception et d'implantation :

1. la spécification porte sur la description abstraite du service complexe et du réseau (complet) sur lequel un système d'exécution est déjà déployé,

2. la conception s'attache à effectuer des choix de mise en œuvre pour la réalisation du service complexe,

3. l'implantation consiste à générer le code des interactions à partir d'une expression concrète et à l'installer.

La terminologie employée dans ce chapitre est liée à celles utilisées notamment dans [IB96, GM01]. Nous ne suivons pas directement la terminologie de la classification des activités du cycle de vie du logiciel proposée par l'ISO (Organisation Internationale de Normalisation) [ISO95], bien qu'elle s'en rapproche. Dans la modélisation du cycle de vie du logiciel proposée par l'ISO (modèle classique en cascade (*waterfall model*)), le développement du logiciel comprend les étapes de :

1. spécification (*requirements phase*) et conception générale,

2. conception détaillée (*detailed design phase*),

3. codage/integration/installation (*implementation phase*)

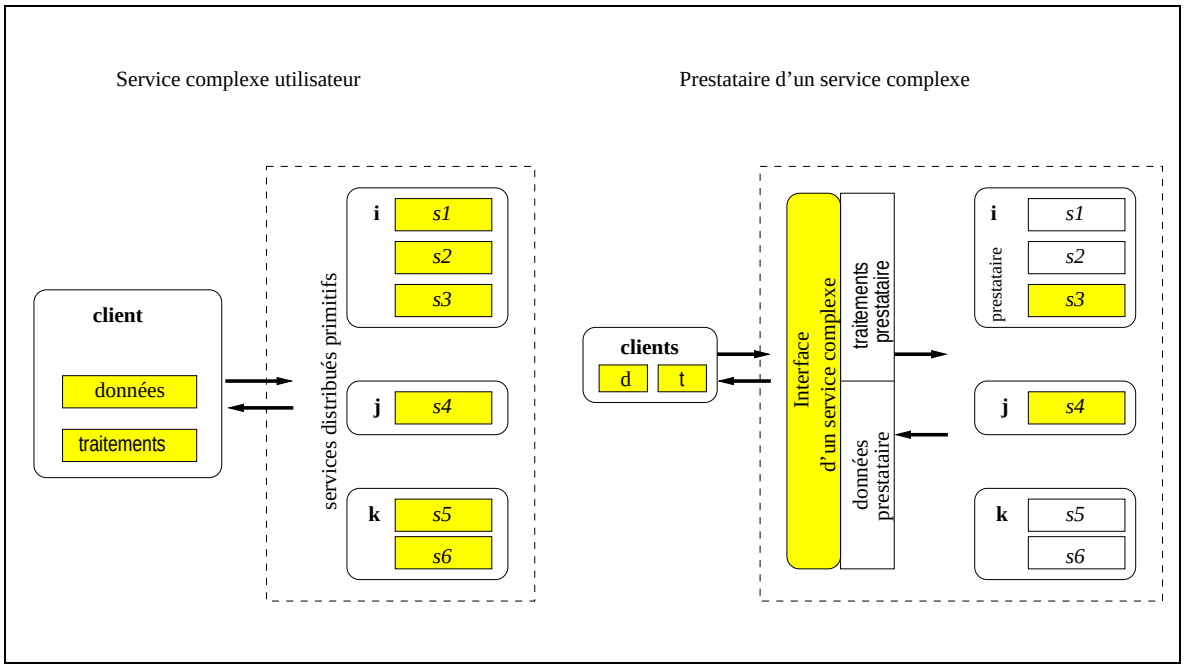

FIG. 5.1 – *Prestation d'un service distribué complexe*

Notons que nous utilisons le terme construction (et non pas développement) pour représenter les différentes étapes du procédé de mise à disposition d'un service distribué complexe.

Dans le second mode, un prestataire met à disposition de ses clients potentiels un service complexe à travers une interface primitive sur l'un de ses sites. Les clients interagissent avec le service complexe construit comme s'il était primitif. Ce prestataire utilise des sites/services qui lui sont propres (p.ex. site *i* sur la figure) et des services primitifs déjà déployés sur le réseau (éventuellement inconnus des clients potentiels). Les sites et services qui lui sont propres peuvent être incomplets dans le réseau (p.ex. pas de localisation associée ou pas de propriétés, cf. section 4.2.2). L'environnement d'exécution du service complexe n'est donc pas totalement développé. Certains services ne sont pas encore déployés ou vont requérir des mécanismes de sécurité ou de tolérance aux fautes. Le prestataire doit alors implanter ses services primitifs quand ils ne sont pas localisés et/ou développer les parties du système qui vont permettre de garantir la qualité de service exigée par les interactions du service complexe. Dans ce mode, le procédé de construction du service complexe porte donc sur la construction de l'application et sur le développement [1] des parties du système qui la supportent. Les étapes à suivre sont :

1. la **spécification** qui porte sur la description abstraite du service complexe et du réseau sur lequel un système d'exécution est/va (être) déployé,

2. la **conception** qui s'attache à effectuer (i) des choix de mise en œuvre pour la réalisation du service complexe et (ii) des choix de localisation et de propriétés pour le système,

3. le **développement** des parties du système supportant l'application et garantissant la qualité exigée. Le développement du système porte sur sa conception et son déploiement.

1. Afin de garder une terminologie non ambiguë pour se différencier de la construction de l'application, nous utilisons le terme «développement», comme dans [IB96], pour caractériser la construction (spécification, conception, déploiement) de l'environnement d'exécution, quand les spécifications sont données.

Le **déploiement** réfère à la distribution des éléments, leur installation et leur configuration, Le développement peut porter sur (i) la simple configuration de l'environnement d'exécution pour un site, (ii) l'enrichissement du système à l'aide de composants matériels/logiciels ou (iii) la spécialisation du système avec des composants logiciels système qui l'adaptent aux exigences de l'application.

4. l'**implantation** qui consiste à générer le code des interactions à partir d'une expression concrète (le service complexe) et à l'installer.

La construction d'un service complexe et de son environnement d'exécution par un prestataire de services peut être un véritable procédé d'ingénierie. Un architecte spécifie le service complexe, qui doit être conçu concrètement par un concepteur et enfin implanté par l'équipe de développement. L'équipe de développement a également en charge de développer des parties du système si des éléments du réseau n'ont pas encore été déployés. Un même acteur peut prendre en charge seul toutes les étapes du procédé de construction. Cependant les rôles des acteurs sont le plus souvent différenciés. Pour garantir un service complexe de qualité, maintenable, évolutif ainsi qu'une coopération efficace entre les différents acteurs du procédé de construction, l'utilisation d'un cadre de conception est primordiale. Les phases de haut niveau dans le procédé de construction, telles que la spécification ou les analyses, ont un apport non négligeable pour ces garanties [SG96]. En ce sens, nous proposons un environnement de construction de services complexes de prestataire, apte à traiter au mieux de la construction de tels services au regard de critères de qualité (c.-à-d. performance, sécurité, fiabilité). Cet environnement intègre les moyens de spécifications proposés dans le chapitre 3 (c.-à-d. langage abstrait et concret) ainsi que la majorité des analyses proposées dans le chapitre 4. Cette intégration permet au concepteur et à l'équipe de développement de bénéficier d'outils voués, autant que faire se peut, à automatiser la construction d'un service complexe et le développement de l'environnement d'exécution.

Ce chapitre vise à valider l'approche proposée et certaines des analyses en présentant une intégration possible dans un environnement de construction. Nous présentons tout d'abord dans la section 5.1 une vue générale de notre environnement. Elle combine les rôles de l'architecte, du concepteur et de l'équipe de développement ainsi que tous les entrelacements possibles d'analyses. Dans la section 5.2, nous décrivons et justifions le cadre de spécification que nous avons choisi, basé sur un langage de description d'architectures. La section 5.3 propose une utilisation possible de l'environnement, dans une suite d'étapes de construction. Cette suite fournit un agencement possible de différentes analyses, en fonction des trois propriétés de qualité étudiées. La séquence d'analyse proposée conduit à une spécification complète du réseau et du service complexe à implanter. Pour illustrer notre propos, nous appuyons cette section sur un exemple et sur un premier prototype qui a été développé. Ce prototype met en œuvre certaines des analyses présentées et permet une implantation automatique sur un intergiciel conforme au standard MASIF, qui utilise à la fois des interactions par invocations distantes et par agents mobiles.

5.1 Vue générale de l'environnement

La Figure 5.2 présente notre proposition d'environnement de construction de services complexes. Sur la partie gauche de la figure sont représentés les différents acteurs (c.-à-d. architecte, concepteur et équipe de développement). Les rôles/actions de chacun correspondent aux boîtes

présentes sur la droite. Ces rôles seront explicités par la suite. Le plus souvent, les boîtes aux bords francs correspondent aux informations nécessaires à l'environnement (p.ex. les spécifications). Celles aux bords arrondis correspondent aux analyses/outils utilisables dans l'environnement. Les boîtes sont grisées quand les informations sont nécessaires. Cet environnement s'appuie sur nos langages de spécification (langage abstrait) et de description de mise en œuvre (langage concret) et utilise la plupart des analyses associées. Il permet de concevoir et implanter un service complexe sur une plate-forme donnée (p.ex. plate-forme à agents mobiles). Cette plate-forme doit permettre au minimum des interactions de type invocation distante. Dans le meilleur des cas[2], elle permet également des interactions par agents mobiles. Cette plate-forme est l'environnement d'exécution du service complexe qui s'appuie sur un système de communication. Suivant la plate-forme, l'environnement d'exécution peut être enrichi à l'aide de composants matériels et/ou logiciels pour garantir des propriétés de qualité sur des sites ou liens (p.ex. *trusted hardware*). Le système de communication peut parfois être spécialisé, à l'aide de composants système, afin de garantir/optimiser les propriétés de qualité exigées par l'application. Nous présentons ci-après les rôles de chacun des acteurs dans cet environnement et les outils dont ils peuvent profiter.

5.1.1 Rôle de l'architecte

L'architecte à en charge de spécifier le service complexe à construire. Il ne se soucie que peu des détails de la plate-forme et des détails de mise en œuvre. C'est principalement la fonctionnalité du service complexe qui l'intéresse. Il délègue sa conception aux autres acteurs. Il a en charge de fournir aux autres acteurs du procédé d'ingénierie un certain nombre de spécifications, utilisées tout au long de la conception, de l'implantation. Il fournit ses spécifications, précises et structurées, à l'aide d'un langage de description d'architectures. Nous présentons et justifions ce cadre de spécification dans la section 5.2. L'architecte peut fournir quatre types de spécifications. Deux d'entre elles sont nécessaires à la construction d'un service complexe (boîtes aux bords pleins), alors que les deux autres sont optionnelles (boîtes aux bords en pointillés). Il doit impérativement fournir :

1. la description du service complexe sous la forme d'une expression abstraite (c.-à-d. sa fonctionnalité). Il utilise pour cela des identificateurs de services primitifs disponibles sur le réseau, de traitements particuliers (qu'il a développé ou qu'il fait développer) et de données. Si le service complexe est rendu disponible par un prestataire, son interface fonctionnelle doivent être accessible aux clients potentiels. Dans une version étendue de l'environnement et suivant son niveau d'expertise, il peut associer à cette expression des contraintes globales (cf. section 4.4.3) ou des choix de mises en œuvre (p.ex. un unique agent pour la réalisation).

2. la configuration du réseau et les propriétés de qualité de ses éléments (c.-à-d. vue réseau). La construction d'un service complexe et l'étude de ses propriétés de qualité nécessitent de connaître sans ambiguïté des informations sur le réseau. Ces spécifications peuvent être issues d'acteurs tiers. Deux modes de spécification du réseau sont envisagés, suivant qu'il est entièrement défini ou non (incomplet) :

   (a) le mode + où la configuration du réseau est connue (cf. section 4.2). Le système qui

2. Dans les réseaux grande échelle, des environnements d'exécution d'agents ouverts sont très loin d'être légion (p.ex. services Web).

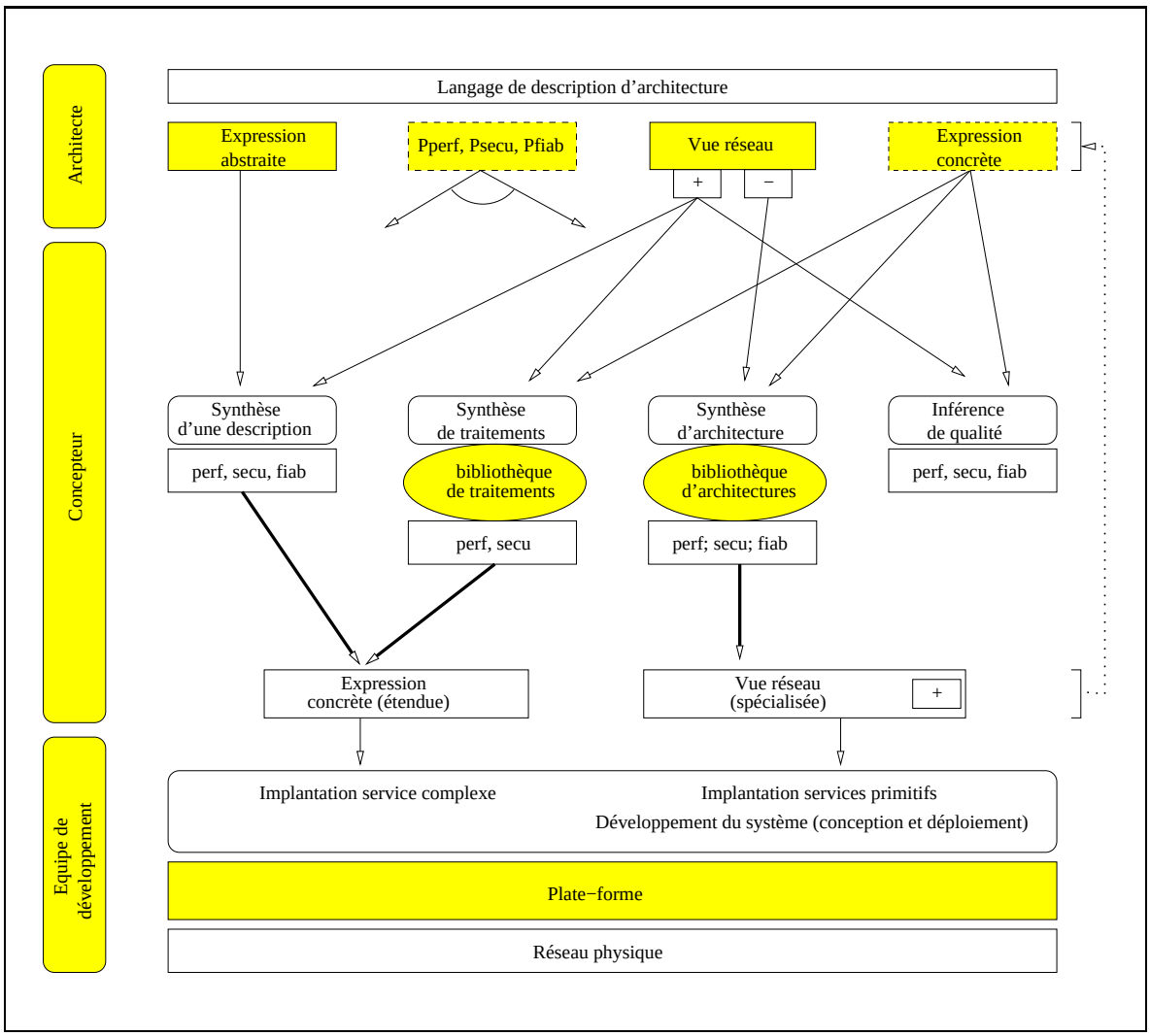

FIG. 5.2 – *Vue schématique de l'environnement*

supporte l'application est dans ce cas déjà déployé. La configuration comprend les descriptions des services de l'expression abstraite et des liens qui les interconnectent. Pour chacun de ces sites et liens, les propriétés de qualité qu'ils garantissent (c.-à-d. matériels ou mécanismes spécifiques implantés), sont données. Ces propriétés portent sur la performance, la sécurité et la fiabilité (cf. chapitre précédent).

(b) le mode – (spécification incomplète qui correspond à des parties du système non encore déployée) où soit la configuration du réseau n'est pas encore entièrement définie, soit des éléments n'ont pas de propriétés de qualité associées. Par exemple :

- un prestataire qui construit un service complexe, en plus de réutiliser des services primitifs déjà implantés (internes ou externes), peut vouloir intégrer de nouveaux services primitifs au sein de son architecture de services (cf. section 4.2.2.1). Ces services primitifs sont à placer sur des sites de son organisation ou sur des sites dédiés (p.ex. ASP).
- des sites ou des liens peuvent être prêts à être déployés (cf. section 4.2.2.2). Il peut être nécessaire de configurer l'environnement d'exécution à l'installation d'un site pour assurer une propriété. Plus techniquement, sur des sites ou liens, il est possible d'associer des composants aptes à garantir/optimiser certaines propriétés de qualité sur des parties du système. Si ces sites ou liens n'ont pas encore été déployés ou sont spécialisables (p.ex. enrichissement du système de communication), les spécifications d'une ou des propriétés de qualité associées sont non définies.

L'architecte a également la possibilité de fournir (optionnellement, boîtes en pointillés) :

1. les propriétés de qualité qu'il souhaite voir vérifiées par le service complexe une fois construit. Cette boîte de spécification sert de donnée d'entrée aux différentes analyses (arc de flèches utilisé sur la figure pour ne pas surcharger). Ces propriétés représentent en quelque sorte la vue non fonctionnelle du service complexe. Elles portent sur la performance (P_{perf}), la sécurité (P_{secu}) et la fiabilité (P_{fiab}). L'expression de ces propriétés est présentée en section 5.3.2. La sécurité comprend à la fois la propriété de confidentialité et la propriété d'intégrité. L'architecte peut imposer qu'une, deux ou toutes les propriétés soient à respecter par le service complexe une fois implanté. S'il exige une propriété, il l'annote avec la valeur minimale qu'il souhaite voir garantie (p.ex. un seuil à ne pas dépasser pour la performance, une valeur du treillis de sécurité, un identificateur de mécanisme voué à améliorer la fiabilité). Il peut également requérir une propriété sans lui associer de valeur. Dans ce cas, la recherche de valeur correspond implicitement au coût minimal pour la performance, au plus grand niveau pour la sécurité et à des identificateurs de mécanismes (invocation distante et agent mobile) pour la fiabilité.

2. une expression concrète destinée à mettre en œuvre le service complexe, dans le cas où l'architecte aurait un niveau d'expertise important. Cette expression peut être complète ou bien comporter certains éléments non définis (p.ex. identificateurs de sites associés aux *go*).

Une fois ces spécifications données ou réutilisées, l'architecte les transmet au concepteur.

5.1.2 Rôle du concepteur

À partir des spécifications fournies par l'architecte, le concepteur doit chercher à concevoir le service complexe demandé, c'est à dire proposer une expression concrète (si elle n'a pas été fournie), associée à une vue réseau (si non complètement spécifié), qui vérifient les propriétés de qualité exigées. Une exigence est définie quand elle correspond à une valeur du domaine de la propriété et est implicite quand elle en est une borne sup/inf.

Dans tous les cas, le concepteur doit proposer à l'équipe de développement une expression concrète et une vue du réseau où tous les éléments sont localisés et associés à d'éventuels composants matériels, logiciels ou système. Pour cela, nous proposons quatre analyses (boîtes aux bords arrondis du concepteur), dont les données d'entrée sont représentés par des flèches fines dans la Figure 5.2, et les données résultats par des flèches épaisses. Ce sont :

1. la synthèse d'une description de mise en œuvre (cf. section 4.2.1) qui permet, à partir d'une expression abstraite, d'une vue réseau complète et d'une ou des propriétés de qualité, de produire une expression concrète qui représente une mise en œuvre du service complexe (cf. flèche épaisse en résultat de l'analyse). Si exigées, les trois propriétés de qualité peuvent être traitées de manière combinée lors de cette étape (cf. la représentation à l'aide de virgules dans la figure). Il est possible de rechercher la meilleure mise en œuvre en terme de performance qui garantit au mieux la sécurité et la fiabilité en une même étape d'analyse.

2. la synthèse d'architecture (cf. section 4.2.2) qui permet, à partir d'une propriété de qualité exigée, d'une expression concrète et d'une spécification du réseau incomplète, de produire une spécification complète du réseau en terme de localisation des éléments et des mécanismes associés. Afin de garantir/optimiser une propriété de qualité sur des sites ou liens appartenant au domaine, des bibliothèques d'architecture (composants/connecteurs) qui correspondent à des mécanismes sont utilisées (cf. ensemble grisé sur la figure, qui représente une connaissance). Les squelettes d'architectures présents dans ces bibliothèques vont servir à enrichir l'environnement d'exécution ou le spécialiser. Les trois propriétés de qualité ne peuvent pas être traitées simplement de manière combinée (cf. noté à l'aide de points-virgules dans la figure). Si plusieurs d'entre elles sont à prendre en compte, il est nécessaire de les traiter en séquence. En effet, le traitement d'une propriété influe souvent sur une autre (cf. section 4.5). Des choix de conception pour une propriété donnée peuvent considérablement influer sur une autre propriété. Ce problème reste un axe de recherche à part entière. Des recherches plus approfondies seraient à mener afin de supporter une analyse combinée des trois propriétés. Dans le cas où plusieurs propriétés sont à traiter, nous les considérerons donc indépendamment. Cette simplification conduit à préférer une seule propriété pour cette analyse. Quand plusieurs propriétés sont traitées, la dernière risque d'influer considérablement sur les précédentes.

3. la synthèse de traitements (cf. section 4.3) qui permet, à partir d'une propriété de qualité, d'une vue réseau entièrement définie et d'une expression concrète, de produire une expression concrète étendue où des traitements sont insérés. Les propriétés de qualité traitées portent sur la performance et la confidentialité, qui peuvent être combinées.

4. l'inférence de qualité (cf. section 4.1) qui permet de se donner une idée de la valeur des propriétés du service complexe à partir de son expression concrète et d'une vue réseau

complète.

Le concepteur, suivant les spécifications qu'il a reçues de l'architecte, peut utiliser ces différentes analyses de manière combinée. Par exemple, suite à la synthèse d'une description de mise en œuvre, l'expression concrète produite peut être utilisée par la synthèse de traitements afin de générer une expression concrète étendue. Les résultats d'une analyse peuvent ainsi servir de données pour une autre (cf. flèche en pointillé sur la droite de la figure), et ce jusqu'à ce qu'une expression concrète et un réseau entièrement spécifié aient été produits, pour les trois propriétés de qualité. Ces spécifications complètes servent ensuite à l'équipe de développement.

5.1.3 Rôles de l'équipe de développement

L'équipe de développement a la charge de l'implantation du service complexe (c.-à-d. codage de l'application et de ses interactions) et du développement des parties du système sur lequel l'application doit reposer. Pour cela, l'équipe dispose d'une expression concrète (représentative du service complexe à mettre en œuvre) et d'une vue réseau complète. L'expression concrète est utilisée pour générer le code des interactions, par agents mobiles et/ou par invocations distantes (les évaluations distantes pouvant être conçues à l'aide d'agents mobiles). L'équipe de développement doit connaître les interfaces d'invocation des différents services primitifs du réseau ainsi que disposer des codes sources des traitements conformes aux environnements d'exécution (cf. plate-forme).

La vue réseau correspond à une architecture déjà déployée si l'analyse de synthèse d'architecture n'a pas été utilisée. Dans le cas contraire, la vue réseau comporte des spécifications d'éléments qui ne sont pas encore déployés dans l'architecture, auxquels sont associés des propriétés. Ces propriétés attachées à des éléments correspondent à des mécanismes destinés à garantir/optimiser une propriété et correspondent à des mécanismes spécifiques présents dans les bibliothèques d'architectures. Dans ce cas, l'équipe de développement a un rôle d'administrateur responsable de l'implantation de services primitifs, et un rôle de technicien responsable de l'intégration de matériels ou de mécanismes voués à respecter des contraintes de qualité (attachés aux bibliothèques d'architectures).

La plate-forme utilisée pour le déploiement d'un service complexe peut par exemple être à base d'agents mobiles (p.ex. Grasshopper, Aglets), d'un intergiciel de type CORBA avec des invocations distantes (éventuellement interopérable avec une plate-forme d'agents mobiles) ou encore d'un intergiciel propriétaire (p.ex. .NET). Si l'expression concrète comporte des interactions par agents, la plate-forme doit bien entendu supporter de tels moyens de communication (ce qui n'est pas encore le cas dans les environnements prestataires de services Web par exemple). Ainsi, les spécifications initiales du réseau doivent intégrer en partie des informations sur les environnements d'exécution proposés par les différents sites du réseau (support pour agents mobiles). Nous verrons par la suite un moyen d'assister l'équipe de développement dans l'implantation d'un service complexe sur la plate-forme à agents mobiles Grasshopper et de la guider dans le développement du système distribué le supportant à l'aide des techniques de spécialisation proposées dans l'environnement de développement de système Aster.

5.2 Cadre de spécification

Les intergiciels utilisés pour le développement de systèmes distribués à base de composants ou de services visent à faciliter l'assemblage d'éléments indépendants mais ne permettent pas directement de raisonner sur la structure et la qualité de ces assemblages. Dans un cadre de génie logiciel distribué [Kra94], le domaine des architectures de logiciels, initialement non directement lié aux applications/systèmes distribués, permet d'ouvrir la voie à des analyses propres à prédire la qualité et faciliter le développement à partir d'une spécification haut niveau. Les récentes propositions standardisées sur les spécifications de systèmes distribués commencent à suivre cette approche (cf. standard MDA qui utilise la notation UML). Nous justifions ici de l'intérêt d'utiliser une approche de type architecture logicielle pour nos spécifications en présentant le domaine, puis nous discutons des langages de description d'architectures avant de présenter l'intégration de notre cadre formel au sein d'un tel langage. Le langage choisi est le langage Aster, étendu pour supporter nos spécifications et qui offre de nombreux outils pour guider le développement d'un système distribué au regard de critères de qualité.

5.2.1 Intérêts d'une approche architecture logicielle

Alors que les logiciels deviennent de plus en plus complexes, la structure générale d'un système ou d'une application (c.-à-d. son architecture) devient un problème central lors des étapes de conception et de maintenance. La structuration est souvent représentée comme une organisation haut niveau (c.-à-d. abstraite) de composants reliés par des connecteurs (cf. section 5.2.1.1), ceux-ci servant de médiums pour la communication, la coordination ou la coopération entre les composants. La description de l'architecture est en général peu concernée par les algorithmes et les structures de données. Elle se concentre sur l'organisation et les mécanismes d'interaction dans un cadre distribué.

Les éléments d'une architecture logicielle se divisent ainsi en trois parties. Premièrement, les **composants** définissent abstraitement les unités de calcul. Ils peuvent être par exemple des procédures, des processus, des clients ou des serveurs. Les **connecteurs** définissent abstraitement les types d'interactions entre les composants. Les interactions entre ces composants peuvent être de simples appels de procédure, des canaux de données, mais aussi des communications plus évoluées telles que des protocoles d'accès à des bases de données ou des invocations distantes [SG95]. Finalement, la **configuration** définit la structure de l'application en terme d'interconnexion des composants à l'aide de connecteurs. La configuration est une partie de l'architecture.

L'étape de conception de l'architecture joue un rôle significatif dans le succès d'un système/application. Une bonne étude et spécification de l'architecture permettent en général :

- une meilleure compréhension et une documentation non ambiguë des systèmes en les présentant de façon abstraite [SG96],
- une réutilisation simplifiée des composants et connecteurs [SG96],
- une meilleure évolution et amélioration du système [PW92],
- des analyses sur l'architecture spécifiée si celle-ci est décrite de manière formelle [BIM98, NACO97, SG98, SW98].

Le milieu de la dernière décennie [SG96] a vu la reconnaissance de l'utilité du concept d'architecture logicielle qui a conduit à des recherches tant dans l'industrie que dans les milieux académiques. Parmi les points les plus étudiés, notons particulièrement :

- l'introduction de langages (cf. section 5.2.2) ou modèles pour décrire les architectures et leurs propriétés [Gar95]. En réalisant une séparation nette entre l'organisation du système et son implantation, l'application de techniques formelles est facilitée,
- la production d'outils d'analyse des architectures [NACO97],
- l'étude du raffinement d'architectures [MQR95, Gar96],
- l'étude et la classification des styles d'architecture [GS93]. Un style architectural représente une famille d'architectures et définit des contraintes sur la forme et la structure. L'utilisation des styles ouvre la voie à des conceptions abstraites de plus haut niveau et peut conduire à des analyses plus précises que celles permises sur une architecture classique [MKMG97]. Garlan et Shaw [GS93, SG96] ont identifié une demi-douzaine de styles et ont illustré leurs avantages et inconvénients ainsi que leurs utilisations (p.ex. style client-serveur).

5.2.1.1 Notion de connecteur

Un connecteur est un élément conceptuel de la description d'une architecture qui définit les règles permettant de gouverner les interactions entre les composants. Il peut décrire des interactions simples tel qu'un appel de procédure ou un accès à une variable partagé. Toutefois, il permet de représenter des interactions plus complexes, telles que des mécanismes d'invocation distante ou des protocoles d'accès à une base de donnée. Ainsi, il peut décrire les transferts de données entre composants mais aussi les flots de contrôle. Un connecteur relie deux ou plusieurs composants. L'utilisation de connecteurs, en plus des composants, permet de spécifier des interactions complexes, de manière abstraite. La séparation claire entre les calculs ou services réalisés par les composants et leurs interactions permet de :

- minimiser les interdépendances entre composants,
- mieux supporter l'évolution du système,
- favoriser l'hétérogénéité des composants,
- aider aux analyses et tests du système/application.

En ce sens, nous avons choisi de représenter un agent mobile par un connecteur au sein de nos spécifications.

5.2.2 Langages de description d'architectures pour systèmes distribués

Afin de décrire formellement les architectures des applications/systèmes (distribués ou non), différents langages spécialisés ont vu leur apparition [MT00, Gar00]. Ces langages de description d'architectures (ADL pour *Architecture Description Language*) fournissent des notations pour déclarer les composants, les connecteurs et la configuration, c.-à-d. comment ces

éléments se combinent pour former son architecture. Deux principaux modèles d'architectures sont considérés par ces langages :

- le modèle structurel (statique) où l'architecture est complètement décrite par ses composants, connecteurs, configuration et certaines contraintes/propriétés associées,
- le modèle dynamique, complémentaire du modèle structurel, qui prend également en compte certaines propriétés comportementales telles que l'évolution dans le temps ou la reconfiguration.

Certains ADL permettent la description de styles, d'autres simplement la description d'architectures. La plupart sont associés à des outils permettant des analyses (p.ex. cohérence, raffinement). Notons que de plus en plus, les langages de description d'architectures s'attachent à utiliser des notations standardisées afin de faciliter leur diffusion. C'est notamment le cas avec la notation UML et ses diagrammes, de plus en plus adoptée pour les spécifications dans le domaine de l'ingénierie logicielle. UML propose un ensemble de constructions prédéfinies, extensibles. Cette notation permet de s'associer avec de nombreux outils standards. Bien que la notation ne soit pas initialement prévue pour la description d'architectures, une architecture peut se décrire facilement en termes de diagrammes de classes, d'objets, de séquences, de collaboration, d'activités ou encore de graphes d'états [MRRR02].

Il existe de nombreux langages et environnements qui utilisent un ADL, par exemple Aster [IB96], Darwin [MK95], Olan [BABR96] ou ODAC [GM01]. La méthodologie ODAC utilise un ADL fondé sur la notation UML et s'appuie sur le standard ODP afin de fournir un ensemble de méthodes et d'outils pour la construction de systèmes à base d'agents. Des ADL sont plus spécifiquement dédiés à la construction de systèmes distribués. Ainsi, l'approche suivie par l'environnement Aster [IB96] consiste à s'appuyer sur les idées du domaine des architectures de logiciels afin de faciliter la construction d'applications distribuées sur un intergiciel donné et à spécialiser le système associé. Initialement conçu pour les applications devant s'exécuter sur une plate-forme CORBA, l'environnement Aster supporte conceptuellement des intergiciels tels que EJB ou DCOM [Zar00]. L'environnement Aster comprend :

1. l'ADL Aster pour la description d'applications en terme d'interconnexion de composants logiciels. Il permet d'associer des propriétés de qualité de service aux composants et utilise la syntaxe de l'IDL CORBA2,
2. un ensemble d'outils logiciels d'aide à la construction de systèmes distribués, adaptés aux applications, à partir de la description Aster de ces applications et en particulier des propriétés de qualité de service spécifiées.

L'atout de l'environnement Aster par rapport à d'autres environnements est de permettre la construction automatique (par spécialisation) d'un système d'exécution (pour le service complexe) implantant des mécanismes (p.ex. tolérance aux fautes, contrôle d'accès) adaptés aux exigences non fonctionnelles exprimées par l'architecte. Les travaux menés jusqu'ici ont porté sur les politiques de sécurité [BI97], les mécanismes de tolérance aux fautes [SI99], le transactionnel [ZI98] ou encore les garanties temporelles [DAI99]. L'approche suivie consiste à associer des propriétés de qualité aux composants de l'application, à l'ORB (système de communication) et aux composants offerts par la couche intergicielle (services système) sur laquelle va être construit le système. La spécialisation du système initial avec des services système permet, autant que faire se peut, de garantir la qualité de service exigée par l'application.

Les capacités d'un ADL à décrire formellement les interactions correspondent généralement aux possibilités offertes par le langage de spécification sur lequel il est basé. Citons l'utilisation de CSP [AG94], de Z [AAG93], de la logique [MQR95], des grammaires de graphe [Le 96], du π-calcul [MK95], de Gamma [IW95] ou encore des ensembles partiellement ordonnés d'événements [LKA+95]. Ces formalismes n'adressent pas toujours les mêmes descriptions et se restreignent souvent à un sous-ensemble des besoins réels d'un architecte. Par exemple, la plupart des approches précédentes se concentrent sur les aspects fonctionnels de l'architecture. Il reste difficile de spécifier ou d'étudier des propriétés non fonctionnelles telles que la sécurité [Rou97], la performance ou encore la fiabilité dans un même cadre de spécification.

5.2.3 Extension du langage de description d'architectures Aster

Pour contrôler la complexité de la construction de services complexes, nous suivons une approche par ADL. Cette approche offre un ensemble de concepts rigoureux qui facilitent la réutilisation, les analyses et l'implantation d'applications. Nous avons choisi d'étendre l'ADL Aster pour spécifier un service complexe et la vue réseau. Le choix de ce langage est plus lié aux concepts et outils pour le développement du système distribué supportant un service complexe qu'à sa syntaxe. En plus de permettre l'implantation d'une application, une description dans le langage Aster peut être utilisée pour synthétiser de manière automatique une configuration intergicielle (schéma d'organisation client-serveur mis en œuvre par de l'invocation distante) qui garantit des exigences non fonctionnelles. Dans notre cadre, nous utilisons, en plus des invocations distantes, des interactions par agents mobiles. Pour notre domaine d'applications, le but à long terme est d'étendre les techniques Aster aux agents mobiles.

5.2.3.1 Spécification des interfaces

Les interfaces des services primitifs sont définies à part (réutilisation) et contiennent les définitions des fonctions. Elles correspondent au prototype des fonctions mises en œuvre dans un fichier source, dans l'exemple ci-dessous, *tradPrototype* pour le service de presse personnalisé tiré de l'exemple 1 (cf. section 3.2.3, Figure 3.3, page 52). Nous y associons, en plus, les définitions non fonctionnelles de objets (cf. environnement e utilisé dans les analyses du chapitre précédent) dans le champ **properties**. Le nom du fichier source (champ **implementation**) est donné si le service primitif est interne à l'organisation.

interface trad {
  Ascii tradPrototype(d_1 : *Ascii*, d_2 : *Int*);
  properties
    performance $\lambda x.\ x$;
    confidentiality $\lambda x.$ public;
    integrity $\lambda x.$ sûr;
  implementation trad.java;};

L'interface d'un traitement suit le même format, avec en plus la taille du code source et ses niveaux de sécurité (cf. environnement e_c).

interface fusion {
  Ascii fusionPrototype(d_1 : *Ascii*, d_2 : *Ascii*);
  properties
    performance $\lambda(x, y).\ x + y$;

confidentiality $\lambda(x, y)$. min(x,y);
integrity $\lambda(x, y)$. non-sûr;
sourcesize 32Ko;
sourceconfidentiality public;
sourceintegrity sûr;
implementation fusion.java;};

Vu que pour les analyses les informations sur les données d'entrée (arité 0-aire) du service complexe sont nécessaires, nous les définissons de la même manière (types, taille et niveaux de sécurité), bien qu'elles ne soient pas directement utilisées pour la génération du code des interactions.

5.2.3.2 Spécification des composants

Dans notre cadre, un composant correspond à un site dans le sens des chapitres 3 et 4 et plus particulièrement à une place (terminologie des plates-formes à agents mobiles) pour notre prototype. La spécification d'un composant fournit le nom du composant et les identificateurs des services primitifs qu'il expose (champ **provides**). Les services primitifs sont bien entendu définis par leurs interfaces. La définition d'une interface inclue la définitions non fonctionnelles du service primitif. La présence d'un environnement d'exécution d'agents sur le site qui héberge le composant peut être stipulée à ce niveau (l'exécution d'agents mobiles nécessite un environnement spécifique). Les attributs non fonctionnels du composant sont intégrés dans le champ **properties**[3]. Si le composant a déjà été déployé (dans ce cas, les trois champs non fonctionnels sont complets), l'adresse de celui-ci (localisation) sur le réseau est indiquée (champ **address** avec ici l'adresse IP et le port). L'exemple ci-dessous spécifie un composant, hôte du service primitif *trad* et d'un autre service de comptage de mots. Ce site est de niveau *public* pour la confidentialité et *sûr* pour l'intégrité (cf. section 4.1.2.1). Il propose un mécanisme de tolérance aux fautes. Le service *trad* peut être répliqué et ce de manière invisible pour un client.

component SiteTraduction {
provides trad, wc;
receiveMobileAgents non;
properties
confidentiality public;
integrity sûr;
reliability fiable;
address 194.2.94.60:7000;}

De manière similaire, un composant client (site à partir duquel le service complexe sera accessible) peut être défini avec les identificateurs d'interfaces de traitement et son adresse. Il ne requière pas de propriétés de qualité.

La spécification abstraite d'un connecteur du réseau s'apparente à celle d'un composant. Le champ **properties** comporte seulement des informations sur la bande passante et sur la

3. IBM a proposé récemment le langage WSEL (*Web Services Endpoint Language*), qui permet de décrire, à l'aide d'une notation XML, les caractéristiques non opérationnelles des services, telles que les propriétés de qualité.

sécurité associée. Sur l'exemple ci-dessous, le connecteur garantit à la fois la non-divulgation et non-modification des informations qui y circulent.

connector ConnecteurLan {
properties
performance α;
confidentiality secret;
integrity sûr;}

Pour un composant comme pour un connecteur, quand un des champs **properties** n'est pas défini, c'est qu'il n'a pas été déployé et par défaut le niveau minimal est considéré (cf. propriétés des sites et liens dans la vue réseau). Dans ce cas, si utilisé dans la configuration, il est nécessaire de déterminer un tel niveau, qui pourra requérir un mécanisme voué à le garantir/optimiser.

5.2.3.3 Spécification de la vue réseau

La spécification de la vue réseau, en plus d'utiliser les différentes instances de composants et connecteurs, utilise un champ **bindings** pour décrire les liaisons du réseau sous-jacent. Un exemple de vue réseau sera donné par la suite. À ce stade de spécification, quelques analyses simples (p.ex. typage) permettent de vérifier une certaine cohérence entre la vue réseau et la vue du service complexe (à la fois fonctionnelle et non fonctionnelle). Par exemple, un service primitif s_i ne peut assurer un niveau de sécurité élevé si le composant sur lequel il est implanté est défini comme totalement non sécurisé.

network Nom {
instances < liste des sites et liens >
bindings < liaisons logiques entre les sites et liens >}

5.2.3.4 Spécification de la configuration

La configuration, dite abstraite, contient l'expression fonctionnelle du service complexe (champ **expression**), associée aux propriétés non fonctionnelles que l'architecte souhaite voir vérifiées par le service complexe à construire. Optionnellement, la spécification de la mise en œuvre du service complexe peut être donnée à travers le champ **bindings**. Elle correspond à une expression concrète. Dans ce cas, les nouvelles interfaces des versions CPS sont produites automatiquement.

configuration Nom {
expression < expression abstraite >
properties < exigences non fonctionnelle sur le service complexe >
performance critère; **confidentiality** niveau; **integrity** niveau; **reliability** niveau;
bindings < expression concrète représentative des connexions >}

Une description de configuration est définie en terme d'interconnexion d'interfaces *via* la liaison entre entrées et sorties. Le champ **bindings** permet de spécifier une configuration concrète qui fait apparaître les différents mécanismes d'interaction employés (imbrication de constructions **let**) entre les différentes interfaces de services primitifs. Une invocation distante représente un protocole requête/résultat reliant deux interfaces, alors qu'un agent mobile, vu

également comme une interaction, effectue la liaison ordonnée entre plusieurs interfaces. Ainsi, il est possible de voir une expression concrète au sein de l'ADL comme une liste de connecteurs logiciels entre interfaces. Un connecteur logiciel agent mobile est orienté, les migrations étant explicites dans une expression concrète. À notre connaissance, il n'y a pas d'autres propositions qui considèrent un agent mobile comme un connecteur de l'architecture logicielle.

5.3 Instanciation du cadre dans un prototype

Nous avons présenté dans la section 5.1 un cadre général qui couvre l'ensemble des analyses utiles au concepteur et les différents entrelacements possibles de celles-ci ainsi que différents moyens pour le développement de parties de l'environnement d'exécution (système d'exécution et système de communication) garantissant la qualité exigée par l'application. Afin d'illustrer plus clairement différentes étapes pour la construction d'un service complexe au regard de propriétés de qualité, nous choisissons de nous concentrer sur une instanciation possible de ce cadre général.

L'agencement des analyses proposé permet d'appliquer les quatre analyses pour les trois propriétés de qualité. D'autre choix sont possibles (agencement différent, moins d'analyses, moins de propriétés). Les quatre analyses existantes sont utilisées en séquence sur une unique propriété (une propriété par analyse) alors qu'il pourrait être plus judicieux de combiner plusieurs propriétés au sein d'une même analyse. Toutefois, notre but est ici d'illustrer l'applicabilité des différentes analyses. L'agencement présenté permet de se donner une idée des atouts de l'environnement de construction et nous a conduit à proposer un premier prototype voué à automatiser le procédé de construction. L'instanciation proposée identifie clairement les tâches et outils des acteurs. Nous supposons ici que les sites des services permettent l'exécution d'agents, bien que dans les réseaux grande échelle, des environnements d'exécution d'agents ouverts soient très loin d'être légion (p.ex. services Web). Le procédé d'ingénierie est schématisé dans la Figure 5.3. Les actions de chaque boîte, les données et résultats seront explicités par la suite.

Toutes les étapes présentées dans la figure n'ont pas été mises en œuvre. Nous avons toutefois développé un premier prototype qui permet de construire, de manière systématique, des services distribués complexes à partir de leur spécification abstraite (conception et implantation sur un système déjà déployé). Ce prototype intègre juste quelques analyses pertinentes, afin de valider leur faisabilité effective. Il traite notamment de la synthèse d'une description de mise en œuvre, analyse particulièrement délicate en terme de complexité quand la performance est prise en compte. Les analyses telles que la vérification de la qualité de mise en œuvre ou la synthèse de traitement n'ont pas été mises en œuvre car elles sont de complexité linéaire et sans réels intérêts techniques. L'analyse de synthèse d'architecture et plus particulièrement son association avec l'étape de déploiement est particulièrement délicate. Nous proposons ici une association conceptuelle de l'environnement de construction avec les outils Aster pour spécialiser l'environnement d'exécution basé sur un ORB conforme au standard CORBA. Envers le critère de fiabilité, cette association nécessite d'étendre la systématisation Aster aux agents mobiles pour la tolérance aux fautes. Le prototype que nous avons réalisé pour le développement de parties de l'environnement d'exécution se limite à l'utilisation d'un service de persistance sur les sites (configuration de l'environnement d'exécution). La plate-forme de mise en œuvre retenue est un intergiciel à agents mobiles.

Il existe désormais un nombre considérable de plates-formes à agents mobiles. Nous avons

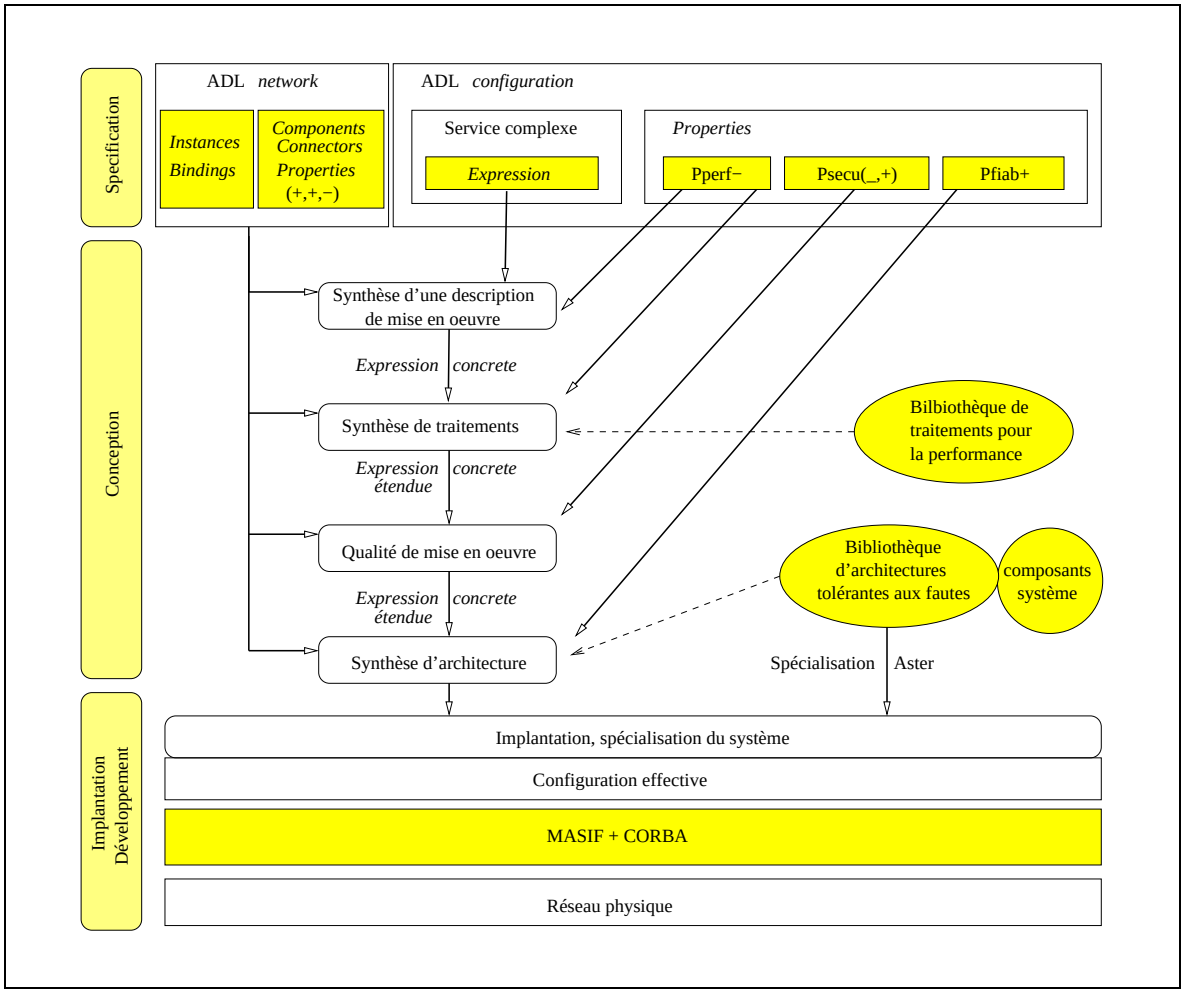

FIG. 5.3 – *Un agencement de l'environnement*

choisi de reposer sur Grasshopper [IKV] mais d'autres plates-formes auraient également pu être choisies (p.ex. voir [Ber99] pour une étude comparative). La plate-forme Grasshopper a été le premier système conforme au standard MASIF (cf. section 1.4.2.1). Grasshopper offre à la fois des interactions par invocations distantes et par agents mobiles. Son association avec le intergiciel CORBA garantit un ensemble conséquent de services intergiciels (cf. section 1.2.2).

Le prototype développé est présentée dans la Figure 5.4. Dans cet environnement, les étapes de conception sont automatisées. Les agents mobiles aptes à réaliser le service complexe sont générés automatiquement. Seuls l'architecte et le responsable de l'installation interviennent.

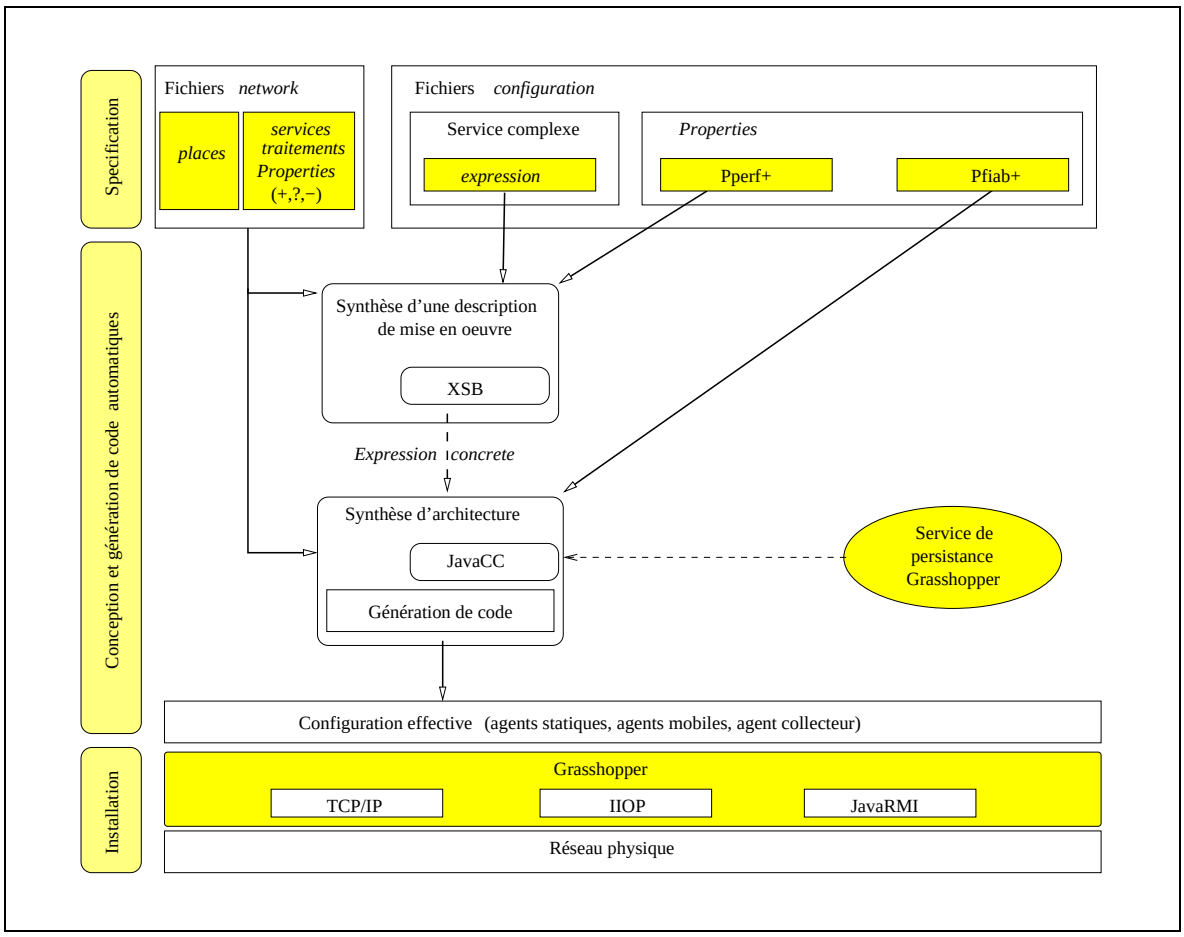

FIG. 5.4 – *Environnement prototype*

Après avoir présenté rapidement la plate-forme Grasshopper dans la section 5.3.1, nous détaillons les rôles spécifiques des différents acteurs pour l'agencement de l'environnement, à savoir la section 5.3.2 pour l'architecte, la section 5.3.3 pour le concepteur et la section 5.3.4 pour l'équipe de développement. Pour chacune des ces sections nous présentons les éléments du prototype qui ont été mis en œuvre. Finalement dans la section 5.3.5, nous discutons de l'instanciation proposée et de quelques points qu'il serait intéressant d'approfondir avec plus de temps et d'efforts de mise en œuvre.

5.3.1 Survol de la plate-forme Grasshopper

La plate-forme Grasshopper [IKV] est un environnement de développement et d'exécution pour agents mobiles. Elle est mise en œuvre en Java et supporte la migration d'agents (mobilité faible, cf. section 1.4.1) ainsi que les invocations distantes de manière transparente. Cette plate-forme permet donc l'implantation d'applications qui combinent les avantages des deux technologies. Plusieurs protocoles de transport (système de communication) pour les agents sont supportés (c.-à-d. TCP/IP, JavaRMI, CORBAIIOP, MAFIIOP, TCP/IPSSL, RMISSL) et la plate-forme peut facilement être étendue en accédant à d'autres applications basées sur un même ORB.

Les agents dans Grasshopper (stationnaires ou mobiles) sont vus comme des services. La classe agent peut être dérivée de la classe *StationaryAgent* ou *MobileAgent*, fille de la superclasse *Service*. Ainsi, nos services primitifs sont mis en œuvre par des agents stationnaires Grasshopper. Un agent peut invoquer des méthodes sur d'autres agents (c.-à-d. services statiques en ce qui nous concerne)). L'agent n'a pas besoin de s'occuper de la localisation du service invoqué (transparence de la localisation). Ainsi, dans le code d'un agent, il n'y a pas de différence entre une invocation de méthode distante et une invocation locale (agent mobile sur la même place que le service).

La Figure 5.5 présente une vue globale d'un environnement Grasshopper. Un agent est localisé dans une place (une machine virtuelle) (cf. section 1.4.2.1) qui assure l'exécution. Les agents peuvent communiquer entre eux au sein d'une place, avec l'extérieur en utilisant le service intergiciel de communication de l'agence ou encore utiliser des services ou ressources locaux du site. Une place est incluse dans une agence qui permet le contrôle, le transport et l'enregistrement des agents. La sécurité des communications (c.-à-d. SSL) est prise en charge par l'agence.

Grasshopper permet d'utiliser un service intergiciel de persistance sur chacune de ses agences (sauvegarde des places et agents). Ce service sera utilisé dans le prototype pour optimiser la propriété de fiabilité suite à la synthèse d'architecture. Ce service d'objets persistants (API) fournit au système hôte une interface pour conserver et gérer l'état des objets. Les informations d'exécution des agents sont préservées afin qu'ils puissent continuer leurs tâches en cas d'arrêts imprévus du système (c.-à-d. résistance aux pannes). Les états des objets sont écrits dans un fichier local du site hôte. Il contient, en plus des sauvegardes d'agents, les informations et états d'exécution de toutes les places qui existent au sein de l'agence persistante. Une agence sauvegarde périodiquement (laps de temps déterminé au déploiement) ses places et les agents qui s'y exécutent. En cas d'arrêt du système, l'agence est redéployée et tous les agents sauvés ainsi que les places sont automatiquement redémarrés. Le fichier de sauvegarde est utilisé pour reconstruire dynamiquement les agents. L'instance d'un agent est recréée avec son état et le processus (*thread*) correspondant est relancé. L'agent redémarre son exécution au point de programme sauvegardé qui précède son interruption. Ce service de persistance réduit considérablement les performances temporelles du système d'exécution.

Une agence peut utiliser un module MAF (*Mobile Agent Facility*) pour communiquer avec d'autres agences conformes avec le standard MASIF. Les agences peuvent être regroupées au sein d'un même domaine (*region*) pour faciliter les opérations d'administration. Ce domaine propose, entre autre, un service intergiciel d'enregistrement qui permet de connaître les places et agents présents au sein du domaine. Ce service est lié aux différents services d'enregistrements des places actives.

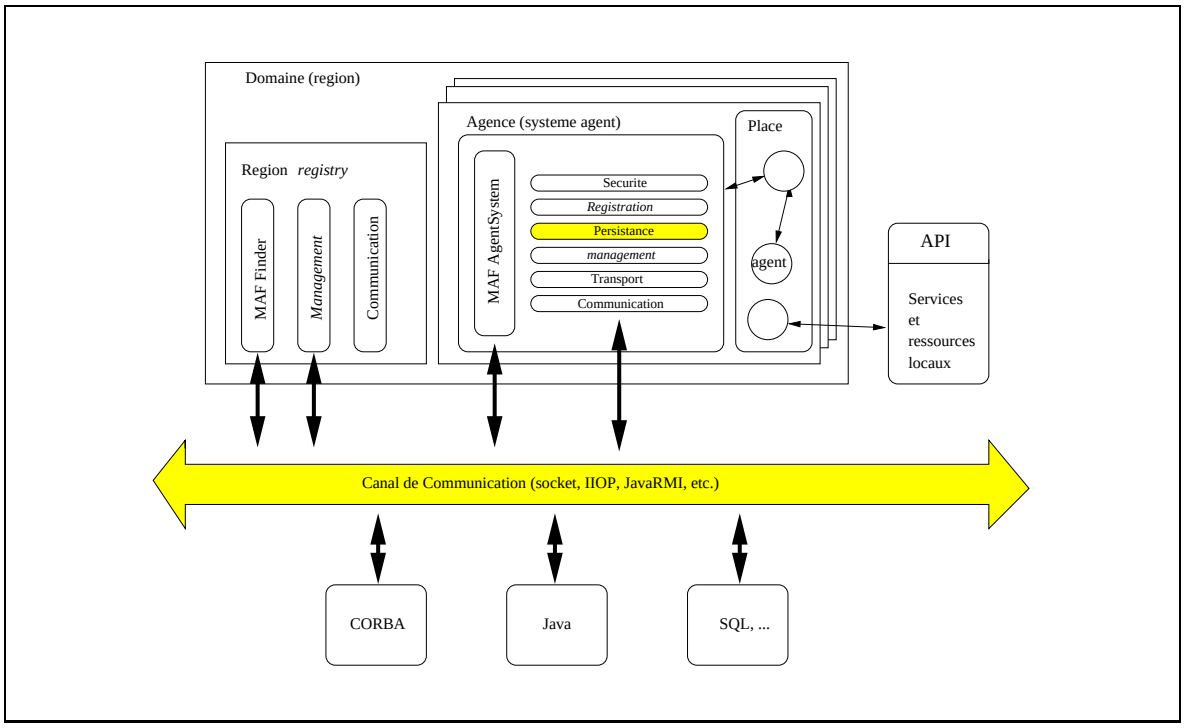

FIG. 5.5 – *Structure d'un environnement d'exécution basé Grasshopper*

5.3.2 Rôle de l'architecte

L'architecte dispose de services primitifs accessibles à travers son réseau. Il en connaît les interfaces et leurs localisations. À l'aide de l'ADL (champ **network**), il fournit la vue réseau (cf. ADL *network* sur la Figure 5.3, page 118) où la localisation des composants et services est entièrement définie, ainsi que les liaisons des composants (champ **bindings**). Il spécifie également la vue fonctionnelle de son service complexe par une expression abstraite (champ **expression**) dans l'ADL *configuration*). Il y exprime ses exigences non fonctionnelles (champ **properties**). Pour l'agencement des analyses proposé, il exige la performance minimale ($P_{perf\_}$), la garantie de la sécurité des données et traitements ($P_{secu(\_,+)}$) et l'optimisation de la propriété de fiabilité (P_{fiab+}). Pour la sécurité, il ne s'intéresse qu'à l'intégrité. Ainsi, le niveau public est suffisant pour la confidentialité. La vue réseau, complète pour les localisations et les propriétés de performance et de sécurité est toutefois incomplète pour la propriété de fiabilité $(+,+,-)$. Des composants et connecteurs utilisés dans l'ADL *network* n'ont donc pas toujours de propriétés de fiabilité associées.

Après avoir effectué des vérifications (automatisables) sur ses différentes spécifications (p.ex. vérification de type des interfaces, composition valide des interfaces), l'architecte les soumet en données d'entrée de l'étape de conception.

Exemple

Afin de proposer un exemple de spécification, nous reprenons ici comme service complexe l'exemple 1 de presse personnalisé (cf. section 3.2.3, Figure 3.3, page 52) défini abstraitement

par :

$$\mathsf{fusion}(\mathsf{select}(journal\ d_1), trad(actu\ d_2, d_3))$$

Ce service complexe est à construire sur un réseau dont le site proposant le service de traduction est externe et ceux proposant les services *journal* et *actu* appartiennent à l'organisation. Ce service complexe devra être accessible aux clients à partir d'un site que nous appelons *client*. Le site supportant le service *journal* est externe au site client alors que celui supportant *actu* appartient au réseau local et est déconnecté de l'extérieur (nous supposons que les liaisons sont bidirectionnelles). La vue réseau, ainsi que le service complexe, peuvent être spécifiés comme suit :

network VueReseau {
instances C:Client; A:Actu; J:Journal; T:Trad;
C1:ConnecteurLan; C2:ConnecteurWan;
bindings C **to** A **through** C1; C **to** J **through** C2; C **to** T **through** C2;
J **to** T **through** C2;}
configuration ServicePresse {
expression
$C.\mathsf{fusion}(C.\mathsf{select}(J.journal\ C.d_1), T.trad(A.actu\ C.d_2, C.d_3))$;
properties
performance min; **confidentiality** _; **integrity** sûr; **reliability** fiable;}

Les définitions des composants et connecteurs correspondent à celles proposées dans la section 5.2.3.2, où la propriété **reliability** n'est pas définie pour les composants A et J (ils n'ont pas encore été déployés et il est encore possible de leur associer un mécanisme de tolérance aux fautes).

Prototype

L'intégration avec le langage de description d'architectures Aster reste conceptuelle au stade actuel du prototype. Nous nous sommes contentés d'utiliser des fichiers qui contiennent les différentes informations dans un format fixé (XML aurait pu être utilisé). Les fichiers en entrée comprennent :

- l'expression abstraite représentant le service complexe. Chaque identificateur y est unique,
- les adresses de chacunes des places Grasshopper (cf. terminologie page 17) où se trouvent les services primitifs et le client,
- les définitions des services primitifs présents sur ces places (p.ex. adresse, interfaces fonctionnelles, définitions non fonctionnelles, type des paramètres et résultats, nom de la méthode qui met en œuvre le service primitif),
- les codes sources correspondant aux traitements,
- les données fournies par le client.

Par hypothèse, nous considérons que les spécifications fournies sont cohérentes (p.ex. composition correcte des fonctions). Un analyseur syntaxique est utilisé (écrit en XSB [War]) pour vérifier la conformité de l'expression abstraite par rapport à la grammaire du langage. Le langage choisi se restreint aux constructions de base, tel que présenté dans la section 3.2.1, page 49. L'analyseur est basé sur la notation DCG (*Definite Clause Grammar*) et sert de *frontend* pour synthétiser certains attributs formatés dans des fichiers (p.ex. liste des identificateurs, nombre de fonctions, données arguments d'une fonction, dépendances entre fonctions, ordre des fonctions). Ces attributs seront utilisés par la suite pour la mise en œuvre de la synthèse d'une expression concrète et la génération automatique du code des interactions.

5.3.3 Rôle du concepteur

Le concepteur utilise en séquence quatre analyses (cf. boîtes aux bords arrondis de la Figure 5.3) pour le guider et lui permettre de proposer une configuration effective s'il en existe une. Nous décrivons ci-après ces quatre analyses qui reposent sur les spécifications de l'architecte, notamment la vue réseau.

5.3.3.1 Synthèse d'une description de mise en œuvre par raffinement suivant la performance

La vue réseau est connue ainsi que les propriétés de performance associées à ses éléments. Ici, la propriété traitée est la performance qui est requise minimale. Il est possible de combiner la recherche de l'expression «optimale» en terme de trafics générés avec les besoins en sécurité et fiabilité. Toutefois, dans le prototype, nous traitons la performance, indépendamment des deux autres critères. Au vu de la complexité de l'analyse, dans le cas où il serait possible de prendre en compte les autres critères de qualité de manière combinée (c.-à-d. sécurité et fiabilité), des services complexes plus conséquents pourraient être facilement traités. La synthèse d'une description de mise en œuvre minimale (voir section 4.2.1) permet d'obtenir, à partir de la description abstraite du service complexe, une spécification d'une mise en œuvre qui minimise les volumes de données échangées. Elle permet de dériver automatiquement une description de mise en œuvre minimale pour des services complexes constitués de quelques dizaines de services primitifs. Le concepteur utilise cette analyse de manière entièrement systématique ou bien comme une aide afin de guider certains choix de mises en œuvre. L'analyse génère une expression concrète qui combine éventuellement des interactions locales, des invocations distantes et des agents mobiles. Les interfaces des fonctions CPS produites dans l'expression concrète sont ensuite simplement générées et introduites dans les spécifications pour les analyses suivantes.

Exemple

Sur l'exemple spécifié précédemment, nous pouvons imaginer que la meilleure expression concrète correspond à un agent mobile pour interagir avec le service *journal*, appartenant à l'organisation et dont le site permet un traitement déporté (ici select). L'interaction avec le service *actu* se fait par invocation distante sur le réseau local. L'interaction avec le service externe de traduction se fait également par invocation distante. Finalement, le traitement fusion se fait en local, sur le site à partir d'où est construit le service complexe. La configuration est donc étendue pour intégrer les différentes liaisons entre les interfaces (champ **bindings**),

sous la forme d'une expression concrète :

configuration ServicePresse {
expression
$C.\mathsf{fusion}(C.\mathsf{select}(J.journal\ C.d_1), T.trad(A.actu\ C.d_2, C.d_3))$;
properties
performance min; **confidentiality** _; **integrity** sûr; **reliability** fiable;
bindings
let $r_1 = (go_{C,J}; J.\overline{journal}; C.\overline{\mathsf{select}}; go_{J,C}; end)\ \ C.d_1$ **in**
let $r_2 = (go_{C,A}; A.\overline{actu}; go_{A,C}; end)\ \ C.d_2$ **in**
let $r_3 = (go_{C,T}; T.\overline{trad}; go_{T,C}; end)\ \ (r_2, C.d_3)$ **in**
let $r_4 = (C.\overline{\mathsf{fusion}}; end)\ \ (r_1, r_3)$ **in** r_4}

Prototype

Pour la mise en œuvre de la synthèse d'une description d'expression concrète, une première piste qui vient à l'esprit pour rechercher une expression concrète «optimale» est de s'intéresser au problème des plus courts chemins dans un graphe orienté pondéré (taille de données en ce qui nous concerne). En effet, rechercher la meilleure expression concrète impose d'étudier les différents itinéraires des agents mobiles. Ces itinéraires sont représentables à l'aide d'un graphe, avec les volumes de données associés aux arcs. Les algorithmes classiques de recherche du plus court chemin exploitent la propriété qui veut qu'un plus court chemin entre deux sommets soit lui-même composé de plus courts chemins [CLR96] (sous-structure optimale à l'intérieur d'une solution optimale). Cette propriété permet d'envisager aussi bien une technique de programmation dynamique qu'une méthode gloutonne (c.-à-d. complexité polynomiale). Or cette propriété n'est pas toujours vérifiée dans notre cadre. Nous ne pouvons pas construire le meilleur chemin «à la volée».

Le problème de trouver une expression minimisant les coûts est intrinsèquement global. Un début d'interaction par un agent mobile peut être optimal localement et devenir le pire par la suite. Sans prendre en compte les coefficients de bande passante et en ne s'intéressant qu'aux transfert de données, ajouter une fonctionnalité dans une expression abstraite remet en cause l'intégralité de l'analyse. L'application d'une nouvelle fonction dans une continuation peut tout d'un coup littéralement faire exploser le sac à dos, alors qu'un autre itinéraire, localement moins bon, peut être celui qui minimise les coûts sur l'itinéraire complet. Dans l'exemple de la Figure 5.6, le service complexe $Exp_{a1} \equiv s_2(s_1\ d_1, d_2)$ s'associe à une expression concrète optimale composée d'interactions par invocation distante. La donnée d_2 est supposée de taille importante (flèche large sur la figure) et la taille du résultat de s_2 encore plus importante (flèche très large sur la figure). Si l'on étend cette expression abstraite en une expression $Exp_{a2} \equiv t(s_2(s_1\ d_1, d_2))$ utilisant en plus un traitement t faisant décroître la taille du résultat de s_2, une interaction par agent mobile devient alors optimale (la taille du code du traitement est supposée négligeable). Il n'est donc pas possible de profiter des analyses d'une sous-expression.

Si l'on souhaite une analyse exacte, il est nécessaire d'étudier toutes les combinaisons d'interactions et les itinéraires d'agents qui mettent en œuvre un service complexe, de manière exhaustive. Toutefois, en cours de conception des interactions, il est possible de partager certaines d'entre elles. Par exemple, quand différents itinéraires d'agents réalisent un même sous-service complexe, le reste du service complexe à réaliser par les agents ne remet pas en

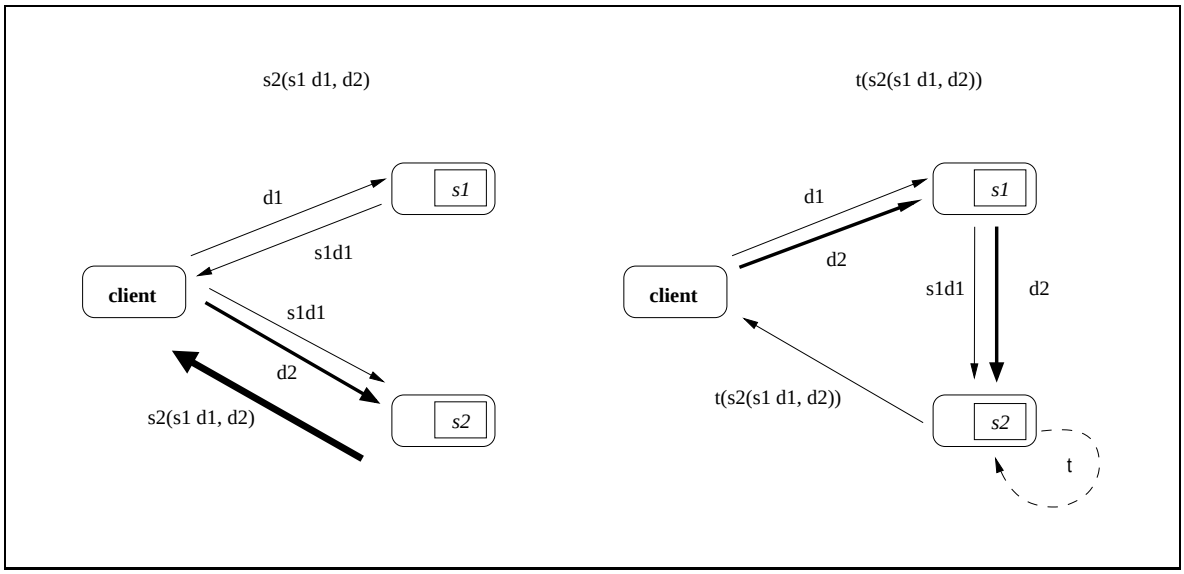

FIG. 5.6 – *Besoin d'un coût global*

cause l'agent optimal du sous-service si ces différents agents arrivent sur un même site. La Figure 5.7 propose cinq schémas d'interactions pour le sous-service complexe $(s_1\ d_1, s_2\ d_2)$, les trois débuts d'interactions du haut de la figure peuvent être partagés en fonction du service complexe $s_3(s_1\ d_1, s_2\ d_2)$. Le premier schéma comporte deux agents réalisant une invocation distante et les deux suivants un agent itinérant. Ces trois alternatives réalisent le même sous-service complexe avec retour du résultat chez le client. Il est possible de conserver la solution optimale des trois pour finaliser le service complexe. Les deux autres schémas du bas de la figure considèrent deux itinéraires différents d'agents pour réaliser l'intégralité du service complexe. Il y a également possibilité de partager sur le dernier site.

Pour profiter au mieux de ces types de partage sur les coûts d'interactions, nous avons choisi de produire l'équivalent d'un graphe d'états. Nous avons mis en œuvre un algorithme en XSB [War], langage qui permet de traiter le partage grâce au mécanisme de tabulation. Un état est composé d'une expression abstraite qui contient les fonctions restant à appliquer et l'identificateur du site courant de l'itinéraire. L'état initial comprend l'expression abstraite du service complexe où les données sont sous forme évaluée et le site client. L'état final comprend une valeur (forme évaluée de l'expression abstraite) et le site client. L'application d'une fonction d'un service complexe génère des transitions dans le graphe d'états. Pour un service s, une transition correspond soit à la continuation de l'agent courant (création d'un nouvel état où s est appliqué et où le site est mis à jour), soit au retour de l'agent chez le client pour initier un nouvel agent devant interagir avec s (nouvel état où le site devient client). Nous avons considéré qu'un traitement t dans un agent mobile est appliqué sur le site de sa dernière invocation de service ou bien sur le site du client (les traitements réalisés n'importe où sur le réseau ne sont pas traités). Un premier prédicat $trans(+Etat_1, -Etat_2)$ permet de produire, de manière non déterministe, les états successeurs $Etat_2$ d'un état $Etat_1$ donné. Ce prédicat synthétise, pour chaque transition et en plus de l'état $Etat_2$, la valeur du résultat transmis par la fonction précédemment appliquée ainsi que la valeur des arguments de la fonction qui va être appliquée.

Le prédicat *trans* est utilisé par un prédicat récursif non déterministe $cost(+Etat, -Cout)$

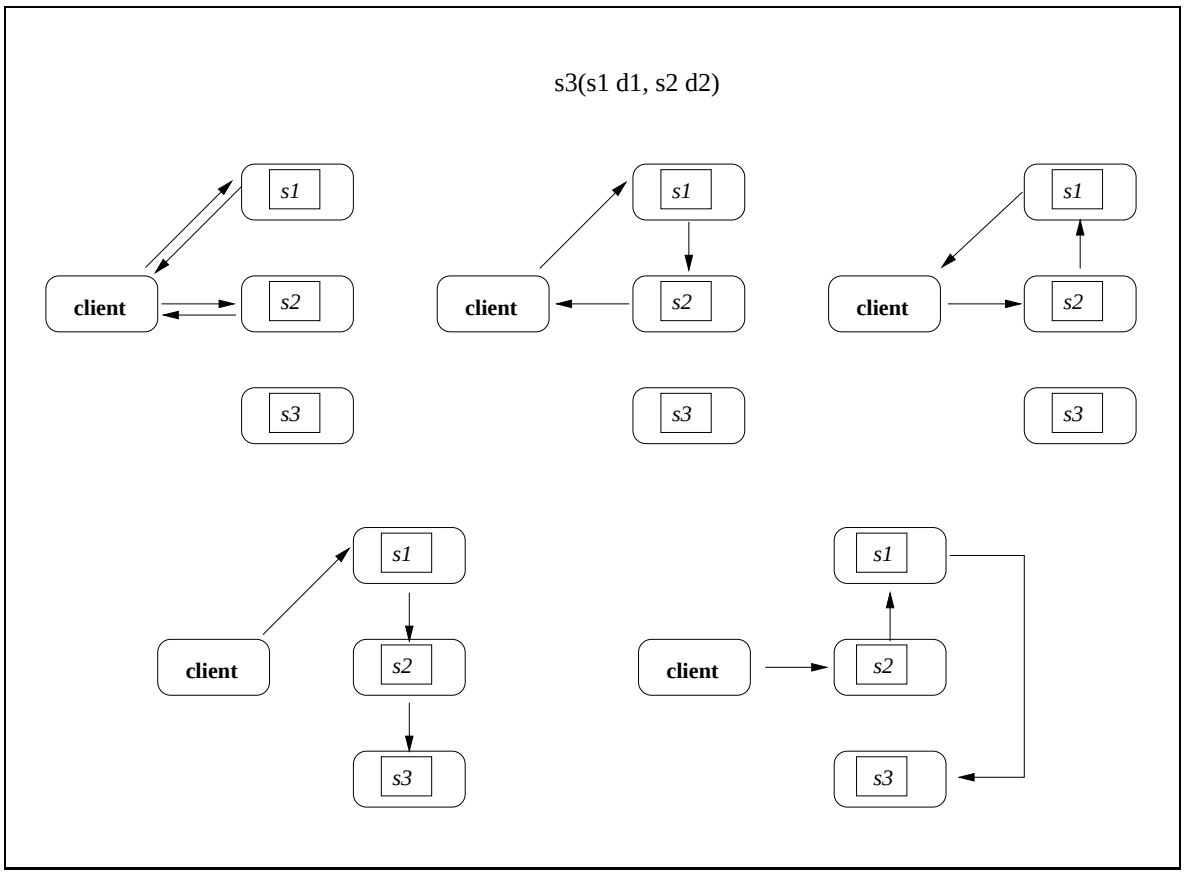

FIG. 5.7 – *Exemples de partage*

qui calcule le coût global des interactions réalisant un service complexe à partir d'un état. Pour les différentes descriptions de mises en œuvre d'un service complexe, en suivant une construction de l'espace d'états en largeur d'abord, nous pouvons comparer au fur et à mesure les états qui définissent des interactions qui sont sur un même site et qui ont les mêmes fonctions appliquées, tout en ayant la même continuation. Pour chaque transition, des états équivalents peuvent donc être partagés, réduisant ainsi la complexité du graphe. Le prédicat *cost* calcule la valeur du sac à dos (c.-à-d. taille des données et des traitements) et la valeur du coût courant à travers les transitions qui mènent de l'état initial à l'état final. Le coût induit par les transitions en cours est synthétisé à chaque étape et les états équivalents qui ont un coût identique pour un même sac à dos de données sont partagés. Les appels de prédicats sont tabulés et ainsi enregistrés dans une structure (c.-à-d. table) avec leurs résultats. Le partage est utilisé par le prédicat prédéfini *filterPO*, qui retourne le chemin minimal (meilleur coût pour réaliser le service complexe) suivant une relation d'ordre partielle, ici la relation inférieure sur les coûts.

Jeux d'essais

Nous avons réalisé quelques jeux d'essais à l'aide de cette mise en œuvre, restreinte au seul critère de performance. Le tableau de la Figure 5.8 présente quelques temps de résolution, sur un PC (800Mhz, 512Mo), pour différents types d'expressions abstraites.

Expression	Temps (sec.)
$f_n(f_{n-1}(f_{n-2}(....d)...))$	
n=10	0.02
n=50	0.2
n=150	34
n=300	187
$f(f_1\ d_1, ..., f_n\ d_n)$	
n=3	0.02
n=6	3
n=8	58
n=9	301
fusion(select(journal d1), trad(actu d2,d3))	0.02
s8(s1 d1,s6(s2 d2,s3 d3),s7(s4 d4,s5 d5))	1.6
s10(s7(s1 d1,s2 d2),s8(s3 d3,s4 d4),s9(s5 d5, s6 d6))	16

FIG. 5.8 – *Jeux de tests*

La recherche de la meilleure expression concrète sur un service complexe linéaire, du type $f_1(f_2(f_3(....d)...))$, est relativement efficace pour des expressions assez importantes en terme de fonctions utilisées. En effet, la complexité due aux parcours d'agents est réduite car la séquence d'application des fonctions est préfixée par l'expression abstraite. Le partage est donc maximal. Le cas des expressions constituées d'un n-uplet, c.-à-d. $f(f_1\ d_1, ..., f_n\ d_n)$, laisse apparaître la complexité exponentielle. En effet, dans ce cas, il est nécessaire de considérer tous les itinéraires d'agents qui permettent d'atteindre un même état. Le partage n'est possible que sur des agents qui arrivent sur un même site avec les mêmes fonctions appliquées et ayant à appliquer ensuite les mêmes fonctions. Bien que pour ce type d'expressions l'approche puisse sembler limitée, l'analyse d'expressions mixtes, comme celles proposées dans les exemples du chapitre 3 (cf.

Exemple 1), est viable. Cette première mise en œuvre réalisée dans un langage interprété reste de plus peu optimisée et pourrait être associée à des heuristiques admissibles (p.ex. traitement linéaire fortement décroissant sur ses entrées, toujours à déporter). Par ailleurs, rappelons que nous avons choisi ici de trouver la meilleure solution (et non pas une solution approchée) et de ne pas tirer profit des autres critères de qualité (qui peuvent permettre d'élaguer largement l'espace de recherche par simples tests booléens). L'adaptation de notre algorithme avec ces critères reste à réaliser.

5.3.3.2 Synthèse de traitements

Une fois une première proposition d'expression concrète dérivée, la synthèse de traitement suivant la performance permet de chercher à raffiner le service complexe (l'expression concrète) par l'utilisation de traitements spécifiques. Ces traitements peuvent faire décroître la taille des données devant transiter sur le réseau. Si ces traitements sont encapsulés dans des interactions par agents mobiles, il est clair que le concepteur doit s'assurer qu'un serveur censé accueillir de nouveaux traitements propose un environnement d'exécution pour agents. Il utilise pour cela les spécifications du réseau. La sélection de traitements peut se faire de manière systématique en connaissant les interfaces de traitements prédéfinis. L'algorithme de l'analyse a été présenté dans la section 4.3. Selon la nature des spécifications, rien n'empêche d'essayer d'appliquer plusieurs fois un traitement sur une même donnée. Nous convenons qu'un traitement de la bibliothèque ne s'applique qu'une seule fois et qu'il porte sur des données «brutes». Quand un traitement et son inverse sont utilisés pour étendre une expression concrète, les spécifications du service complexe sont mises à jour (dans l'expression abstraite et l'expression concrète). Les interfaces des traitements utilisés sont ensuite intégrées aux spécifications, et ce pour les deux propriétés de qualité que sont la performance et la sécurité.

Les traitements, provenant par exemple d'une API ou de codes programmeurs, sont définis par l'intermédiaire d'une bibliothèque (cf. ensemble sur la Figure 5.3, page 118), comme présenté dans la Figure 5.9. Les définitions comportent l'identificateur du traitement ainsi que les fonctions associées pour la performance et la sécurité. En plus de l'identificateur du traitement inverse, les niveaux de confidentialité et d'intégrité du code source sont également fournis.

Nom	Taille	DefPerf	DefSecu	SecuSource	t^{-1}
HuffmanCode	8Ko	$\lambda x.\ x/2$	$\lambda x.\ x$	public, sûr	HuffmanDecode
Bzip2	80Ko	$\lambda x.\ x/4$	$\lambda x.\ x$	public, sûr	Bunzip2
MyCompress	2Ko	$\lambda x.\ x*(3/4)$	$\lambda x.\ x$	secret, sûr	MyUncompress
HuffmanDecode	4Ko	$\lambda x.\ x*2$	$\lambda x.\ x$	public, sûr	-
Bunzip2	80Ko	$\lambda x.\ x*4$	$\lambda x.\ x$	public, sûr	-
MyUncompress	2Ko	$\lambda x.\ x*(4/3)$	$\lambda x.\ x$	secret, sûr	-

FIG. 5.9 – *Bibliothèque de traitements pour la performance*

Les extensions proposées par l'analyse peuvent conduire le concepteur à faire part à l'architecte de la possibilité d'intégrer directement un traitement sur un site de services primitifs. Par exemple, si un service complexe utilise un service primitif qui fournit toujours un résultat de taille très conséquente, il peut être judicieux d'implanter directement une compression sur le site correspondant. Le traitement peut être déployé sur un serveur à la manière d'un service primitif ou encore directement composé avec un service primitif (changement d'interface).

5.3.3.3 Qualité de mise en œuvre

À partir de l'expression concrète éventuellement étendue, le concepteur cherche à vérifier (a *posteriori*) que la mise en œuvre respecte les contraintes de sécurité exigées par l'architecte (P_{secu}). L'analyse s'appuie sur la vue réseau et ses attributs de sécurité complets pour vérifier l'intégrité des données et traitements du service complexe. L'analyse a été présentée dans le chapitre précédent (cf. section 4.1.2.2) et sa mise en œuvre suit très simplement la sémantique du langage concret. Si une des propriétés de sécurité n'est pas vérifiée au cour d'une interaction, l'analyse permet simplement de retrouver les objets qui violent la propriété (p.ex. données, traitements, sites, liens). Si l'analyse ne vérifie pas la sécurité, il devient nécessaire de modifier certains éléments. Dans ce cas, le concepteur peut être amené à :

- demander à l'architecte/administrateur la possibilité d'un changement de niveau de sécurité dans le réseau (p.ex. niveau d'un site ou d'un lien),
- tenter une synthèse de traitement sur l'expression concrète non valide suivant le critère de sécurité,
- considérer une autre expression concrète, soit en traitant l'expression concrète produite par la première analyse (dans la cas où elle aurait été étendue), soit en s'intéressant à une autre mise en œuvre possible du service complexe (pas la meilleure au sens de la synthèse d'une description de mise en œuvre).

Dans l'idéal, cette étape aurait du être traitée en même temps que les deux étapes précédentes (c.-à-d. synthèse de mise en œuvre suivant la performance et la sécurité puis synthèse de traitements suivant la performance respectant la sécurité) afin d'éviter les retours-arrières dans l'agencement. Toutefois, si cela avait été le cas, nous n'aurions pas pu intégrer et présenter indépendamment l'analyse de qualité de mise en œuvre dans l'agencement.

5.3.3.4 Synthèse d'architecture

L'expression concrète générée à l'aide des outils précédents représente une configuration (c.-à-d. liens logiciels entre interfaces) pour un service complexe donné sur un intergiciel qui supporte les invocations distantes et les agents mobiles. L'analyse de synthèse d'architecture suivant la fiabilité (propriété demandée par l'architecte) permet de compléter les éléments de la vue réseau (c.-à-d. composants) qui n'ont pas de propriété de fiabilité identifiée dans le champ **properties**. Tous les services sont localisés au sein de la représentation de l'architecture et l'analyse de synthèse d'architecture revient à l'étude des propriétés des sites incomplets (les liens ne sont pas considérés pour la fiabilité). Les sites qui ne sont pas encore déployés pourront profiter d'un mécanisme apte à améliorer la fiabilité si l'analyse le requiert (p.ex. configuration de l'environnement d'exécution à l'installation ou spécialisation du site comme un groupe de sites). L'analyse revient à une vérification de la qualité de mise en œuvre, où à chaque migration (c.-à-d. *go*), le site cible est étudié pour déterminer son niveau de fiabilité.

Pour une invocation distante, les mécanismes qui s'associent à l'environnement d'exécution afin qu'il puisse garantir la qualité exigée sont identifiés (cf. section 5.3.4 suivante) et inclus dans une bibliothèque d'architectures. Ces mécanismes utilisent par exemple des services intergiciels de transaction (p.ex. OTS) et de courtage pour la réplication primaire/secondaire. Pour un agent mobile, un même mécanisme doit être utilisé sur tout l'itinéraire. Les trois mécanismes présentés dans le chapitre 2 de la bibliographie (cf. section 2.3.3) sont également

identifiés au sein de la bibliothèque. Cette bibliothèque contient, pour chaque entrée, l'identifiant d'un mécanisme, la propriété qu'il traite au sein du domaine de valeur de la propriété de fiabilité, le type de mécanisme utilisé et enfin le squelette d'architecture qui permet de le construire systématiquement à l'aide des services de la couche intergicielle (ce point est étudié plus tard). Un exemple abstrait d'une telle bibliothèque est présenté dans la Figure 5.10 et correspond à l'ensemble grisé de la Figure 5.3, page 118.

Nom	Mécanisme	ConfigurationSpécialisée
Invocation distante		
N1	réplication active	ID1
N2	réplication semi-active	ID2
N3	réplication passive	ID3
Agent mobile		
Schneider	réplication active	AM1
Straßer	réplication semi-active	AM2
Silva	réplication semi-active	AM3

FIG. 5.10 – *Bibliothèque d'architectures pour optimiser la fiabilité*

À ce stade, le concepteur dispose d'une spécification complète du réseau, des propriétés de qualité et de la configuration du service complexe. Dans la spécification étendue du réseau, le concepteur a identifié les sites (internes à l'organisation) qui vont nécessiter d'intégrer des mécanismes spécifiques de tolérance aux fautes (p.ex. réplication). Il transmet toutes ces informations aux acteurs responsables de l'implantation du service complexe et du déploiement du système.

Exemple

Sur l'exemple du service de presse, les sites dont l'organisation a le contrôle sont ceux qui hébergent les services primitifs *journal* et *actu*. En supposant que l'architecte exige l'optimisation de la fiabilité de son service complexe, les deux interactions par invocation distante et celle par agent mobile de l'expression concrète doivent être fiable. Le site hôte du service *trad* est défini comme fiable (site déployé externe), selon la spécification du composant proposée dans la section 5.2.3.2. L'analyse de synthèse d'architecture impose donc l'intégration d'un mécanisme voué à optimiser la fiabilité sur les sites J et A. Le site A du réseau local est accédé par invocation distante (p.ex. RPC). Il va par exemple profiter de mécanismes de tolérance aux fautes existants pour ce type d'interactions. Le site J du réseau externe est accédé par un agent mobile. Il va par exemple profiter de mécanismes de tolérance aux fautes proposés dans la bibliographie (cf. section 2.3.3).

5.3.4 Rôles de l'équipe de développement

Dans l'instanciation du cadre, l'équipe de développement a en charge d'implanter le service complexe sur un environnement d'exécution à agents mobiles (Grasshopper), interopérable avec un intergiciel conforme au standard CORBA. Une telle plate-forme peut profiter des nombreux services intergiciels (représentés par des composants systèmes) offerts autour du système de communication CORBA (cf. section 1.2.2), tels que les services de sécurité, de transaction ou de courtage. L'ORB associé est chargé de gérer l'interconnexion des différents

composants du système. Dans le cas où le système offert par la plate-forme ne suffit pas à garantir la propriété de fiabilité sur un site, les acteurs du développement ont en charge de spécialiser ce système pour certaines interactions.

Quand le système de communication ne suffit pas à garantir une propriété, un système distribué implantant la propriété de fiabilité est construit par interconnexion de composants système à l'ORB utilisé pour l'interaction. Pour systématiser la spécialisation, l'équipe de développement du système utilise les outils proposés dans l'environnement Aster (au stade actuel, nous ne nous sommes intégrés que conceptuellement à ces outils). À partir d'une description de la configuration dans son ADL, Aster propose un ensemble d'outils pour systématiser l'intégration d'une application au sein d'une architecture distribuée suivant des contraintes non fonctionnelles imposées. Aster permet de construire des systèmes distribués implantant des mécanismes de tolérances aux fautes adaptés aux exigences du service complexe parmi l'ensemble de ceux disponibles dans les bibliothèques. La recherche des éléments dans les bibliothèques Aster se fait formellement par appariement de spécifications des propriétés de fiabilité. La spécification des propriétés de fiabilité se base sur les travaux de Saridakis (cf. section 2.3.1) qui a proposé une décomposition récursive de celles-ci (basée sur la logique des prédicats, étendue avec un opérateur de précédence [Sar99]). Cette décomposition permet d'identifier les propriétés qui caractérisent une technique de tolérance aux fautes et pour laquelle il existe des composants système primitifs les garantissant. Pour cela, l'outil utilise comme spécifications :

1. les éléments de notre service complexe, à savoir l'application cliente et les services primitifs,

2. les propriétés de l'ORB (système de communication) qui forme le système distribué de base. CORBA implante par défaut un appel synchrone avec une sémantique de défaillance « au plus une fois » et permet du « meilleur effort ».

3. les propriétés des composants système, issus de services intergiciels, qui implantent des fonctions de gestion de ressources distribuées. Par exemple, les opérations du service OTS (atomicité des défaillances garantie) ou du service de courtage (sélection dynamique pour la réplication primaire/secondaire d'un service) peuvent être utilisés pour spécialiser l'ORB.

La recherche des composants système nécessaires à la satisfaction de la propriété pour une configuration donnée peut être automatisée au moyen d'un outil s'appuyant sur un démonstrateur de théorèmes. Dans le cas où les composants système ne suffiraient pas à garantir une propriété exigée, il est à la charge du concepteur système de proposer des connecteurs spécialisés au sein de bibliothèques. Ces connecteurs spécialisés pourront ensuite être utilisés pour former une nouvelle entrée dans la bibliothèque d'architecture associée au intergiciel pour de futures spécialisations.

Les différents outils Aster pour spécialiser la couche méditiacielle s'appliquent aux interactions à base d'invocations distantes. À ce jour, les outils Aster ne traitent pas les interactions par agents mobiles. L'équipe de développement doit donc elle-même s'attacher à spécialiser la couche intergicielle, de manière non formelle et non systématique), à l'aide des composants système existant pour l'invocation distante et de services système qu'elle a pu développer. Toutefois, la cohabitation du standard MASIF avec CORBA laisse espérer que les interactions par agents mobiles sont spécialisables à la manière de Aster. En effet, les approches agents qui traitent de la réplication s'appuient sur des concepts communs à ceux des invocations distantes (réplication d'un service, transaction, vote, etc.). Il est donc vraisemblable

d'intégrer des mécanismes de tolérance aux fautes pour agents au sein des bibliothèques spécialisantes Aster. Toutefois, ce point technique n'a pas été poussé plus en avant. Pour cela, il est nécessaire de chercher à formaliser, à l'aide de la logique étendue utilisée dans Aster, les divers mécanismes existants de tolérance aux fautes pour agents. Nous avons brièvement étudié l'intégration formelle de ces mécanismes dans ce cadre où les agents peuvent être vus comme des objets de la logique. Toutefois, plus de temps serait nécessaire pour rendre l'approche effective/systématique et étendre les outils de sélection automatique (prouveur).

Une fois l'environnement d'exécution développé (si requis), adapté aux exigences de fiabilité exprimées par l'architecte, les interactions du service complexe sont implantées (mise en œuvre automatisée dans le prototype). Le code des interactions est produit, une par construction **let** du langage concret, qui vont réaliser le service complexe. Les appels de service par invocation distante sont traitées de manière classique (invocation transparente du talon de l'objet serveur). Pour les agents, les primitives de migration sont intégrées directement dans le code et les appels de services se font au travers des talons.

Prototype pour l'installation des sites au regard de la fiabilité

À défaut d'avoir intégré les mécanismes existants de tolérances aux fautes pour agents mobiles au sein d'une bibliothèque d'architectures (cf. approche Aster), nous proposons ici de nous appuyer sur un service intergiciel de persistance, apte à garantir la résistance aux pannes d'un service complexe. Nous ne développons donc pas un système spécialisé mais profitons d'un service intergiciel à l'installation (configuration de sites). La plate-forme Grasshopper permet d'associer un service de persistance aux agences. Pour activer cette fonctionnalité, une agence doit être installée de manière spécifique (paramètre). Dans une expression concrète, où tous les services primitifs sont placés, les sites qui nécessitent ce service de persistance sont identifiés lors de la synthèse d'architecture. Lors de la création d'une agence pour un site, le paramètre de persistance est donné si l'analyse de synthèse d'architecture l'a imposé, pour garantir l'exécution résistante aux pannes des agents en transit.

Les services primitifs sont ensuite installés dans une agence, éventuellement persistante. Un service primitif s_i est identifié par l'adresse de son agence et le nom de l'agent statique qui le représente. Les étapes à suivre pour implanter un service primitif sont les suivantes :

1. créer le code Java de l'agent service ($<$ *nomFichier.java* $>$), ainsi que le fichier contenant l'interface de cet agent, c.-à-d. les méthodes qu'il offre aux autres agents.

2. compiler cet agent service et son interface, générer les talons Grasshopper de l'interface de cet agent (fichiers *proxy*) et les compiler, puis installer l'agent.

Prototype pour l'implantation du service complexe

Nous avons développé un compilateur qui permet de produire des agents Grasshopper à partir d'une expression concrète. Il utilise le système JavaCC (*Java Compiler Compiler*) qui est un générateur d'analyseurs syntaxiques orienté vers le langage Java. Son association avec un préprocesseur JJTree permet de décorer automatiquement la grammaire concrète et engendrer la hiérarchie de classes réalisant les arbres syntaxiques correspondants. La Figure 5.11 représente la structure du compilateur. Dans la partie haute apparaissent les fichiers que le compilateur prend en entrée.

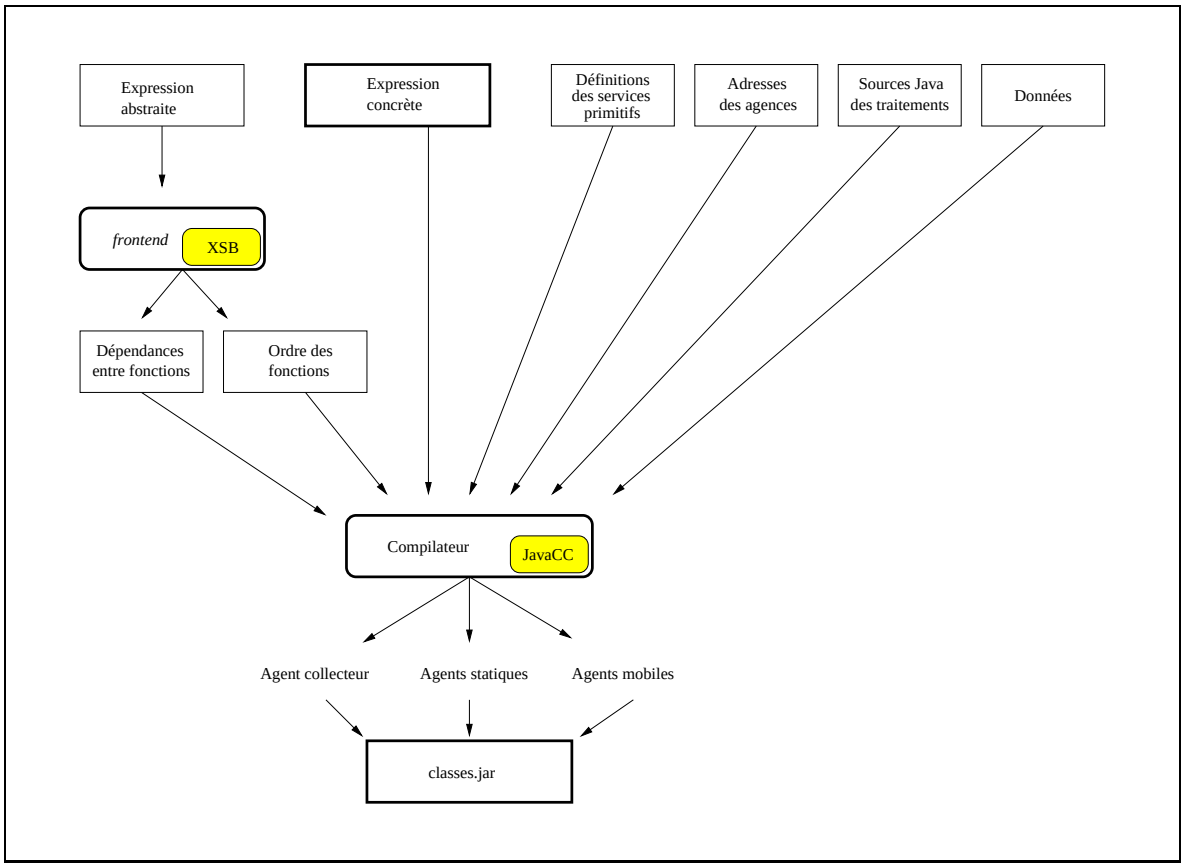

FIG. 5.11 – *Structure du compilateur d'expressions concrètes*

À partir de l'expression concrète, le compilateur crée les différents agents qui mettent en œuvre le service complexe. Les agents sont mobiles quand la construction **let** du langage concret représente une interaction par agent mobile (c.-à-d. plus d'un service appliqué ou au moins un traitement délocalisé). Ils sont stationnaires quand cette construction représente une invocation distante ou un traitement local (c.-à-d. un seul service appliqué ou pas de migration). Un agent stationnaire supplémentaire, l'agent collecteur, est créé sur le site client. Il a en charge de créer et initialiser les autres agents (un agent par construction **let**), de synchroniser l'arrivée de leurs résultats et de les détruire après exécution. Cet agent collecteur peut également être utile pour afficher graphiquement les résultats au client (p.ex. JavaSwing). Le compilateur possède deux modes d'utilisation :

1. automatique : les agents clients et leurs *proxy* sont compilés et placés dans un fichier `.jar`.

2. manuel : seuls les fichiers sources Java sont générés. Le responsable du déploiement peut alors spécialiser les codes (p.ex. traitements d'exceptions) avant de les compiler.

Finalement, après installation, soit le client lance son agent collecteur, soit un agent d'un client distant invoque le service (devenu primitif) réalisé par l'agent collecteur.

5.3.5 Discussion

Notre cadre général peut s'adapter conceptuellement à tout type de plates-formes qui offre des interactions par invocations distantes et par agents mobiles. Afin de présenter plus concrètement les concepts et outils de cet environnement de construction de services distribués complexes, nous avons proposé une illustration de son utilisation pour une plate-forme à agents mobiles interopérable avec un intergiciel CORBA. L'agencement des analyses proposé est un choix parmi d'autres. Le prototype présenté ne traite que deux analyses. Il permet toutefois de montrer la faisabilité de l'approche. Bien que restreint, il est utile au concepteur d'applications de services clients et d'autres analyses peuvent être intégrées. Nous avons traité les propriétés de qualité indépendamment, sans nous inquiéter des impacts d'un choix de mises en œuvre d'un critère de qualité envers les autres. Par exemple, dans le prototype présenté, le choix de l'expression concrète optimale en terme de performance peut être remis en cause suite à la synthèse de l'architecture quand il est nécessaire d'introduire un mécanisme de tolérances aux fautes, qui induit du trafic.

Malgré la complexité théorique de l'analyse de synthèse d'une description de mise en œuvre minimale, le prototype a montré qu'il reste possible de traiter systématiquement, et de manière exacte, des services complexes de taille raisonnable en terme de nombre de services primitifs utilisés. Vu que la grande majorité des exemples réels de services complexes composent peu de services primitifs, l'approche semble viable. Au-delà, l'intervention du concepteur devient nécessaire pour guider l'analyse afin d'écarter certains points de choix. L'expertise du concepteur peut le conduire à proposer des découpages particuliers aptes à contrer la complexité induite par les itinéraires d'agents ou encore utiliser certaines heuristiques. L'algorithme présenté se base sur le calcul de performance que nous avons donné en section 4.1.1.2. L'interprétation sémantique est liée à la plate-forme cible utilisée. Cette définition de l'analyse n'est pas entièrement exacte au regard de la plate-forme Grasshopper. Plus particulièrement, les données du client en Grasshopper sont les variables globales de l'agent. Elles suivent tout l'itinéraire de l'agent mobile et ne sont pas perdues suite à leur utilisation. Nos analyses de performance et

de sécurité sont donc inexactes pour la plate-forme Grasshopper, car nous considérons que les données sont détruites après utilisation. Toutefois, il serait simple d'adapter l'algorithme ou de modifier les interfaces des fonctions abstraites (passage en résultat de la donnée argument).

Pour la synthèse de l'architecture, nous n'avons pas traité directement de mécanismes de tolérance aux fautes. Nous avons utilisé le service de persistance offert par la plate-forme Grasshopper, qui est un mécanisme de résistance aux pannes. Toutefois, ce service intergiciel reste basique au regard de la propriété de fiabilité. Nous l'avons intégré au sein des agences (c.-à-d. sites) qui proposent les services primitifs sur le réseau. Les agents sont sauvegardés à intervalles réguliers sur les agences qu'ils visitent. L'arrêt d'une agence va conduire à relancer un agent seulement au point de sauvegarde précédent l'interruption. Entre cette sauvegarde et l'interruption effective, des calculs ont pu être perdus. L'atomicité des tâches d'exécution d'un agent n'est donc pas toujours garantie. Toutefois, dans Grasshopper, les agents mobiles peuvent intégrer directement dans leurs codes les instructions de sauvegarde. Il devient alors possible de chercher à spécialiser le code d'un agent mobile (et non pas étendre le service complexe), en proposant des sauvegardes aux points de contrôle stratégiques. Le problème revient à de la synthèse de traitement, comme ce qui a été présenté avec les mécanismes de chiffrement de code (cf. section 4.3.2.2, page 93).

L'utilisation d'une plate-forme compatible avec CORBA permet de s'associer conceptuellement à l'approche par spécialisation de la couche intergicielle offerte par l'environnement Aster. La plate-forme Grasshopper peut cohabiter avec l'intergiciel CORBA et donc profiter des différents services systèmes offerts au sein de l'ORB. L'utilisation effective des techniques Aster dans notre environnement aurait un double bénéfice. Premièrement, dans le cadre d'architectures de service client, Aster ne traite que les interactions de type invocation distante. Notre approche permet l'introduction des agents mobiles comme nouveau mécanisme d'interaction dans cet environnement. Deuxièmement, les techniques de déploiement envers des critères de qualité proposées dans les outils Aster, facilitent le développement systématique du système. Au sein du prototype, il serait souhaitable, avec plus de temps et d'efforts de mise en œuvre, d'étudier plus en avant l'intégration des mécanismes de tolérance aux fautes pour agents mobiles dans les bibliothèques d'architectures et les outils Aster.

Conclusion

Nous avons proposé dans ce document une approche à la construction de services distribués complexes dans le contexte des réseaux grande échelle. Pour répondre à ce problème d'ingénierie, nous avons défini un cadre de spécifications auquel s'associent des analyses intégrées dans un environnement. Le réseau Internet, en plus d'un espace de données gigantesque, tend à devenir un espace d'applications inter-opérables, disponibles sous la forme de services primitifs. Les architectures émergeantes orientées services de ce réseau utilisent principalement le schéma d'organisation client-serveur. Le schéma d'interaction associé repose traditionnellement sur les invocations distantes entre clients et prestataires de services (c.-à-d. serveurs). Toutefois, la prise en compte de propriétés de qualité (c.-à-d. performance, sécurité, sûreté de fonctionnement), pour le compte de clients, amène à considérer dans ces architectures l'utilisation d'interactions à base de code mobile (c.-à-d. évaluation distante, code à la demande, agents mobiles). Plus particulièrement, les agents mobiles ont un intérêt non négligeable pour respecter des contraintes de performance. Actuellement sur l'Internet, les environnements d'exécutions employés sur les serveurs ne permettent pas le transfert de code sous la forme d'agents mobiles. Il est à espérer que des prestataires de services chercheront à ouvrir leurs serveurs à l'accueil de tels agents.

Bilan

Nous avons basé notre description de services distribués complexes sur la composition et la spécialisation de services primitifs. Notre approche se fonde sur des spécifications abstraites qui utilisent les interfaces de services primitifs et de traitements de personnalisation. Elle nous a conduit à proposer deux langages déclaratifs : (i) un langage abstrait pour la spécification de services complexes, (ii) un langage concret pour la description des interactions entre clients et services dans un service complexe. Une expression dans le langage concret est fonctionnellement équivalente à une expression du langage abstrait. Le langage concret permet de représenter des interactions à base d'invocations distantes, d'évaluation distante et d'agents mobiles. L'invocation distante et l'évaluation distante y sont représentées comme des formes simplifiées d'agents. Grâce aux continuations, le langage concret permet de faire clairement apparaître le type des interactions et les itinéraires d'agents. Il nous a permis de conforter l'idée que les mises en œuvre de services complexes ont un intérêt à être hybrides, en permettant à la fois des interactions par invocation distante et par agents mobiles.

Dans les étapes de conceptions de services complexes, la qualité perçue par les clients doit être un souci de tous les instants. L'approche déclarative proposée permet des analyses simples de qualité. Pour cela, nous avons défini une analyse d'inférence de la qualité de mise en œuvre qui porte sur la performance, la confidentialité, l'intégrité et la fiabilité. Cette analyse

permet de comparer des mises en œuvre qui utilisent une combinaison d'interactions. Pour la performance, l'étude porte sur le volume total de données à faire transiter à travers le réseau pour réaliser un service complexe donné. En ce qui concerne la sécurité, notre analyse permet de déterminer si les interactions d'un service complexe préservent le secret et la non-modification des données et des codes des clients. Du point de vue de la fiabilité, elle permet de s'assurer qu'un service complexe est un minimum sûr de fonctionnement en présence de défaillances de sites et de canaux. Enfin, la capacité des agents à effectuer des traitements personnalisés et déportés sur des sites distants facilite la spécialisation de services. Une analyse de synthèse de traitements permet de profiter de cette flexibilité offerte.

Le cadre proposé nous a permis de dériver des outils, particulièrement utiles à la conception de services distribués complexes. La synthèse d'une description de mise en œuvre s'attache à déterminer une expression concrète « optimale » pour la mise en œuvre d'un service complexe. La synthèse d'architecture s'attache à faciliter la construction d'un système amené à supporter un service complexe donné. Elle traite du placement de services primitifs et des besoins en qualité que doivent garantir les éléments du réseau. Ces différents outils trouvent leur place dans un environnement de construction de services distribués complexes. Cet environnement, basé sur un langage de description d'architectures, guide le concepteur dans les choix d'interactions, le placement de services primitifs et l'intégration de mécanismes voués à garantir/optimiser une propriété de qualité donnée. De plus, son association avec l'environnement Aster permet un déploiement systématique du système distribué supportant un service complexe (l'application) au regard des propriétés traitées.

Contributions

Le principal objectif de cette thèse a été de démontrer que les agents mobiles sont adaptés pour la composition et la spécialisation de services, particulièrement quand des propriétés de performance ou de flexibilité sont nécessaires. Pour ce faire, nous avons :

1. défini un cadre, simple d'utilisation, pour une spécification déclarative de services distribués complexes. L'approche adoptée, à deux niveaux, permet de compiler automatiquement une spécification de haut niveau en une mise en œuvre précisant les mécanismes d'interaction. Cette compilation permet de comparer et sélectionner des interactions traditionnelles et à base de code mobile dans les architectures de services. Nous ne connaissons pas d'autres approches déclaratives permettant ainsi de guider la conception de services complexes par composition de services primitifs.

2. défini des analyses de qualité (c.-à-d. performance, sécurité, fiabilité) vouées à évaluer les avantages/inconvénients des agents mobiles dans les architectures considérées. L'approche déclarative suivie, restreinte à l'utilisation des interfaces, nous a permis d'intégrer uniformément, dans un même cadre, plusieurs propriétés de qualité. Nos analyses de qualité permettent de sélectionner les meilleurs mécanismes d'interaction et itinéraires statiques d'agents. Le critère de performance, souvent mis en avant pour justifier de l'intérêt des déports de traitements à l'aide d'agents mobiles, est ici pris en compte. Les analyses prennent également en compte les nouveaux problèmes de confidentialité et d'intégrité posés par les agents mobiles, ainsi que les contraintes de fiabilité des exécutions.

3. intégré le cadre et ses analyses dans un environnement de construction qui différencie clairement les rôles de l'architecte, du concepteur et de l'équipe de développement de ser-

vices distribués complexes. L'environnement permet de traiter les contraintes de qualité à la fois dans l'étape du choix des mécanismes d'interactions (c.-à-d. sélection) et dans l'étape de déploiement du système (c.-à-d. spécialisation de la couche intergicielle), et ce de manière invisible à l'architecte. L'architecte a en charge de spécifier abstraitement les fonctionnalités d'un service distribué complexe, sans ce soucier directement des détails de la construction tels que la qualité de la mise en œuvre. Le concepteur du service complexe se charge d'orchestrer « au mieux» les interactions avec les services primitifs, qu'ils soient locaux ou distants. L'équipe de développement se charge de développer le système et implanter le service complexe, assisté de l'environnement Aster pour la spécialisation du système.

Perspectives

Notre langage de spécification de services complexes s'apparente aux langages de scripts *workflow*. Il s'appuie sur la composition fonctionnelle des interfaces de services primitifs et de traitements. Il nous aurait été possible de considérer d'autres formalismes pour spécifier des services complexes, à la manière de ceux généralement employés pour la spécification de *workflow* ou de systèmes à agents. Bien que ces approches soient largement étudiées, leur expressivité plus fine rend très difficile les analyses statiques de propriétés non fonctionnelles globales. Certaines extensions de notre langage méritent d'être étudiées :

- **l'adaptation dynamique :** les choix d'interactions et les itinéraires d'agents sont « câblés» dans les expressions du langage concret en fonction de critères définis statiquement. Pour permettre aux interactions de s'adapter aux services primitifs ou à leurs propriétés non fonctionnelles, nous avons proposé d'utiliser une méthode d'adaptation qui utilise plusieurs mises en œuvre d'un service complexe (cf. section 4.4.4, page 98). La spécialisation statique envisagée utilise des branches conditionnelles dans les codes d'agents, dont les choix sont dépendants de l'environnement. Toutefois, pour les analyses et l'implantation des agents, nous avons restreints ces choix à être exécutés sur le site client. Il serait intéressant de trouver un moyen d'inclure ces choix au sein des agents (en court d'itinéraire) sans remettre en cause l'interprétation sémantique de nos analyses. Les agents seraient ainsi équipés avec la capacité de s'adapter à l'exécution. Les propriétés de qualité seraient affectées par cette spécialisation car les agents transporteraient plusieurs branches conditionnelles alors que seulement certaines seraient exécutées. L'adaptation dynamique pourrait apporter un début de solution à ce problème. L'adaptation dynamique consiste à créer des agents mobiles qui échangent dynamiquement des parties de code (p.ex. des branches conditionnelles) avec le client en court d'exécution [BR01]. Les parties de code adaptables sont contenues dans un répertoire présent sur le site du client. Une procédure d'adaptation interne aux agents sélectionne, rapatrie et intègre la mise en œuvre adéquate. Seules les parties de code réellement à exécuter sont transférées à travers le réseau, parfois sur des canaux différents de l'itinéraire d'un agent.

- **la communication à distance par les agents :** nous nous sommes restreints au concept d'agents mobiles vus comme une variation du schéma client-serveur. Un agent mobile interagit en local avec des services primitifs et effectue des traitements personnalisés sur les sites visités. Cette définition des agents mobiles peut paraître restrictive à la vue des travaux existants dans les communautés de recherche agents (p.ex. intelligence

artificielle, système). Au sens large, un agent est capable de communiquer, en local ou à distance, avec les services, le client, voire avec d'autres agents mobiles. Le dernier point porte plutôt sur les systèmes multi-agents et nos analyses n'y sont pas adaptées dans la mesure où nous nous appuyons sur la localisation statique des services primitifs sur le réseau. Les formalismes à base de calcul de processus (p.ex. Ambiant, Join, Seal) sont plus favorables pour représenter plusieurs entités mobiles susceptibles d'entrer en rendez-vous. De plus, des services complexes basés sur des services primitifs nomades restent à être justifiés dans le cadre des architectures de services.

Si, au cour de son itinéraire, un agent a la capacité de communiquer à distance avec un service primitif (c.-à-d. par une invocation distante), la complexité de nos analyses va s'accroître dans la mesure où chaque site sélectionné d'un itinéraire peut devenir un centre d'invocation à distance d'autres services primitifs. Ces interactions délocalisées peuvent être traitées dans nos analyses *via* quelques adaptations. Toutefois, nous considérerons que dans les architectures de services multi-prestataires, les sites sont ouverts à l'accueil d'agents clients quand les exécutions sont liées aux services primitifs offerts et non simplement liées aux ressources de calcul et de communication disponibles. Dans ce cas, il est cependant envisageable de permettre à un agent mobile de transmettre des résultats partiels au client au cour de son itinéraire. Ces communications peuvent faciliter des garanties/optimisations de qualité, en permettant par exemple à un agent de se délester de résultats de son sac à dos. La communication par message (p.ex. cf. MOM, systèmes multi-agents) est une alternative aux invocations distantes et aux agents mobiles. Dans notre approche, les résultats ne sont rapportés qu'au moment où une primitive de retour de l'agent $go_{i,client}$ apparaît. Les répercussions de ce moyen de communication restent à être approfondies dans nos analyses.

Pour ce qui est des analyses, nous avons proposé un cadre de travail qui a permis de traiter uniformément des propriétés de qualité. Ces propriétés sont particulièrement pertinentes dans le cadre des architectures de services. Les analyses sont exactes à leur niveau d'abstraction et fournissent plus un guide au concepteur qu'une méthode de prototypage. Ainsi, pour la performance, nous nous sommes limités à l'étude de la taille des messages, sachant que la prise en compte des critères temporels, sur un réseau tel que l'Internet, impose des connaissances dynamiques sur l'environnement. D'autre part, pour la sécurité et la fiabilité, nous nous sommes limités à un domaine de valeur binaire. Pour la fiabilité, il aurait été possible de considérer un domaine basé sur la probabilité de défaillances sur les canaux et sites.

Le concepteur peut être amené à considérer d'autres propriétés non fonctionnelles souvent mises en avant pour justifier de l'intérêt des agents mobiles. Des propriétés telles que la consommation d'énergie dans le cas des terminaux clients mobiles ou encore la tarification des connexions en fonction de volumes de données sont très proche de l'analyse de performance. Certaines propriétés sont liées à des aspects dynamiques qui ne sont pas traités dans les fondements de ce document. Par exemple, la disponibilité de service est relativement délicate à traiter de manière précise. D'autres propriétés ne peuvent être associées à un domaine de valeur (p.ex. asynchronisme) et paraissent donc difficilement intégrables dans notre cadre. En dehors de ces considérations de précision/finesse des analyses, les points à approfondir sur les analyses portent sur :

- **la mobilité des clients :** nous avons considéré une vue statique du réseau pour chacune de nos analyses. Les valeurs non fonctionnelles des connexions sont fixées. De plus

en plus, les prestataires de services complexes auront à faire à des clients nomades dotés de terminaux sans fils. L'attrait des traitements déportés à l'aide d'agents mobiles prend ici tout son sens pour ces terminaux à faibles ressources, aux accès payants et souffrants de déconnexions fréquentes à cause de la mobilité. Par utilisation d'un site cache comme point d'entrée personnel au réseau pour le client (c.-à-d. site intermédiaire fixe), nos analyses restent utilisables en tant que tel. Alternativement, si l'on considère des connexions directes entre le client nomade et les prestataires, la validité des analyses reste à être étudiée. En effet, pour une propriété telle que la performance, la bande passante ou la latence peuvent changer dramatiquement quand le client se déplace. Les moyens de traiter la mobilité des clients dans nos analyses s'orientent naturellement vers une adaptation dynamique des services complexes, dépendantes à un temps donné des propriétés de l'équipement du terminal et de l'infrastructure de communication.

- **l'étude d'heuristiques :** les analyses proposées sont intrinsèquement globales et nécessitent une étude exhaustive des différents choix de mise en œuvre. Elles sont exactes et coûteuses, sans doute trop si le langage est utilisé dans ses versions étendues. Elles peuvent toutefois vraisemblablement bénéficier d'heuristiques admissibles afin de limiter ces choix (p.ex. traitement fortement décroissant pour la performance).

  S'intéresser à l'extensibilité semble être une première piste de recherche d'heuristiques. Les services distribués complexes ont parfois besoin d'être mis à jour pour s'adapter aux changements des besoins des clients. Modifier (p.ex. redistribution ou substitution de services) ou étendre un service complexe donné et déjà analysé impose le plus souvent une nouvelle analyse globale. Il serait intéressant de chercher à déterminer les cas où une nouvelle analyse allégée, peut-être non exacte, s'appuyant sur des résultats antérieurs, pourrait être satisfaisante sur certains motifs.

- **les interactions inter-propriétés :** nous avons traité quasi-indépendamment les propriétés de qualité dans l'environnement de construction. Cependant, la prise en compte «au mieux» d'une propriété peut avoir des répercussions néfastes sur d'autres propriétés. Par exemple, les propriétés de performance et de sécurité peuvent entrer en contradiction quand un mécanisme de sécurité induit une augmentation de trafic. Ceci est un problème d'autant plus délicat que certains attributs sont parfois antagonistes et nécessitent de faire des compromis. Ainsi, dans notre environnement, si des propriétés sont considérées indépendamment à différentes étapes, des choix de conception peuvent devenir non valides dans les étapes finales. La cohérence des vues non fonctionnelles est un problème à considérer dans notre approche. Un début de solution à ce problème serait de déterminer les contraintes d'intégration d'une propriété au regard des autres propriétés. Les contraintes seraient issues des mécanismes qu'il convient de mettre en œuvre pour que le système résultant soit de la qualité requise.

En ce qui concerne l'environnement de construction, il serait utile de proposer des outils de vérification à l'architecte d'un service complexe. Par exemple, des vérifications de correspondance de types entre les interfaces des fonctions utilisées, seraient à même d'éviter des compositions non valides. Par ailleurs, de plus en plus de conceptions se font *via* des techniques de spécification associées à des outils largement répandus (p.ex. notation UML pour le futur standard MDA). Il est possible d'adapter notre approche par ADL dans de telles notations.

Pour le concepteur, l'intégration des différentes analyses dans un environnement graphique peut permettre d'identifier plus facilement les points de choix des conceptions au sein de la vue réseau. En effet, le concepteur dispose de plusieurs analyses et de nombreuses spécifications qui peuvent altérer la visibilité des problèmes pour des études non triviales.

Pour l'équipe de développement, nous avons proposé un prototype lié à une plate-forme particulière. Il serait intéressant de le compléter et l'adapter aux nouveaux/futurs standards intergiciels, adaptés à l'Internet et susceptibles de supporter les agents mobiles en plus des interactions traditionnelles. Des études restent à être menées pour intégrer des mécanismes et fournir des outils aptes à garantir des propriétés de qualité (c.-à-d. en s'inspirant des techniques de spécialisation proposées au sein de l'environnement Aster) pour de tels intergiciels.

Bibliographie

[AAG93] Abowd (Gregory), Allen (Robert) et Garlan (David). – Using Style to Understand Descriptions of Software Architecture. *In : Proceedings of the ACM SIGSOFT'93 Symposium on the Foundations of Software Engineering*. pp. 9–20. – Software Engineerings Notes, vol. 18, n 5, ACM Press, décembre 1993.

[AG94] Allen (Robert) et Garlan (David). – Formalizing Architectural Connection. *In : Proceedings of the 16th International Conference on Software Engineering*. pp. 71–80. – IEEE Computer Society Press, mai 1994.

[AMSR93] Attie (Paul C.), Munindar (P.), Sheth (Amit P.) et Rusinkiewicz (Marek). – Specifying and Enforcing Intertask Dependencies. *In : Proceedings of the 19th VLDB Conference*. – Morgan Kaufmann Publishers, août 1993.

[AWO+99] Arnold (Ken), Wollrath (Ann), O'Sullivan (Bryan), Scheifler (Robert) et Waldo (Jim). – *The Jini Specification*. – Addison-Wesley, 1999.

[BABR96] Bellissard (Luc), Atallah (Slim Ben), Boyer (Fabienne) et Riveill (Michel). – Distributed Application Configuration. *In : Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96)*. pp. 579–585. – IEEE, mai 1996.

[Ber99] Bernard (Guy). – Applicabilité et performances des systèmes d'agents mobiles dans les systèmes répartis. *In : Actes 1ère conférence française sur les systèmes d'exploitation (CFSE)*, pp. 57–68. – IRISA, Rennes, juin 1999.

[BGL00] Boudol (Gérard), Germain (Florence) et Lacoste (Marc). – *Analyse des langages et modèles de la mobilité*. – rapport de recherche n3930, INRIA, avril 2000.

[BI95] Bidan (Christophe) et Issarny (Valérie). – *Un aperçu des problèmes de sécurité dans les systèmes informatiques*. – publication interne n 959, IRISA, campus universitaire de Beaulieu, 35042 Rennes cedex, octobre 1995.

[BI97] Bidan (Christophe) et Issarny (Valérie). – Security Benefits from Software Architecture. *In : Proceedings 2nd International Conference on Coordination Models and Languages*, éd. par Garlan (David) et Le Métayer (Daniel). pp. 64–80. – Lecture Notes in Computer Science, n 1282, Springer, septembre 1997.

[BIM98] Balsamo (Simonetta), Inverardi (Paola) et Mangano (Calogero). – An Approach to Performance Evaluation of Software Architecture. *In : Proceedings of the 1998 Workshop on Software and Performance*. – octobre 1998.

[BN84] Birrell (Andrew D.) et Nelson (Bruce Jay). – Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, vol. 2, n 1, février 1984, pp. 39–59.

[BN01] Bettini (Lorenzo) et Nicola (Rocco De). – Translating Strong Mobility into Weak Mobility. *In : Proceedings of the 5th International Symposium on Mobile Agents (MA'01)*, éd. par Picco (G. P.). pp. 182–197. – Lecture Notes in Computer Science, n 2240, Springer, décembre 2001.

[BR01] Brandt (Raimund) et Reiser (Helmut). – Dynamic Adaptation of Mobile Agents in Heterogenous Environments. *In : Proceedings of the 5th International Symposium on Mobile Agents (MA'01)*, éd. par Picco (G. P.). pp. 70–87. – Lecture Notes in Computer Science, n 2240, Springer, décembre 2001.

[BZS93] Bershad (Brian N.), Zekauskas (Matthew J.) et Sawdon (Wayne A.). – The Midway Distributed Shared Memory System. *In : Proceedings of the 38th IEEE International Computer Conference (COMPCON'93)*. pp. 528–537. – IEEE, février 1993.

[Car99] Cardelli (Luca). – Abstractions for mobile computations. *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, 1999, pp. 51–94.

[CaSW01] Curbera (Francisco) et ans Sanjiva Weerawarana (Nirmal Mukhi). – *On the Emergence of a Web Services Component Model.* – rapport, IBM T.J. Watson Research Center, NY, USA, juin 2001.

[CCPP95] Casati (Fabio), Ceri (Stefano), Pernici (Barbara) et Pozzi (Guiseppe). – Conceptual Modeling of Workflows. *In : Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference*, éd. par Papazoglou (M. P.). pp. 341–354. – Lecture Notes in Computer Science, n 1021, Springer, décembre 1995.

[CDK01] Coulouris (George F.), Dollimore (Jean) et Kindberg (Tim). – *Distributed Systems, Concepts and Design.* – Addison-Wesley, 3ème édition, 2001.

[CG98] Cardelli (Luca) et Gordon (Andrew D.). – Mobile Ambients. *In : Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS), European Joint Conferences on Theory and Practice of Software*. pp. 140–155. – Lecture Notes in Computer Science, n 1378, Springer, mars 1998.

[CHK94] Chess (David M.), Harrison (Colin G.) et Kershebaum (Aaron). – *Mobile Agents: Are They a Good Idea?* – rapport technique nRC 19887, T.J. Watson Reserach Center, NY, USA, IBM Research Division, février 1994.

[CK97] Chia (T.) et Kannapan (S.). – Strategically Mobile Agents. *In : Proceedings of the First International Conference on Mobile Agents (MA'97)*, éd. par Rothermel (K.) et Popescu-Zeletin (R.). pp. 149–161. – Lecture Notes in Computer Science, n 1219, Springer, avril 1997.

[CLR96] Cormen (Thomas H.), Leiserson (Charles E.) et Rivest (Ronald L.). – *Introduction à l'algorithmique (trad.).* – Dunod, juin 1996.

[CP97] Chevochot (Pascal) et Puaut (Isabelle). – *Tolérance aux fautes dans les systèmes d'exploitation temps-réel à sûreté critique.* – publication interne n1142, IRISA, campus universitaire de Beaulieu, 35042 Rennes cedex, novembre 1997.

[CPV97] Carzaniga (Antonio), Picco (Gian Pietro) et Vigna (Giovanni). – Designing Distributed Applications with Mobile Code Paradigms. *In : Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, éd. par Taylor (R.). pp. 22–32. – ACM Press, mai 1997.

[Cri91] Cristian (Flaviu). – Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM*, vol. 34, n2, février 1991, pp. 56–78.

[DAI99] Demairy (Erwan), Anceaume (Emmanuelle) et Issarny (Valérie). – The Correctness of Multimedia Applications. *In : Proceedings of the 11 th Euromicro Conference on Real Time Systems.* – juin 1999.

[FGL+96] Fournet (Cédric), Gonthier (Georges), Lévy (Jean-Jacques), Maranget (Luc) et Rémy (Didier). – A Calculus of Mobile Processes. *In : CONCUR '96: Concurrency Theory, 7th International Conference.* pp. 406–421. – Lecture Notes in Computer Science, n 1119, Springer, août 1996.

[FIR00] Fradet (Pascal), Issarny (Valérie) et Rouvrais (Siegfried). – Analysing Non-functional Properties of Mobile Agents. *In : Proceedings of the Third International Conference on Fundamental Approaches to Software Engineering (FASE 2000). Partie de ETAPS'00*, éd. par Maibaum (Tom). pp. 319–333. – Lecture Notes in Computer Science, n 1783, Springer, mars/avril 2000.

[FLP99] Fradet (Pascal), Le Métayer (Daniel) et Périn (Mickaël). – Consistency Checking for Multiple View Software Architectures. *In : Proceedings of the joint 7th European Software Engineering Conference (ESEC) and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-7).* – Lecture Notes in Computer Science, Springer, septembre 1999.

[FPV98] Fuggetta (Alfonso), Picco (Gian Pietro) et Vigna (Giovanni). – Understanding Code Mobility. *IEEE Transactions on Software Engineering*, vol. 24, n5, mai 1998, pp. 342–361.

[FSB+00] Ferreira (Paulo), Shapiro (Marc), Blondel (Xavier), Fambon (Olivier), Garcia (Joao), Kloosterman (Sytse), Richer (Nicholas), Roberts (Marcus), Sandakly (Fadi), Colouris (George), Dollimore (Jean), Guedes (Paulo), Hagimont (Daniel) et Krakowiak (Sacha). – PerDiS: Design, Implementation and Use of a PERsistent DIstributed Store. *In : Recent Advances in Distributed Systems*, éd. par Krakowiak (S.) et Shrivastava (S. K.). pp. 427–452. – Lecture Notes in Computer Science, n 1752, Springer, février 2000.

[Gar95] Garlan (David). – Introduction to the Special Issue on Software Architecture. *IEEE Transactions on Software Engineering*, vol. 21, n4, avril 1995.

[Gar96] Garlan (David). – Style-Based Refinement for Software Architecture. *Joint Proceedings of the Second International Software Architecture Workshop (ISAW2)*

and Intrenational Workshop on Multiple Perspectives in Software Development, octobre 1996, pp. 72–75.

[Gar00] Garlan (David). – Software Architecture. *In : Proceedings of the 22th International Conference on Software Engineering (ICSE-00).* pp. 91–102. – ACM Press, juin 2000.

[Gel85] Gelernter (David). – Generative Communications in Linda. *ACM Transactions on Programming Languages and Systems*, vol. 7, n1, janvier 1985, pp. 80–112.

[GG96] Gardarin (Georges) et Gardarin (O.). – *Le Client-Serveur.* – Eyrolles, mars 1996.

[GM01] Gervais (Marie-Pierre) et Muscutariu (Florin). – Towards and ADL for Designing Agent-Based Systems. *In : Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering (AOSE'01)*, éd. par Wooldridge (M.J), Weiß(G.) et Ciancarini (P.). – Lecture Notes in Computer Science, n 2222, Springer, mai 2001.

[GS93] Garlan (David) et Shaw (Mary). – An Introduction to Software Architecture. *In : Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, Vol 2*, éd. par Ambriola (V.) et Tortora (G.), pp. 1–39. – Singapore, Worlds Scientific Publishing Company, 1993.

[Han94] Hankin (Chris). – *Lambda Calculi: A Guide for Computer Scientists.* – Clarendon Press, 1994.

[Hoh] Hohl (Fritz). – The Mobile Agent List: Names only Table of Known Mobile Agent Systems. *In :* `mole.informatik.uni-stuttgart.de/mal/mal.html`.

[Hoh98a] Hohl (Fritz). – A Model of Attacks of Malicious Hosts against Mobile Agents. *In : 4th ECOOP Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations.* – 1998.

[Hoh98b] Hohl (Fritz). – Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts. *Mobile Agents and Security*, 1998, pp. 92–113.

[Hol94] Hollinsworth (D.). – *The Workflow Reference Model.* – Rapport technique n TC00-1003, Workflow Management Coalition, décembre 1994.

[IB96] Issarny (Valérie) et Bidan (Christophe). – Aster: A Framework for Sound Customization of Distributed Runtime Systems. *In : Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS'96), Hong Kong.* pp. 586–593. – IEEE, mai 1996.

[IBM01] IBM et Microsoft. – Web Services Framework. *W3C Workshop on Web services: Position papers*, no51, avril 2001. – `www.w3.org/2001/03/WSWS-popa/paper51`.

[IH99] Ismail (Leila) et Hagimont (Daniel). – A Performance Evaluation of the Mobile Agent Paradigm. *In : Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 306–313. – 1999.

[IKV] IKV++ GmbH. – Grasshopper. – `www.grasshopper.de`.

[ISO92] ISO. – *Basic Reference Model of Open Distributed Processing, Part 1: Overview and Guide to use.* – Rapport technique n ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Standards Organisation, 1992.

[ISO95] ISO. – *Information Technology - Software Life Cycle Processes.* – Rapport technique nISO/IEC 12207, International Standards Organisation, 1995.

[IW95] Inverardi (Paola) et Wolf (Alex). – Formal Specification and Analysis of Software Architectures using the Chemical Abstract Machine Model. *IEEE Transactions on Software Engineering*, vol. 21, n4, avril 1995, pp. 373–386.

[Joh98] Johansen (Dag). – Mobile Agent Applicability. *In : 2nd International Workshop on Mobile Agents, MA'98, Stuttgart, Germany.* pp. 80–98. – Lecture Notes in Computer Science, n 1477, Springer, septembre 1998.

[Joh00] Johansen (Dag). – Trend Wards: Mobile Agent Applications. *IEEE concurrency*, juillet 2000, pp. 80–90.

[KGMW99] Karamanolis (Christos), Giannakopoulou (Dimitra), Magee (Jeff) et Wheater (Stuart). – *Modelling and Analysis of Workflow Processes.* – Rapport technique n99/2, Londres, Angleterre, Departement of Computing, Imperial College, septembre 1999.

[Kle91] Klein (Johannes). – Advanced Rule Driven Transaction Management. *In : Proceedings of the 36th IEEE Computer Society International Conference COMPCON*, pp. 562–567. – 1991.

[Kra94] Kramer (Jeff). – Distributed Software Engineering. *In : Proceedings of the 16th International Conference on Software Engineering*, éd. par Fadini (Bruno). pp. 253–266. – IEEE Computer Society Press, mai 1994.

[Kra98] Krakowiak (Sacha). – Introduction aux applications réparties. *École INRIA : construction d'applications réparties*, août 1998.

[Lap95] Laprie. (Jean Claude) (édité par). – *Guide de la sûreté de fonctionnement.* – Cépaduès Éditions, 1995, *LIS : laboratoire d'Ingénierie de la Sûreté de fonctionnement.*

[Le 96] Le Métayer (Daniel). – Software Architecture Styles as Graph Grammars. *Proceedings of the ACM SIGSOFT Symposium of the foundations of Software Engineering*, 1996, pp. 15–23.

[Ley01] Leymann (Frank). – *Web Services Flow Language (WSFL 1.0).* – rapport technique, IBM Software Group, mai 2001.

[LKA+95] Luckham (David C.), Kenney (John J.), Augustin (Larry M.), Vera (James), Bryan (D.) et Mann (Walter). – Specification and Analysis of System Architecture using Rapide. *IEEE Transactions on Software Engineering*, vol. 21, n4, avril 1995, pp. 336–355.

[LO99] Lange (Danny B.) et Oshima (Mitsuru). – Seven Good Reasons for Mobile Agents. *Communications of the ACM*, vol. 42, n3, mars 1999, pp. 88–89.

[MBB+98] Milojicic (D.), Breugst (M.), Busse (I.), Campbell (J.), Covaci (S.), Friedman (B.), Kosaka (K.), Lange (D.), Ono (K.), Oshima (M.), Tham (C.), Virdhagriswaran (S.) et White (J.). – MASIF: The OMG Mobile Agent System Interoperability Facility. *In : Proceedings of the 2nd International Workshop on Mobile Agents*, éd. par Rothermel (Kurt) et Hohl (Fritz). pp. 50–67. – Lecture Notes in Computer Science, n 1477, Springer, 1998.

[McL94] McLean (John). – Security Models. *Encyclopedia of Software Engineering (Ed. John Marciniak), Wiley and Sons, Inc.*, 1994.

[Mil98] Milner (Robin). – The Pi Calculus and Its Applications. *In : Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, éd. par Jaffar (Joxan). – MIT Press, juin 1998.

[MK95] Magee (Jeff) et Kramer (Jeff). – Modelling distributed Software Architectures. *Proceedings of the First International Workshop on Architecture for Software Systems*, avril 1995.

[MKMG97] Monroe (Robert T.), Kompanek (Andrew), Melton (Ralph) et Garlan (David). – Architectural Styles, Design Patterns, and Objects. *IEEE Software*, janvier 1997, pp. 43–52.

[MQR95] Moriconi (Mark), Qian (Xiaolei) et Riemenschneider (R.A.). – Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, vol. 21, n4, avril 1995, pp. 356–372.

[MRRR02] Medvidovic (N.), Rosenblum (D. S.), Redmiles (D. F.) et Robbins (J. E.). – Modeling Software Architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, vol. 11, n1, 2002, pp. 2–57.

[MT00] Medvidovic (Nenad) et Taylor (Richard N.). – A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, vol. 26, n1, janvier 2000, pp. 70–93.

[NACO97] Naumovich (Gleb), Avrunin (George S.), Clarke (Lori A.) et Osterweil (Leon J.). – *Applying Static Analysis to Software Architectures.* – rapport technique n97-008, Computer Science Department, University of Massachusetts at Amherst, janvier 1997.

[NL98] Necula (George C.) et Lee (Peter). – Safe, Untrusted Agents using Proof-Carrying Code. *In : Mobile Agent Security*, éd. par Vigna (Giovanni), pp. 61–91. – Lecture Notes in Computer Science, n 1419, Springer, 1998.

[omg91] *Common Object Request Broker: Architecture and Specification.* – Rapport technique, The Object Management Group, 1991. `www.omg.org`.

[Ord96] Ordille (Joan J.). – When Agents Roam, Who can you Trust? *In : First Conference on Emerging Technologies and Applications in Communications (etaCOM).* – mai 1996.

[osf91] *OSF DCE 1.0 Application Development Guide.* – Rapport technique, Open Software Foundation, décembre 1991.

[p2p] *What is Peer-to-peer?* – Rapport technique, Peer-to-Peer Working Group. `www.peer-to-peerwg.org`.

[Pap00] Papaioannou (Todd). – *On the Structuring of Distributed Systems: The Argument for Mobility.* – Thèse de PhD, Loughborough University, février 2000.

[Par90] Parrington (Graham D.). – Reliable Distributed Programming in C++: The Arjuna Approach. *In : USENIX C++ conference proceedings: C++ Conference*, éd. par USENIX Association. pp. 37–50. – USENIX, avril 1990.

[Per97] Perret (Stephane). – *Agents mobiles pour l'accès nomade à l'informatique répartie dans les réseaux de grande envergure.* – thèse de doctorat, Université de Grenoble (LSR-IMAG), novembre 1997.

[Pic98] Picco (Gian Pietro). – *Understanding, Evaluating, Formalizing, and Exploiting Code Mobility.* – thèse de doctorat, Politecnico di Torino, Italie, février 1998.

[Pra90] Pratt (Vaugham R.). – Action Logic and Pure Induction. *In : Logics in AI: European Workshop JELIA'90.* pp. 97–120. – Lecture Notes in Computer Science, n 478, Springer, 1990.

[PW92] Perry (Dewayne E.) et Wolf (Alexander L.). – Foundations for the Study of Software Architecture. *ACM SIGSOFT, Software Engineering Notes*, vol. 17, n 4, octobre 1992, pp. 40–52.

[Rey93] Reynolds (John C.). – The Discoveries of Continuations. *Lisp and Symbolic Computation*, vol. 6, n3/4, novembre 1993, pp. 233–248.

[RG91] Russel (D.) et Gangemi (G. T.) (édité par). – *Computer Security Basics.* – O'Reilly & Associates, 1991.

[Rou97] Rouvrais (Siegfried). – *Architectures logicielles et propriétés de sécurité.* – mémoire de DEA, IFSIC, Université de Rennes1, septembre 1997.

[Rou02] Rouvrais (Siegfried). – Construction de services distribués : une approche à base d'agents mobiles. *Technique et science informatiques*, vol. 21, n 7, 2002. – à paraître.

[San93] Sangiorgi (Davide). – *Expressing Mobility in Process Algebra.* – Edinburgh, Ecosse, thèse de doctorat, University of Edinburgh, 1993.

[Sar99] Saridakis (Titos). – *Construction robuste de systèmes logiciels sûrs de fonctionnement.* – thèse de doctorat, Université de Rennes1, mars 1999.

[Sch86] Schmidt (David A.). – *Denotational Semantics: a Methodology for Language Development.* – Allyn and Bacon, 1986.

[Sch97] Schneider (Fred B.). – *Towards Fault-tolerant and Secure Agentry.* – rapport technique n97-1636, Cornell University, USA, juillet 1997.

[Ses97] Sessions (Roger). – *COM and DCOM: Microsoft's Vision for Distributed Objects*. – John Wiley & Son, 1997.

[SG90] Stamos (James W.) et Gifford (David K.). – Remote Evaluation. *ACM Transactions on Programming Languages and Systems*, vol. 12, n4, octobre 1990, pp. 537–565.

[SG95] Shaw (Mary) et Garlan (David). – Formulations and Formalisms in Software Architecture. *Computer Science Today, Recent Trends and Developments*, 1995, pp. 307–323.

[SG96] Shaw (Mary) et Garlan (David) (édité par). – *Software Architecture : Perspectives on an Emerging Discipline*. – Upper Saddle River, New Jersey 07458, Prentice-Hall, Inc., 1996.

[SG98] Spitznagel (Bridget) et Garlan (David). – Architecture-Based Performance Analysis. *In : Proceedings of the 1998 Conference on Software Engineering and Knoledge Engineering*. – 1998.

[SH82] Shoch (John F.) et Hupp (Jon A.). – The "Worm" Programs – Early Experience with Distributed Computing. *Communications of the ACM*, vol. 25, n3, mars 1982, pp. 172–180.

[SI99] Saridakis (Titos) et Issarny (Valérie). – Developing Dependable Systems using Software Architecture. *In : Proceedings of the 1st Working IFIP Conference on Software Architecture*, pp. 83–104. – février 1999.

[SIB98] Saridakis (Titos), Issarny (Valérie) et Bidan (Christophe). – Customized Remote Execution of Web Agents. *In : Proceedings of the 31st Hawaii International Conference on System Science*, pp. 614–620. – janvier 1998.

[SK88] Sloman (Morris) et Kramer (Jeff). – *Distributed Systems and Computer Networks*. – Prentice Hall, 1988.

[SM96] Schill (A.) et Mittasch (C.). – Workflow Management Systems on top of the OSF DCE and OMG CORBA. *Distributed System Engineering Journal*, décembre 1996.

[SM98] Sahai (Akhil) et Morin (Christine). – Enabling a Mobile Network Manager (MNM) through Mobile Agents. *In : Proceedings of the 2nd International Workshop on Mobile Agents*, éd. par Rothermel (Kurt) et Hohl (Fritz). pp. 249–260. – Lecture Notes in Computer Science, n 1477, Springer, 1998.

[SPZ98] Silva (F.M. Assis) et Popescu-Zeletin (R.). – An Approach for Providing Mobile Agent Fault Tolerance. *In : Proceedings of the 2nd International Workshop on Mobile Agents*. pp. 14–25. – Lecture Notes in Computer Science, n 1477, Springer, 1998.

[SR98] Straßer (Markus) et Rothermel (Kurt). – Reliability Concepts for Mobile Agents. *International Journal of Cooperative Information Systems*, vol. 7, n4, 1998, pp. 355–382.

[SRC84] Saltzer (Jerome H.), Reed (David P.) et Clark (David D.). – End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, vol. 2, n 4, novembre 1984, pp. 277–288.

[SS97] Straßer (Markus) et Schwehm (Markus). – A Performance Model for Mobile Agent Systems. *In : Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'97*, éd. par Arabnia (H. R.), pp. 1132–1140. – 1997.

[SSM+00] Silva (Luís M.), Soares (Guilherme), Martins (Paulo), Batista (Victor) et Santos (Luís). – Comparing the Performance of Mobile Agent Systems: A Study of Benchmarking. *In : Journal of Computer Communications, Special Issue on Mobile Agents for Telecommunications.* – mars 2000.

[ST98] Sander (Thomas) et Tschudin (Christian F.). – Protecting Mobile Agents Against Malicious Hosts. *In : Mobile Agents and Security*, éd. par Vigna (G.). pp. 44–60. – Lecture Notes in Computer Science, n 1419, Springer, 1998.

[Sun] *Object Serialization.* – documentation, Sun Microsystems, Inc. `java.sun.com/j2se/1.3/docs/guide/serialization/`.

[Sun91] *Design and Implementation of Transport–Independent RPC.* – rapport technique (*white paper*), ONC. Solaris 2.0, Sun Microsystems, Inc., 1991.

[Sun97] *Java Remote Method Invocation Specification.* – rapport technique, Sun Microsystems, Inc., 1997. `www.javasoft.com`.

[Sun98] *JavaSpaces: Innovative Java Technology that Simplifies Distributed Application Development.* – rapport technique (*white paper*), Sun Microsystems Inc., Palo Alto, USA, 1998. `java.sun.com/products/javaspaces/whitepapers/index.html`.

[Sun01] *Project JXTA: An Open, Innovative Collaboation.* – document technique, Sun Microsystems, Inc., 2001. `www.sun.com/software/jxta/jxta-doc.html`.

[SW98] Stafford (Judith A.) et Wolf (Alexander L.). – Architecture-Level Dependence Analysis in Support of Software Maintenance. *In : Proceedings of the 3rd International Wrokshop on Software Architecture (ISAW'98)*, pp. 129–132. – novembre 1998.

[SY97] Sekiguchi (Tatsurou) et Yonezawa (Akinori). – A Calculus with Code Mobility. *In : Proceedings 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS).* pp. 21–36. – Chapman and Hall, 1997.

[Tho97] Thorn (Tommy). – Programming Languages for Mobile Code. *ACM Computing Surveys*, vol. 29, n3, 1997, pp. 213–239.

[UB97] Ugai (T.) et Bursell (M.). – *Comparison of Autonomous Mobile Agent Technologies.* – Rapport technique, APM Ltd , Cambridge, FollowME ESPRIT Project, octobre 1997.

[VC98] Vitek (Jan) et Castagna (Giuseppe). – Towards a Calculus of Secure Mobile Computations. *In : IEEE Workshop on Internet Programming Languages.* – mai 1998.

[vdA00] van der Aalst (W. M. P.). – Verification of Workflow Task Structures: a Petri Net based Approach. *Information Systems*, vol. 25, n1, 2000, pp. 43–69.

[Vig97] Vigna (Giovanni). – Protecting Mobile Agents through Tracing. *In : 3rd ECOOP Workshop on Mobile Object Systems.* – juin 1997.

[W3C00] W3C. – *Extensible Markup Language (XML) 1.0 (Second Edition).* – Rapport technique, XML Protocol Working Group, 2000. `www.w3.org`.

[W3C01] W3C. – *SOAP Version 1.2.* – rapport de travail (*working draft*), XML Protocol Working Group, décembre 2001. `www.w3.org`.

[War] Warren (David). – The XSB Programming System. *In :* `xsb.sourceforge.net`.

[WBS98] Wilhelm (Uwe G.), Buttyàn (Levente) et Staamann (S.). – On the Problem of Trust in Mobile Agent Systems. *In : Symposium on Network and Distributed System Security.* pp. 114–124. – Internet Society, mars 1998.

[Whi94] White (James E.). – *Telescript Technology: The Foundation for the Electronic Marketplace.* – rapport technique (*white paper*), 2465 Latham Street, Mountain View, CA, USA, General Magic, Inc., 1994.

[WLAG93] Wahbe (Robert), Lucco (Steven), Anderson (Thomas E.) et Graham (Susan L.). – Efficient Software-Based Fault Isolation. *In : Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pp. 203–216. – 1993.

[WSB99] Wilhelm (Uwe G.), Staamann (S.) et Buttyàn (Levente). – Introducing Trusted Third Parties to the Mobile Agent Paradigm. *In : Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, éd. par Vitek (Jan) et Jensen (Christian D.), pp. 471–491. – New York, USA, Lecture Notes in Computer Science, n 1603, Springer, 1999.

[WWWK97] Waldo (Jim), Wyant (Geoff), Wollrath (Ann) et Kendall (Sam). – A Note on Distributed Computing. *In : Mobile Object Systems: Towards the Programmable Internet*, pp. 49–64. – Lecture Notes in Computer Science, n 1222, Springer, avril 1997.

[Xme02] XMethods, Web Services List. *In :* `www.xmethods.net`. – 2002.

[Yee99] Yee (Bennet S.). – A Sanctuary for Mobile Agents. *In : Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, éd. par Vitek (Jan) et Jensen (Christian D.), pp. 261–273. – Lecture Notes in Computer Science, n 1603, Springer, 1999.

[YY97] Young (Adam) et Yung (Moti). – Sliding Encryption: A Cryptographic Tool for Mobile Agents. *In : Proceedings of Fast Software Encryption Workshop.* pp. 230–241. – Lecture Notes in Computer Science, n 1267, Springer, 1997.

[Zar00] Zarras (Apostolos). – *Configuration systématique de middleware.* – thèse de doctorat, Université de Rennes1, mars 2000.

[ZI98] Zarras (Apostolos) et Issarny (Valérie). – A Framework for Systematic Synthesis of Transactional Middleware. *In : Middleware'98.* – 1998.

Table des matières

Table des figures

Résumé

La construction d'un service distribué complexe, à partir de services applicatifs primitifs, nécessite de garantir des propriétés qui ne dépendent pas directement des fonctionnalités de l'application telles que les propriétés de qualité de service (p.ex. performance, sécurité ou sûreté de fonctionnement). Le plus souvent, les mécanismes mis en oeuvre pour interagir avec les services reposent sur le schéma d'organisation client-serveur. Toutefois, les schémas avec mobilité du code fournissent une alternative intéressante à ces interactions traditionnelles. La thèse que nous soutenons est que la technologie agents mobiles a toute sa place dans l'ingénierie de services distribués complexes. En effet, des interactions combinant les schémas client-serveur et agents mobiles permettent souvent de mieux répondre aux exigences de qualité de service.

Dans cette thèse, nous proposons un cadre fonctionnel pour la spécification de compositions de services. Il prend en compte les interactions de type invocation distante, évaluation distante et agents mobiles. Ce cadre permet l'étude et la comparaison de différents choix d'interactions afin de garantir des contraintes de performance, de sécurité ou encore de fiabilité. Associé à des outils d'analyse, il s'intègre naturellement dans un environnement de construction qui différencie clairement les rôles de l'architecte, du concepteur et de l'équipe de développement. Cet environnement s'appuie sur le concept d'architecture de logiciels afin de guider le concepteur de services complexes dans les choix de mise en oeuvre. Nous illustrons cette intégration avec l'environnement de développement de systèmes distribués Aster qui apporte des techniques de spécialisation du système d'exécution au regard des propriétés de qualité.