Une approche hybride pour la spécification de système reconfigurable

Mâamoun Bernichi – Fabrice Mourlin
LACL Laboratoire d'Algorithmique, Complexité et Logique, Université Paris 12,
94010 Créteil Cedex, France
bernichi@gmail.com - fabrice.mourlin@wanadoo.fr

Résumé
La mobilité d'action au sein d'un logiciel est une voie d'évolution pour obtenir un logiciel plus réactif en particulier à son contexte d'exécution. Les travaux de recherche présentés dans ce document explorent cette notion suivant trois aspects. Une première facette est la spécification de la mobilité au travers de langages formels tel que le pi calcul d'ordre supérieur. Les spécifications obtenues représentent des supports d'analyse essentiels. Elles offrent des possibilités de génération d'informations importantes telles que des tests ou du code exécutable. Ainsi, la réalisation ou la mise en œuvre de la mobilité est la deuxième facette de ce travail, où il est davantage question de construction à partir de spécifications formelles. Les implémentations fournies ont alors un objectif essentiel de validation de propriétés. Enfin le dernier aspect abordé porte sur l'architecture des applications à base d'agents mobiles. La définition d'une structure logicielle commune exprime la volonté de réutiliser l'expérience acquise dans des domaines d'intérêt différents.

Abstract
Mobile agent is a key feature for software development; it is a progressive way to adapt software with its own environment and always leads to an application which is more reactive over time. This document describes the process of mobility through three main aspects. The first one is about specification of mobile actions via formal language such that higher order pi calculus. This kind of specification is one of the bases of formal reasoning. The formal specifications can also be used as input data for test and code generation. The second aspect represents the implementation of mobile agent based on design pattern. The main objective of this stage is to validate properties which were previously defined. Finally, the third aspect describes the application architecture using mobile agents. The aim of this stage is to define a software architecture which is used through all our developments. In order to illustrate this stage, we used three different areas of application: software monitoring, mobile agent server and numerical computing. They all share the main principles, but each item brings its own particularity to the others for the final solution.

1 Introduction

La mobilité au sein des logiciels répartis est en constante progression. Elle devient la réponse incontournable aux problèmes de reconfiguration réseau, de surveillance logicielle, de gestion de performance ou de sécurité. Elle apparaît comme solution pour assurer l'évolutivité des logiciels. Enfin, la mobilité ne se limite pas au réseau d'entreprise mais

gagne aussi les plateformes de chacun d'entre nous et plus particulièrement nos plateformes nomades : téléphone, PDA, ordinateur portable, etc…

Des problèmes apparaissent alors pour la construction de logiciels sur ces différentes plateformes. Ils portent sur des aspects complémentaires de la mobilité au cours du cycle de vie du logiciel, à savoir la spécification, la réalisation et l'architecture. Il devient alors incontournable de construire une approche génie logiciel orientée mobilité. Le présent document propose au travers des sections suivante une démarche formelle pour la conception de logiciels mobiles de qualité.

Notre travail porte sur notre approche formelle de la mobilité d'actions et les moyens de décrire de manière rigoureuse les aspects matériel et logiciel. Cette approche hybride est à l'origine du pilotage des phases de construction d'applications à base d'agents mobiles. De premières concrétisations issues des spécifications formelles sont ainsi décrites : tests et squelette de code.

2 Notion d'agent mobile

La notion d'agent mobile correspond avant tout à un concept du monde de la programmation répartie. Le terme d'agent est en lui même souvent utilisé pour tout et n'importe quoi. Par opposition à ces multiples utilisations, souvent dépendantes du domaine d'étude, nous proposons de débuter par des considérations simples concernant la notion d'agent: il représente une entité logicielle indépendante qui s'exécute sur un noeud d'un réseau. Un agent peut s'exécuter quand son utilisateur est déconnecté de son poste, même involontairement. Des agents peuvent s'exécuter sur des machines spéciales ou des serveurs dédiées pour des activités spécifiques. Ainsi l'indépendance de la plateforme support est une caractéristique première qui apparaît dans de nombreux exemples de systèmes d'agents en particulier ceux liés au web. Il est de plus capable de voyager d'un noeud à un autre noeud du réseau.

Wilhelm R. Rossak et Peter Braun donnent une définition opératoire [1] en insistant sur une phase transitoire d'acquisition de connaissance. Ainsi un agent mobile est considéré comme un programme autonome et personnalisable et; pour les plus aboutis, présentant des caractéristiques d'auto apprentissage et de communication avec ses alter ego pour une action coopérative. Ils sont le plus souvent dédiés à des tâches de recherche et de collecte d'information.

Des définitions similaires sont présentes dans de très nombreux travaux [2], [3], [4]. Mais la quasi-totalité des travaux font référence de façon directe ou indirecte aux papiers de Jacques Ferber dont [5], [6] où est employé la notion d'agents mobiles intelligents dans un cadre plus orienté intelligence artificielle.

Quel que soit le contexte d'évolution d'un agent mobile, ses déplacements ainsi que les activités qu'il réalise, s'effectuent sous le contrôle de son utilisateur (c'est-à-dire le receveur de l'agent). Par exemple, l'administrateur d'un serveur pourra utiliser un agent pour effectuer une collecte d'information afin de regrouper les traitements SQL sur une base de données distante. Un autre exemple est la collecte d'états locaux en vue de constituer une représentation de plus haut niveau de l'état d'un système. Ainsi la mobilité de l'agent peut être vue comme un atout sous plusieurs approches: d'une part elle offre une plus grande productivité de chaque composante de calcul du réseau, d'autre part elle permet une extraordinaire adaptabilité en cas d'absence ou de défaillance d'un nœud, support de

réception de l'agent. L'ensemble crée ainsi un environnement de travail où des actions composant des tâches (ou missions) sont exécutées par des agents sur des noeuds d'un réseau.

2.1 Un exemple d'application à base d'agents mobile

Un exemple classique d'application à base d'agents mobiles est la recherche dans une base de données répartie. Considérons un utilisateur qui souhaite avoir accès à une base d'informations réparties sur plusieurs sites distants. Imaginons ainsi que cet utilisateur soit un représentant pour une société de vente de métaux. Arrivé chez un de ses clients, il a besoin du prix de certains produits tels que l'acier inoxydable. Deux cas sont à prendre en considération: le prix des produits est une information de petite taille qui n'évolue que peu souvent (ou à périodicité connue) auquel cas, le représentant peut emmener cette information dans son cartable. Un second cas plus délicat correspond à la situation où le prix évolue constamment ou pire dépend de facteurs propres au client visité (quantité commandée, condition privilégiée, etc.). Il faut alors soumettre une demande vers la base d'information afin de disposer d'une réponse à jour en rapport avec la demande. Une relation client serveur semble suffire mais alors interviennent les problèmes classiques dus à une architecture à deux niveaux pour l'accès à la base d'information. En effet, si la demande nécessite des traitements complexes voire coûteux en temps, pourquoi obliger l'attente du représentant devant son poste, cette attente est de plus un risque évident de panne. Si cela se produisait, la demande devrait être reprise depuis le départ. De plus, toute évolution de l'applicatif coté serveur entraînerait un redéploiement de l'applicatif client. Nous allons voir comment les agents mobiles peuvent répondre à ce problème.

Tout d'abord une architecture logicielle doit être mise en place. Cela signifie qu'au moins un serveur d'agents doit être déployé. Ceci est une opération stratégique qui exige le choix d'une plateforme ayant des aptitudes réseau (intranet ou autres). Cette plateforme existe peut être déjà pour d'autres applications à base d'agents.

Enfin, le représentant doit disposer d'un ordinateur (portable pour ses déplacements) pour communiquer avec son système d'information, et recevoir des agents. Dans tous les cas, cette plateforme portable peut prendre différentes formes depuis l'ordinateur classique jusqu'au téléphone portable. Ses moyens de communication peuvent s'effectuer via un protocole dédié aux aspects locaux tel que bluetooth ou pour des échanges sur de plus grandes distances via wifi ou autre. Dans tous les cas le poste de notre représentant devra disposer de service de communication pré établis.

La troisième étape consiste à configurer la sécurité des différents postes et placer les permissions qui permettront d'identifier les agents intervenants dans l'application mais aussi leurs utilisateurs du système d'information. Cette phase est clairement une étape préalable pour un bon déroulement de toute exécution.

Finalement, lors du déploiement de l'application, le serveur a besoin de créer de premiers agents qui exécuteront les requêtes des représentants. De ce fait lorsque, la demande du représentant en métallurgie arrive au serveur d'agents, celui-ci lui fournit un agent dédié à sa demande. Cela signifie que cet agent va se déplacer sur le laptop du représentant afin de recevoir la requête et lorsque celle-ci sera validée, l'agent se déplacera sur les sites où se trouvent les informations dans le but de satisfaire la demande.

D'autres exemples d'applications basées sur l'emploi d'agents mobiles sont aujourd'hui très nombreux, plus particulièrement en téléphonie [7], en calcul numérique sur une grille [8] ou encore dans des systèmes de gestion de workflow [9].

2.2 Avantage de la notion d'agent mobile

Le premier atout est sans contexte la localité. En effet, une application employant des agents mobiles utilise le réseau pour migrer et ainsi réaliser ses tâches au plus proche des données utiles à ces tâches, plutôt que de rechercher celles-ci de manière distante. Le gain en performance est immédiat, les agents utilisent tout le spectre des services disponibles en chaque point du réseau, tels que les IHM des usagers pour afficher des observations, les interfaces vers les systèmes d'informations. De ce fait un agent mobile tire le meilleur du réseau grâce à ses déplacements.

Un aspect qui semble au départ secondaire est la capacité à utiliser des ressources de faible coût, portable, PC portable, périphériques de communication personnelles. Le réseau support comprend aussi les aspects sécuritaires et peut être pris en charge par un serveur léger capable de gérer les déplacements des divers agents. Associé à un modèle de programmation sophistiqué, cela permet de disposer d'agents de taille faible dont l'occupation mémoire n'est pas pénalisante sur les périphériques utilisateur parcourus. Ainsi, il n'est pas utile de réduire les fonctionnalités des agents.

La sécurité est une part importante d'un framework d'agents mobiles et il doit fournir des communications sécurisées même au dessus d'un réseau public. Les agents transportent des références utilisateurs tout au long de leurs déplacements. Ces références sont un moyen d'authentification durant l'exécution des tâches aux différentes haltes sur le réseau. Pour des traitements stratégiques, les agents et leurs données sont totalement cryptés lors de leur traversée. Bien entendu, toutes les phases de codage et décodage s'effectuent sans intervention de l'utilisateur.

Pour les déplacements de nœud en nœud, un mécanisme de sauvegarde et de transfert est mis en place, il est adapté aux problèmes classiques de communication, plus particulièrement dans le contexte de la mobilité. Les mises en attente par gestion de file ou les sauvegardes d'état à étapes régulières sont des aspects techniques qui participent à ce qu'il n'y ait aucune dégradation de la fiabilité ou de la qualité des réponses aux communications. Parce que les agents manipulent des données de manière locale aux nœuds où ils se trouvent, les variations réseau n'ont que peu d'effet sur le comportement des agents au cours de l'exécution de leur tâche.

Il apparaît clairement qu'une architecture logicielle dédiée aux systèmes d'agents mobiles doit être mise en place afin de permettre la définition de ces agents, leur publication, la gestion complète de leur cycle de vie et même le calcul de mesures permettant l'observation de ces mêmes agents.

3 Problème de spécification adaptée à la mobilité

3.1 Problème de spécification adaptée à la mobilité

En phase d'analyse et conception, la place des méthodes formelles n'est plus à présenter. Elles permettent, grâce à un langage particulier, d'exprimer très rigoureusement les

propriétés issues du cahier des charges. Elles offrent ensuite la possibilité de construire des preuves de manière automatisée ou semi automatisée pour des propriétés non ambiguës, cohérentes et non contradictoires. Elles garantissent par preuves mathématiques que ces propriétés sont respectées tout au long des étapes de conception. Les logiciels ainsi développés vont assurer ces propriétés par construction. On cherche ainsi à limiter au plus tôt et en phase de conception les événements dangereux qui pourraient apparaître en cas de comportement inapproprié du logiciel lors de l'exploitation sur machine.

Dès le début de nos travaux sur les spécifications d'agents mobiles, nous avons distingué clairement deux catégories de communications. Une première famille de communication porte sur les échanges dépendant du site sur lequel se situent les agents receveurs et émetteurs. Ces communications sont classiquement désignées comme communication de bas niveau car proches des aspects physiques du réseau. Une seconde famille de communications comprend les communications ne dépendant pas de caractéristiques physiques.

Pour illustrer cette notion, choisissons le cas d'un agent mobile, nommé collecteur, circulant sur un ensemble de sites pour collecter des informations. A l'arrivée sur chaque site, il demandera à un agent local, nommé lecteur, l'accès à la ressource recherchée. Cette communication est forcément locale et, si le destinataire existe, la communication pourra s'effectuer sans recherche de receveur (Figure 1 (cas a)). En revanche, si la communication ne peut se faire, une alarme devra être levée auprès d'un autre agent, nommé moniteur, présent sur un site à rechercher (Figure 1 (cas b)). Ce site joue ici le rôle d'observateur ou de superviseur du scénario de collecte d'information.

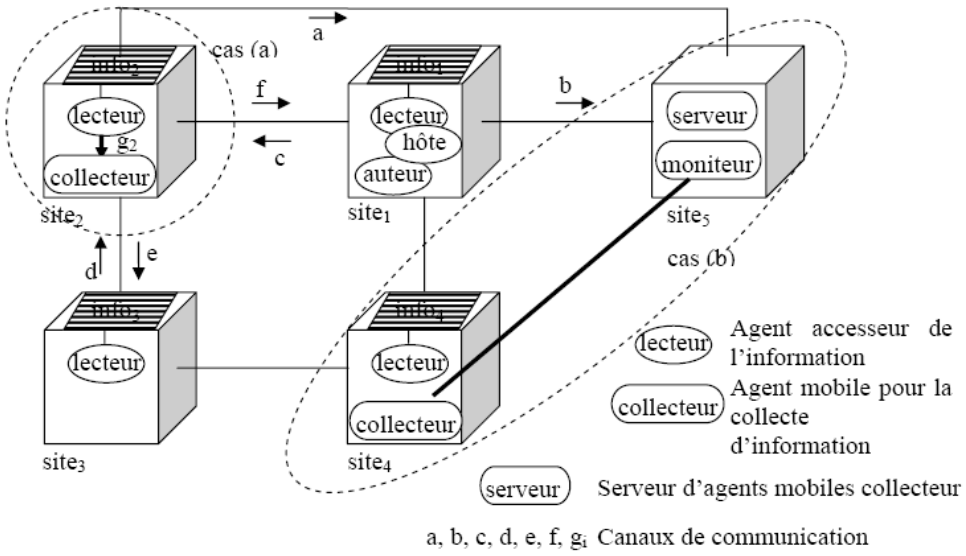

Figure 1 : deux familles de communication

Dans le cas (a), l'agent collecteur souhaite communiquer avec le lecteur du site sur lequel il se trouve, d'où le besoin d'exprimer la dépendance locale de cet agent. En revanche, l'agent moniteur est unique sur le réseau, il est le receveur de l'ensemble des alertes. Sa référence n'est donc pas dépendante du site où le collecteur se trouve.

Les descriptions indépendantes du placement permettent de simplifier grandement une spécification et offrent de fait des possibilités de réalisations plus importantes. De plus, elles

séparent les notions de déplacements et d'interactions du métier propre modélisé dans la spécification. Ainsi, les propriétés sont plus facilement décrites. Cependant, ces spécifications lèvent d'autres difficultés. En effet, une infrastructure répartie est indispensable pour faire le suivi de la migration des agents et le routage des messages à destination d'autres agents. Cette infrastructure doit prendre en compte des caractéristiques intrinsèques telles que l'indisponibilité ou la mise en échec, la latence réseau, le besoin de référentiel local, les aspects concurrents [10]. Ces caractéristiques sont souvent considérées comme inhérentes à un environnement d'exécution, mais leur expression reste délicate et ne peut se faire sans se référer au site précis appartenant à ce réseau. De plus, des applications peuvent être réparties sur différentes architectures : depuis un comportement local jusqu'à des réseaux de grande envergure. Il peut alors être utile d'expliciter des chemins particuliers de communication ou de migration, afin de choisir ou d'évaluer des coûts en performance ou en robustesse. Ces résultats aboutissent alors à des définitions d'architecture dédiée.

Ce besoin de compréhension et d'expérimentation, basées sur un choix d'infrastructure comportant des aspects techniques, met en valeur deux niveaux de descriptions. Ce besoin est incontournable si l'étude porte sur les choix d'architectures optimaux pour un métier donné. Une approche bas niveau consiste en un ensemble simple de descriptions de sites clairement identifiés sur lesquels peuvent être accueillis des agents communicants. Une partie haut niveau comprend un ensemble de descriptions d'agents indépendants de tout site et véhiculant l'information pour accomplir un métier donné. Ces deux approches permettent de disposer d'une représentation commune du niveau physique pour toutes les simulations logicielles qui seront effectuées. Cette double approche se retrouve d'ailleurs dans d'autres formalismes moins rigoureux tels qu'UML [11] avec l'écriture de diagramme de déploiement pour l'architecture matérielle et de diagramme de composants pour l'architecture logicielle.

Au début des années 2000, peu de formalismes offrent un pouvoir d'expression qui permet de prendre en compte cette notion de localité et seules des extensions de calcul de processus existent, sans pour autant disposer d'outils d'analyse. Il faut attendre quelques années pour disposer des calculs de Kells et autres définitions avancées de join calcul. C'est pour cette raison que les travaux que Fabrice Mourlin a initiés sur ce sujet avec la collaboration d'Andreea Barbu qui débuta son doctorat en 2000, consistent en l'association de deux approches formelles existantes. D'une part une algèbre pour le bas niveau, nommée Dpi calcul [12] et le Pi calcul d'ordre supérieur pour la description haut niveau.

3.2 Proposition d'une approche hybride

Afin d'obtenir une approche réaliste de nos spécifications, il nous a semblé essentiel d'adopter une approche hybride à deux niveaux, dont le plus bas soit aisé à projeter dans un monde opérationnel, c'est-à-dire dont la sémantique soit proche des standards réseaux existants. Par exemple, le protocole Internet IP [13] possède les caractéristiques connues : asynchrone, requête non ordonnée, point à point non bufférisée avec des pertes éventuelles [14]. Aussi, nous avons choisi d'exprimer dans une approche bas niveau les aspects point à point, non bufférisé, manque de fiabilité des messages, etc. De l'autre coté, d'autres aspects sont ignorés alors qu'ils sont très présents dans tous les protocoles importants tels que : le découpage de messages en paquets ou le contrôle de la transmission de message, etc. Bien entendu, la migration d'agents est une primitive de base de toute description bas niveau. Ainsi une telle primitive permet de décrire le transfert d'un agent d'un site à un autre.

L'appartenance de cette primitive au langage de spécification de bas niveau permet de concentrer une attention particulière sur la répartition des agents sur le réseau, leurs chemins de migration et ainsi offrir une description indépendante des sites traversés pour le haut niveau. La migration d'agent est accompagnée d'une autre primitive de création d'agents mais aussi d'envoi de messages entre les agents d'un même site ou encore de désignation d'actions locales à un site.

Mais tous les modèles évoqués jusqu'à présent ignorent la répartition spatiale du calcul. On voudrait pourtant pouvoir représenter fidèlement cette répartition. Par exemple, si nos cinq machines effectuent leurs propres calculs mais échangent certains de leurs résultats, et que l'on essaie de modéliser cette situation en pi calcul, rien ne pourra différencier les processus représentant les calculs se déroulant sur une machine de ceux s'exécutant sur l'autre.

Plusieurs calculs formels sont apparus pour raisonner sur de tels systèmes répartis. En particulier, les ambients mobiles [16] se concentrent sur la notion de localité, les réductions de ces entités étant des réagencements de l'emboîtement de telles localités. Plus récemment, Robin Milner a proposé les bigraphes [17] comme formalisme permettant d'unifier les ambients mobiles et le pi-calcul. Le langage Dpi-calcul est introduit par Matthew Hennessy et James Riely dans [15]. Ce calcul se distingue des autres en se posant comme une extension simple du pi calcul, dans laquelle tous les processus sont placés dans une localité, sans que les localités puissent être emboîtées. Il est ainsi beaucoup plus facile de modéliser des comportements dans ce formalisme que dans les ambients mobiles, par exemple. Le choix d'une extension minimale du pi calcul permet d'hériter de la riche littérature existante et de mettre l'accent sur les nouvelles questions portant sur la répartition, telles que les droits d'accès ou la gestion des localités.

Notre approche à deux niveaux (figure 2) s'appuie sur deux algèbres de processus. L'une pour le bas niveau fourni par les travaux de M. Hennessy concernant le pi calcul distribué [15], [18]. Une autre pour le haut niveau fourni par les travaux de R. Milner sur le pi calcul d'ordre supérieur [19]. Ces calculs formels permettent la définition précise d'agent de manière dépendante ou indépendante des sites sur lesquels ils se déplacent. Ainsi, des algorithmes répartis de gestions d'agents sur une topologie particulière de sites peuvent être décrits dans un premier temps. Ensuite par exemple, un algorithme de collecte de données par des agents mobiles peut être décrit de manière indépendante de la topologie physique du réseau. Les sémantiques opérationnelles de ces deux calculs fournissent une compréhension claire des comportements attendus et aident le concepteur de ces algorithmes dans les étapes ultérieures de l'étude d'un système multi agents mobiles.

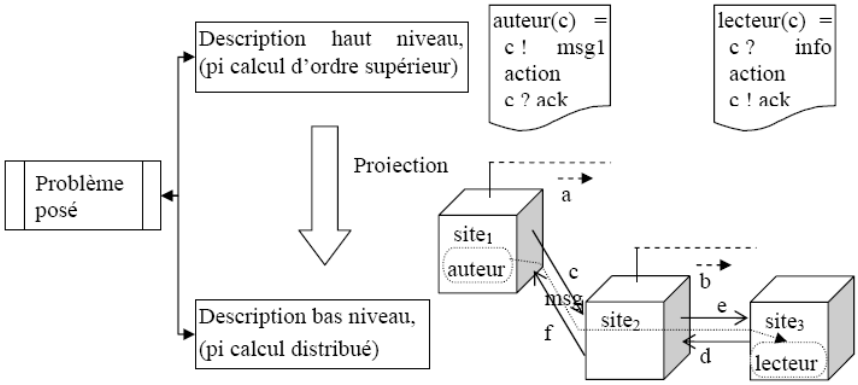

Figure 2 : deux niveaux de description

3.3 Choix d'un langage de spécification d'architecture

Les architectures évolutives sont répandues dans de nombreux domaines d'activités, depuis les applications Web au calcul numérique. La gestion de cette architecture au cours de son évolution est en revanche moins souvent abordée. Cette gestion est pourtant indispensable dans l'écriture de systèmes auto adaptatifs [20] ou auto correcteurs [21].

Cette volonté de séparer la description de l'architecture n'est pas nouvelle et des langages dédiés existent tels que ADL (Architecture Description Language) [23]. Une description ADL permet ainsi de schématiser une vue globale de l'application de la même manière qu'un plan d'architecte modélise une maison en décrivant les différentes pièces de celle-ci et la manière dont elles sont disposées et « communiquent » entre elles. Mais le langage de notre choix devait disposer d'un pouvoir d'expression suffisamment riche pour que les perturbations dues à son emploi se justifient en particulier sur la notion d'accès aux ressources locales où la signature d'une opération ne peut être suffisante.

3.3.1 Approche matérielle en Dpi

Afin de décrire les aspects matériels ou bas niveau, nous nous sommes orientées vers l'emploi du langage Dpi conçu par M. Hennessy [22]. Cette fois, la notion de site de migration est prise en compte en ajoutant la notion d'aptitude ou autorisation. Il est alors possible d'écrire des expressions sur les besoins de migration pour satisfaire une mission, autrement dit effectuer un certain parcours sur un réseau. Par exemple, un agent *collecteur* souhaite faire l'inventaire des agents *lecteur* présents sur un réseau. Son but est alors le parcours de tous les sites de ce réseau où des agents peuvent être présents. La description d'un algorithme de recherche de sites particuliers, afin de passer une fois et une seule par chacun des nœuds physiques, peut s'exprimer à l'aide de ce calcul. En revanche, l'activité précise entre le collecteur et un lecteur ou plutôt le métier propre de ce collecteur sera spécifié à l'aide du polyadique pi calcul.

Comme extension du langage pi calcul, ce langage permet de représenter la répartition du calcul, Dpi repose sur une notion de localités, ou de sites, de sorte que tous les processus sont annotés par la localité dans laquelle ils s'exécutent :

$site_1[lecteur]$: denote que l'agent *lecteur* est physiquement placé sur le site nommé $site_1$.

$site_1[lecteur] \| site_3[auteur]$: au niveau d'un réseau plus complexe, il est possible d'exprimer la mise en parallèle de processus localisés.

Dpi est un langage d'ordre supérieur, basé sur le pi calcul, qui permet d'exprimer le comportement des agents mobiles qui se déplacent entre les sites d'un réseau réparti. Si la syntaxe de ce langage est plus complexe, utilisée conjointement avec une spécification de type polyadique pi calcul, ce langage permet une description des scénarii de déplacements des agents sur un réseau de sites de réception d'agents.

À chaque instant, un des processus d'un système peut réaliser une étape de calcul interne ou bien interagir : une interaction met en jeu deux processus qui décident de se synchroniser et d'échanger un message. Pour avoir lieu, cette communication suppose qu'il existe un canal sur lequel un des deux processus émet un message tandis que l'autre processus écoute. Puisque la synchronisation entre deux processus est une opération complexe, Dpi impose que les communications soient locales, c'est-à-dire que les canaux sur lesquels elles s'effectuent soient eux aussi localisés et ne soient accessibles qu'aux processus s'exécutant dans leur localité. Ainsi, dans l'expression suivante :

$$site_2[lecteur \mid auteur] \| (new\ a : C)(site_5[moniteur] \| site_2[hôte \mid collecteur])$$

Les agents *hôte*, *collecteur*, *auteur* et *lecteur* s'exécutent sur le site nommé $site_2$ alors que l'agent *moniteur* s'exécute sur le site nommé $site_5$. Les agents *moniteur*, *hôte* et *collecteur* partagent un même canal logique privé nommé *a* de type *C*. Dans cet exemple, *auteur*, *lecteur*, *hôte*, *collecteur* et *moniteur* sont des agents traditionnels tels que l'exprime Robin Milner dans ses travaux de présentation de pi calcul d'ordre supérieur. Ils peuvent recevoir des informations et en émettre sur des canaux. Le type de ces canaux indique la nature des valeurs véhiculées.

Cette approche est totalement dynamique et un agent peut décider de migrer sur un site déjà connu ou nouvellement pris en compte. Ainsi :

$$site_2[(new\ site_1 : Site)\ with\ collecteur\ in\ a!\langle site_2 \rangle \mid e!\langle site_2 \rangle]$$

Cette expression décrit qu'à partir d'un site, nommé $site_2$, un nouveau site, nommé $site_1$ est pris en compte. Le code de l'agent, nommé *collecteur*, est placé sur ce nouveau site et le nom de ce site est communiqué sur les canaux *a* et *e*. Dans cette expression, les canaux *a* et *e* sont déclarés à un niveau supérieur autorisant leur utilisation dans cette expression.

Dans une telle spécification, il est important de déclarer initialement tous les sites connus avec les services qui y sont disponibles, afin de ne gérer les reconfigurations que lorsque cela est indispensable. De manière générale, cette déclaration de services correspond au type du site considéré telle que l'expression suivante :

$$site_1 = loc[c : w\langle collecteur \rangle,\ f : r\langle hôte \rangle, b : w\langle hôte \rangle]$$

L'identificateur $site_1$ correspond à un site physique où trois services sont disponibles. L'émission sur un canal *c* est considérée comme un service utilisé par l'agent *collecteur*, ainsi que la réception sur le canal *f* pour l'agent hôte. De plus, le service offert sur le canal *b* permet la remontée d'informations à destination depuis l'agent *hôte* vers le *moniteur*. Cette propriété peut être raffinée en précisant la nature de données émises. De telles contraintes de types peuvent parfois aboutir à des incohérences.

De manière identique aux sites, les canaux entre sites supportent un typage qui permet de limiter l'expressivité du message échangé. Ainsi, supposons la définition du canal *c* depuis $site_1$ conjointement à la précédente définition du type du site $site_1$: $c : w\langle loc[f : r\langle hôte \rangle] \rangle$

Tout d'abord cette expression est cohérente avec la déclaration du type de $site_1$, où le canal *c* était annoncé comme étant utilisé en émission par l'agent *collecteur*. Ce qui signifie par rapport à la figure 1, que lorsque cet agent aura émigré depuis $site_2$ vers $site_1$ il pourra disposer de ce service. De plus, cette déclaration ajoute que la nature de la donnée transportée sur *c* doit disposer d'un service *f* en lecture. Dans l'expression précédente, le receveur de l'information est mentionné, ce sera l'agent *hôte*. Une telle expression joue le rôle de contrainte sur le message émis. Dans notre cas, nous pouvons, à titre d'exemple, utiliser le canal *c* depuis $site_1$ de la manière suivante : $c\,!\,site_1$

Dans cette expression, les agents receveurs de l'information $site_1$ depuis le canal *c*, doivent bien entendu être aptes, sur leur sites, à recevoir sur un canal *f* (site où une déclaration duale de *c* a été faite). Enfin, cette information reçue leur permet d'utiliser le service *f* de $site_1$.

Une autre notation porte sur la migration d'agents en fonction des sites à parcourir. Par exemple, lorsque l'agent *collecteur* souhaite se déplacer depuis $site_2$ vers $site_1$, l'expression initiale de ce déplacement mentionne le canal support de ce déplacement ($goto_f$). L'évaluation d'une telle expression revient à utiliser le service *f* de $site_1$, avec comme message l'agent en cours de déplacement. Cette approche est reprise dans les travaux de M. Hennessy sur une évolution de Dpi nommé SafeDPi [22].

$$site_2\left[goto_f\, site_1.collecteur\right] \mid site_1\left[hôte\right] \rightarrow site_1\left[f\,!\langle collecteur\rangle \mid hôte\right]$$

Un agent doit désigner un canal *f* vers sa destination (ici le site nommé $site_1$) pour effectuer sa migration. Comme le montre la règle d'évaluation ci-dessus, cette expression revient alors à évaluer l'agent sur le site de destination. Un agent qui souhaite migrer ne pourra s'évaluer tant que le site de destination n'acceptera pas cette communication. Cela signifie que le site receveur doit supporter un terme, nommé ici *hôte*, dont l'évaluation permet d'aboutir à : $site_1\left[f\,?(x).x.hôte\right]$

Ce qui signifie que le site, nommé $site_1$, peut contrôler les arrivées des agents mobiles qui souhaitent lui rendre visite. Dans cette description, la répartition d'agent est ainsi dirigée par l'emploi d'une communication d'ordre supérieur. En outre, les canaux utilisés peuvent être typés afin de contrôler davantage les importations d'agents. Le typage des canaux est complété par la définition même des agents migrants.

Quand un agent migre, il peut être instancié à la réception grâce à des valeurs de type adéquat. Ceci procure davantage de contrôle et assure ainsi une meilleure gestion des agents. Ce cas d'instanciation dès la réception par le site receveur est d'ailleurs le plus fréquent, mais une facette de cet agent n'est pas encore prise en compte : les ressources locales utilisées.

Pour limiter les accès aux ressources locales par des agents mobiles, la notion d'agent typé est introduite. Celle-ci permet de décrire les besoins des agents au niveau local, c'est-à-dire par rapport au site qui l'accueille. Ces ressources incluent aussi les canaux utilisés par l'agent à partir de son arrivée sur le site [22]. Une expression de typage d'agent peut s'écrire sous la forme :

$$collecteurType\left[\begin{array}{l} g_1 : C_1 @ site_1, g_2 : C_2 @ site_2, g_3 : C_3 @ site_3, g_4 : C_4 @ site_4, \\ w_1 : W_1 @ site_1, w_2 : W_2 @ site_2, w_3 : W_3 @ site_3, w_4 : W_4 @ site_4 \end{array}\right]$$

Un agent de ce type peut ainsi utiliser au plus les canaux g_1 et w_1 placé sur le site $site_1$ avec la possibilité de véhiculer respectivement les données décrites par le type C_1 et W_1, ainsi que g_2 et w_2 de $site_2$, etc. C_1 est le type de $info_1$ c'est-à-dire de l'information lue sur $site_1$ (voir

figure 1) et W_1 est le type des informations de contrôle à destination de l'agent *hôte* de $site_1$. Cette notation permet de différencier les canaux locaux par le mot clé *here* telle que l'entrée suivante : *c :C@here* . Bien entendu, cette définition de type est incomplète car elle nécessite d'ajouter les autres canaux utilisés. Une telle expression de type offre de nombreuses possibilités de contrôle lorsqu'elle apparaît dans la déclaration de canal entre deux sites. Lors de l'arrivée d'un agent, par exemple, considérons la déclaration du service d'importation d'agent vu depuis le site d'accueil $site_1$ qui n'accepte que des agents de type *collecteurType*:

$$PortType_{site_1}\left(f : r\langle CollecteurType\rangle \rightarrow process\left[g_1 : r\langle C_1\rangle @ here, warn_1 : w\langle W_1\rangle @ site_1\right]\right)$$

PortType représente la stratégie d'importation de l'agent *collecteur*. Ainsi un tel agent arrivant sur ce site peut seulement être instancie à partir d'un canal *f* qui autorise l'importation d'agent de type *CollecteurType*. En outre, l'agent ainsi construit (membre droit de la transition), est seulement autorisé à lire sur un canal local g_1 des données de type C_1 et écrire sur un canal particulier nommé $warn_1$ spécifique au site $site_1$. Dans notre exemple, cela signifie qu'un agent local, par exemple un agent *Host* du $site_1$, peut émettre un tel agent.

Tandis qu'avec une définition d'agent comme suit les contraintes sont attribuées différemment:

$$PortType_{site_1}\left(f : r\langle CollecteurType\rangle @ site_1 \rightarrow process\left[g_1 : r\langle C_1\rangle @ here, warn_1 : w\langle W_1\rangle @ site_1\right]\right)$$

Le site $site_1$ peut recevoir un agent mobile de type *CollecteurType* uniquement sur ce canal localisé sur $site_1$. Une fois migré, cet agent pourra lire des données de type C_1 depuis un canal g_1 local mais aussi écrire sur $warn_1$. Une telle déclaration d'agent importé n'est pas obligatoirement utile qu'au seul $site_1$, mais il peut aussi être employé sur d'autres sites afin de partager une même aptitude.

Dans les exemples précédents, les définitions d'importation d'agents mobiles et de sites d'accueil étaient faites conjointement. Il peut être utile de définir les notions séparément comme cela est le cas pour un serveur d'agents mobiles avec un canal qui permet l'arrivée d'un agent d'un type donné (figure 1 $site_5$). Des contraintes sur les types permettent de limiter les sites appartenant à l'ensemble *Site* de tous les sites déjà définis :

$$serveur_{site_5}\left(site : Site\right) PortType\left(\begin{array}{l} f : r\langle CollecteurType\rangle @ site \rightarrow \\ process\left[g : r\langle C\rangle @ here, warn : w\langle W\rangle @ site\right] \end{array}\right)$$

Cette définition d'agent est paramétrée par le site, où sont utilisées des données de type *C*. Le serveur gère des agents qui peuvent écrire des informations de type *W* sur un canal *warn* sur les sites choisis par le receveur de l'agent. Quel que soit le site de réception, il sera possible pour l'agent importé de pouvoir accéder à une ressource locale en lecture, nommée *g*. Cette définition de type paramétré est essentielle dès que l'on traite un cas d'étude où figure au moins un serveur d'agents mobiles. L'inconvénient premier réside dans la contrainte forte imposée par le serveur. En effet, un client qui souhaite émettre des données correspondant au type paramétré, doit respecter la contrainte imposée par le serveur. Ainsi de nouveaux besoins peuvent être exportés par le serveur vers ses clients et cela peut amener une reconfiguration de l'ensemble des définitions d'agents. Ces contraintes ont pour but de ne pas frauder auprès du serveur d'agents et fournir de ce fait une protection plus grande. Il ne faut pas se tromper sur leur rôle dans une spécification dédiée aux aspects matériels.

Cette rapide présentation d'une description matérielle fait apparaître plusieurs aspects. Tout d'abord la difficulté de rédaction des spécifications d'architecture dues en partie à la richesse du langage employé mais aussi à l'étendue des caractéristiques à modéliser. En effet, ce rapide cas d'étude montre qu'il est essentiel de séparer les connaissances sur les

architectures des connaissances métier propres. Nous avons choisi de décrire l'architecture par l'emploi d'une algèbre de processus adaptée à ce domaine pour l'aspect composable avec l'approche logicielle de la section 3.2.4 mais aussi pour l'expression de la mobilité.

Le cadre des descriptions faites avec Dpi est large car l'architecture décrite comporte clairement deux aspects : matériel et architecture. Les aspects matériels montrent la disposition physique des sites qui composent le système et la répartition des agents sur ces sites. Les ressources matérielles sont caractérisées par certaines aptitudes ou contraintes. Les matériels sont connectés entre eux, à l'aide de supports de communication souvent appelés port. La nature des lignes de communication et leurs caractéristiques peuvent être précisées. Les descriptions de sites peuvent montrer des instances d'agents (un agent précis tel que *moniteur*), ou des classes d'agents (tel que *CollecteurType*).

Les aspects architecture décrits avec Dpi portent sur l'architecture fonctionnelle qui doit être mise en place sur les sites. La séparation est ainsi établie avec l'architecture technique. L'architecture fonctionnelle est en rapport avec le domaine auquel elle s'applique. En d'autres termes, elle a pour objectif d'aider à réaliser le problème posé. Alors que l'architecture technique porte sur la manière d'implémenter l'architecture fonctionnelle. Ce sera en partie le sujet de la section suivante.

Notre exemple précédent porte sur la collecte d'informations sous la surveillance d'un observateur unique. Il est clair que cette situation est fréquente dans le domaine des algorithmes de contrôle. Des patterns d'architecture doivent pouvoir être mis en place rapidement dès le début d'un projet. Le langage formel Dpi permet de définir de tels patterns et ainsi mettre en place une architecture qui sera utilisée dans plusieurs autres spécifications logicielles. Notre description des types d'agents *hôte* et *serveur* illustre cet aspect avec la définition d'interface de comportement imposée par le serveur sur les agents mobiles.

A la suite de ce constat, il semble important d'insister de nouveau sur le rôle des spécifications d'architecture faites en Dpi, afin de cerner avec précision ce qui doit figurer dans une telle description et surtout ce qui ne doit pas y être. De nombreux travaux tentent de mettre en place une algèbre idéale permettant de modéliser tout un système informatique. Il est évident de devoir dissocier les aspects pour une meilleure lisibilité et surtout une gestion plus précise des spécifications dans le temps.

3.3.2 Ecriture de spécification matérielle : pattern d'architecture

La mise en place d'une architecture correspond avant tout à la définition d'une organisation de base du système dans son ensemble à laquelle s'ajoutent des règles de gestion des composants qu'il contient et qui respectent cette organisation. Ces composants interagissent entre eux. Une part importante du travail d'un architecte est dévolue aux manières de découper une application en composants mis en relation pour ensuite comparer les résultats, les propriétés, les performances, etc. En partitionnant une application, l'architecte affecte à chaque composant des responsabilités ou contraintes. Celles-ci définissent les aptitudes que doivent avoir les composants pour réussir les futures interactions.

Plus concrètement, les réalisations de ces contraintes sont les tâches que devront réaliser les composants au cours de leur exécution. Au final, chacun d'eux aura son propre rôle à jouer sur une scène qui respecte les choix de navigation de l'architecture.

De manière générale, un principe de base de toute spécification d'architecture est de minimiser les dépendances entre composants et de créer ainsi un couplage faible entre les

différents morceaux de logiciels s'exécutant dans l'architecture. C'est le cas de l'exemple de la section précédente. Les composants sont peu dépendants mais ont une forte cohérence. En éliminant les dépendances inutiles, cela permet de limiter la portée des changements de manière locale à un site.

Des descriptions textuelles sont fort nombreuses et parfois différentes, l'atout de la formalisation avec Dpi est d'afficher une seule compréhension. Ainsi, nous pouvons formaliser les contraintes de base de la manière suivante dans une version locale au $site_1$:

$$system = newchan\ request : rw\langle T_{req}\rangle\ in\ (server(request) \mid client(request))$$

$$server(req) = site_1[*\ (req?((msg, resp) : T_{req}.w\langle T_{resp}\rangle).business(req, resp).resp!\langle ok\rangle)]$$

$$client(req) = site_1[newchan\ response : rw\langle T_{resp}\rangle\ in\ (req!\langle info, response\rangle.response?(x : T_{resp}).activity(x))]$$

Cette description du pattern client serveur sur un même site permet de visualiser l'envoi de message depuis le client vers le serveur. Le traitement métier fait par le serveur (processus business) est à définir lors de l'application du pattern et l'acquittement de la requête client. L'ensemble *C* représente l'ensemble des canaux utilisés dans cette expression tels que request et response. Chaque canal est typé par les opérations qu'il supporte et la nature des données transmises, T_{req} est le type du message *msg* reçu par le serveur et T_{resp} le type des données reçues par le client. Le serveur a un comportement intrinsèquement parallèle pour supporter plusieurs clients en parallèle. Sa première action est de recevoir une donnée du client et le support pour répondre au client. Ce canal ne doit lui servir qu'à émettre une réponse et donc seule la capacité d'écriture sur ce canal est acquise par le serveur. De plus, afin de contrôler la nature de l'échange, l'écriture sur le canal reçu est limitée aux données de type T_{resp}.

4 Relation entre les niveaux de spécifications

La donnée de deux niveaux de spécifications nécessite le besoin de contrôles supplémentaires entre ces représentations. L'idée maîtresse de ces contrôles est d'assurer qu'il est possible d'utiliser le logiciel sur le matériel. Autrement dit, les contraintes imposées par le niveau matériel sont elles respectées par les descriptions logicielles.

Nous avons défini des types dans la description matérielle pour les sites (tels que $site_1$, $site_2$, etc), pour chaque port de communication (tel que $PortType_{1,site_1}$, etc), ainsi que pour certains agents placés sur ces sites (tel que *CollecteurType*).

Le typage des agents correspond à des aptitudes que doivent posséder les agents qui sont déclarés de ce type. Dans le cadre de cette étude, ces aptitudes sont considérées comme un contrat que doit remplir l'agent. Aussi la description logicielle de chaque agent est analysée pour assurer qu'elle répond à son contrat. Cette phase d'introspection est effectuée lors de l'étude du plongement de la description logicielle.

En premier, la spécification doit respecter le type de l'agent lui-même puis, le type du site sur lequel il est placé. Enfin, pour toute communication sur un port, c'est-à-dire un canal externe au site, la nature de ces échanges est validée (destination, nature des données, sens de parcours, propagation des aptitudes).

Une même spécification d'agent peut ainsi se déployer sur plusieurs sites de type différent si les contraintes minimales de chacun d'eux sont respectées. Dans notre exemple, chaque site excepté $site_5$ supporte un agent *hôte*. Aussi la spécification de cet agent devra être contrôlée pour chacun des sites de placement. En outre, par ce que la mobilité de nom

est au cœur même des calculs que nous utilisons, il est essentiel d'effectuer une nouvelle phase de validation des contraintes après chaque réception d'information. Ceci est d'autant plus important quand le typage des communications n'est pas suffisamment complet.

Cette étude est la base de plusieurs applications dont le point de départ reste les spécifications de système d'agents mobiles pour aboutir à une autre représentation telle qu'un squelette de code ou encore une famille de tests pour valider le squelette de code. Ces travaux nous ont conduits à considérer les spécifications de nos cas d'étude comme une première perspective qui permet le pilotage de la suite du projet.

5 Conclusion

Les choix présentés dans ce document sont le résultat de plusieurs tentatives tant sur les moyens de descriptions formelles que sur les solutions techniques pour la réalisation. Ainsi, une étude a été menée sur l'emploi d'un serveur d'EJB qui montre les possibilités de reconfiguration des ressources mais souligne la nécessité de dimensionner le nombre de composants avant de réaliser toute simulation. Des travaux restent à entreprendre en particulier pour le portage d'applications traditionnelles vers la mobilité d'actions.

Le besoin de confronter ces résultats à la réalité est une constante sur l'ensemble de nos travaux. Ceci a guidé la réalisation d'une plate forme de spécification et de développement qui représente l'illustration de chacun des points de ce document et poursuit la démarche pour aboutir à la notion de cas d'utilisation.

6 Références

[1] Wilhelm R. Rossak, Peter Braun, "Mobile AgentsBasic Concepts, Mobility Models, and the Tracy Toolkit", Elsevier, ISBN 1558608176, publié 2005.

[2] Jefferey J. P. Tsai, Lu Ma, "Security Modeling And Analysis of Mobile Agent Systems", ISBN 1860946348, publié 2006, Imperial College Press.

[3] Mark D'Inverno, Michael Luck, "Understanding Agent Systems", Springer, ISBN 3540407006, publié 2004.

[4] Reza B'Far, collaborateur Roy T. Fielding, "Mobile Computing Principles Designing and Developing Mobile Applications with ..." ISBN 0521817331, publié 2005, Cambridge University Press

[5] Jacques Ferber, "Les systèmes multi-agents : Vers une intelligence collective", InterEditions, 1995, ISBN 2-7296-0572-X.

[6] J. Ferber, "Multi agent systems for telecommunications: from objects to society of agents", Networking 2000, 2000, Paris

[7] Eric Horlait, "Mobile Agents for Telecommunication Applications », ISBN 1903996287, publié 2003, Kogan Page

[8] Hai. Zhuge, Geoffrey C. Fox, "Grid and Cooperative Computing" Gcc 2005 4th International Conference Computer Science, collaborateur Hai Zhuge, Geoffrey C. Fox, publié 2005, Springer, ISBN 3540305106

[9] Joong-Ba Y. et al. Casting mobile agents to workflow systems: On performance and scalability issues Database and Expert Systems Applications 12th International Conference, 2001, 2113, 254-263.

[10] Peter Sewell, Pawel T. Wojciechowski, Benjamin C. Pierce, "Location independent communicaiton for mobile agents: a two level architecture", Lecture notes in computer science (Lect. notes comput. sci.) ISSN 0302-9743, ICCL'98 workshop, Chicago IL , ETATS-UNIS (13/05/1998) 1973, vol. 1686, pp. 1-31, [Note(s) : VII, 143 p.,] (2 p.) ISBN 3-540-66673-7 ;

[11] Paul Evitts Textbook Binding, "A UML Pattern Language (Macmillan Technology Series)", editeur: Sams; 1st edition (February 2000), ISBN-10: 157870118X, ISBN-13: 978-1578701186

[12] A. Francalanza, M. Hennessy, “Location and link failure in a distributed pi-calculus”, Computer Science Report 2005:01, University of Sussex, 2005

[13] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6), Specification", RFC 2460, December 1998.

[14] Crawford, M., Thomson, S., and C. Huitema. "DNS Extensions to Support IPv6 Address Allocation and Renumbering", RFC 2874, July 2000.

[15] Matthew Hennessy , James Riely, “Resource access control in systems of mobile agents, Information and Computation”, v.173 n.1, p.82-120, February 25, 2002

[16] Luca Cardelli and Andrew D. Gordon: “Mobile Ambients”, in Foundation of Software Science and Computational Structures, volume 1378 of Lecture Notes in Computer Science, pages 140-155, Springer Verlag, 1998

[17] O. H. Jensen & R. Milner “Bigraphs and mobile processes (revised)”, Tech. Report UCAM-CL-TR-580, University of Cambridge, Computer Laboratory, 2004.

[18] M. Hennessy , M. Merro , J. Rathke, “Towards a behavioural theory of access and mobility control in distributed systems”, Theoretical Computer Science, v.322 n.3, p.615-669, 6 2004

[19] Milner, R. “ The Polyadic pi-Calculus: A Tutorial “. Dans BAUER, F. L., BRAUER, W., et SCHWICHTENBERG, H., éditeurs, Logic and Algebra of Specification, Proceedings of International NATOSummer School

[20] B. Schmerl and D. Garlan, “Exploiting architectural design knowledge to support self-repairing systems”, In Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2002), pages 241–248. ACM Press, 2002.

[21] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. “An architecture-based approach to self-adaptive software”, IEEE Intelligent Systems, 14(3):54–62, 1999.

[22] Matthew Hennessy, Julian Rathke and Nobuko Yoshida. “SafeDpi: A Language for Controlling Mobile Code”, (Extended abstract) Proc. of 7th International Conference, Foundations of Software Science and Computer Structures (FoSSaCs 2004).

[23] N. Medvidovic. “ADLs and dynamic architecture changes”, In Joint Proc. of the 2nd Int. Software Architecture Work. (ISAW-2) and Int. Work. on Multiple Perspectives in Software Development (Viewpoints ’96) on SIGSOFT ’96 Workshops, pages 24–27. ACM Press, 1996.