AGNI : encore des agents mobiles !

Mudumbai Ranganathan[*]—Laurent Andrey[*,]—Virginie Galtier[*,**]— Virginie Schaal[*]**

[*] *National Institute of Standards and Technology*
820 W. Diamond Ave
Gaithersburg, MD 20899, U.S.A.
{mranga,virginie.schaal}@nist.gov

[**] *Loria*
BP 239
F54506 Vandoeuvre-lès-Nancy, France
{Laurent.Andrey,Virginie.Galtier}@loria.fr

RÉSUMÉ. AGNI est une plate-forme à agents mobiles implantant une mobilité restreinte : un processus, "un agent" ne peut se déplacer que lorsqu'il est inactif. Par contre, le système privilégie les communications entre agents en assurant la transparence de leur position. Une application bâtie au-dessus d'AGNI est vue comme un ensemble de noms qui réagissent à la réception de messages et à l'occurrence d'un certain nombre d'autres événements. Les actions exécutées lors de l'occurrence d'un événement sur un nom sont écrites en TCL. Les applications envisagées au-dessus de cette plate-forme sont -l'enregistrement/réactionnement d'applications distribuées pour des tests de fonctionnement de réseau, -l'aide au déploiement et à l'application de test, -la décentralisation de certaines fonctions de gestion de réseau ou service.

ABSTRACT. AGNI is a mobile agents platform offering a limited mobility: a process "an agent" can only move when idle. The system rather emphasizes on inter-agents communications where position transparency is total. In one way an application built on top of AGNI is a set of names that are reacting to message reception and occurrences of some other special events. Actions executed on events are TCL scripts. Target applications for AGNI are –record/replay of distribued applications for network stress tests, -support for distributed applications tests execution, -distribution of some network or service management functions.

MOTS-CLÉS: *agent mobile, code mobile, système distribué.*

KEY WORDS: *mobile agent, mobile code, distributed system.*

1. Introduction

Le *National Institute of Standards and Technology* (NIST) a développé AGNI une plate-forme supportant une notion restreinte de mobilité des activités (***agents***) au dessus d'un réseau IP. Cette mobilité n'est pas restreinte dans l'espace, mais dans les moments où elle est effectivement prise en compte par le système. En effet, la demande de migration d'un agent d'une place à une autre du réseau (***location*** dans la suite) n'est qu'un événement parmi les autres événements que perçoit un agent (initialisation, erreurs, réception d'un message…). La migration est donc sérialisée à la suite des autres opérations faites en réponse d'événements. Ce “ modèle ” de mobilité (mobilité de serveur) utilisé dans [GMOd, JVS 95, SBH 96, VEN 97,OSVoy] est plus simple et léger à mettre en œuvre que la mobilité de thread supportée dans [CAR 95,GRA 96,RASS 97] où après l'exécution d'une instruction de migration par un thread, l'exécution de celui-ci continue exactement à l'instruction suivante mais sur la nouvelle *location*. Le premier cas nécessite juste la distribution du code et éventuellement la gestion d'une partie des variables de l'agent comme mobile : AGNI introduit une notion de valise (***briefcase***) dans laquelle un agent place des variables. Le deuxième cas nécessite l'extraction, la sérialisation, et la génération du contexte complet d'exécution d'un thread (pile, tas…) opérations difficiles à implanter (modification profonde des interprètes ou machines virtuelles) et d'une exécution lourde. Certains systèmes vont jusqu'à une mobilité assurant la transparence sur les fichiers, les entrées sorties et toutes autres ressources utilisées par le thread déplacé.

La particularité d'AGNI est sa notion de base : le “point de communication nommé et mobile” ou *mobile stream* appelé ***Mstream*** dans la suite de l'article. C'est en fait une approche très pragmatique comparable à l'intégration d'un système de fichiers dans un système d'exploitation. En effet une partie importante du code d'une application correspondant à des entrées/sorties dont le système cherche à masquer l'implantation (table de montage, correspondance fichiers spéciaux/device, correspondance noms/inodes sur UNIX). D'un autre côté l'utilisateur a un certain contrôle sur les ressources gérées par le système de fichiers : droit d'accès, déplacement, copie… Dans le cas d'AGNI la première tâche de l'utilisateur est la création des noms globaux qu'il veut utiliser. Les noms sont arbitraires et indépendants des *locations*. Après ouverture d'un *Mstream* l'utilisateur peut y envoyer des messages ou associer une action (script TCL[S]) à effectuer en cas de réception d'un message. La “sémantique” des noms et des événements est de la responsabilité de l'application. Par exemple des programmes extérieurs existants peuvent être instrumentés pour générer des messages vers des noms de la plate-forme en cas d'événements comme : une erreur système (modification de *syslog*), ou un événement d'une interface graphique. Le contrôle qui peut être fait sur un nom est principalement sa position. Le déplacement d'un nom peut être demandé par des programmes externes connectés à la plate-forme, par un script associé à lui-même ou à un autre *Mstream*. La décision du déplacement (réduction de la latence, arrêt d'une *location*) est de la responsabilité de l'utilisateur ou d'une partie de ses codes. Introduisons tout de suite une propriété importante d'un *Mstream M* :

> Pour tout émetteur *m* donné (*Mstream* ou programme externe) : l'ordre de consommation des messages venant de *m* par *M* est le même que l'ordre d'émission à partir de *m*.

La section 2 présente un exemple complet de l'utilisation d'AGNI. La section 3 donne plus de détails sur son utilisation et son implantation. La section 4 donne un aperçu des réalisations faites sur AGNI. La section 5 situe et compare ce travail avec d'autres systèmes. Nous finissons par une conclusion et quelques idées sur le futur d'une telle plate-forme.

2. Exemple introductif

Nous allons implanter une version très simplifiée d'un forum de discussion interactif à la irc/chat au-dessus d'udp-tcp/IP sans disposer de multicast. Chaque participant peut à tout moment envoyer un message à tous les autres participants. Nous imposons de plus que **tous les participants voient les messages dans le même ordre**. Du point de vue de l'implantation nous allons essayer de limiter la latence de l'écho sur l'utilisateur le plus bavard.

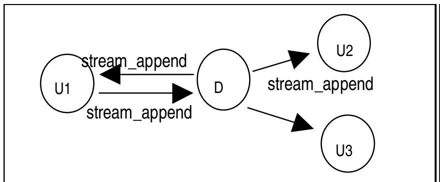

Figure 1.a. Structure d'un forum interactif

```
stream_create D
```

Figure 1.b. Création du distributeur central

```
set my_id [get_uid]
stream_open U$my_id                              c1
stream_move U$my_id [current_location]           c2
register_agent U$my_id {} {                      c3
    on_stream_append { puts stdout $argv }       c4
}
```

Figure 1.c. Affichage des messages

```
stream_open D                                    d1
while { 1 } {
    stream_append D [gets]                       d2
}
```

Figure 1.d. Gestion des entrées d'un utilisateur

```
stream_open D                                            e1
register_agent D { U1 U2 U3 } {                          e2
    add_briefcase bUsers                                 e3
    set bUsers $argv                                     e4
    foreach u $bUsers {                                  e5
        set userCount($u) 0
        stream_open $u                                   e6
    }
    set totalCount 0                                     e7
    on_stream_append {                                   e8
        incr totalCount                                  e9
        incr userCount([get_sender_location])            e10
        foreach u $bUsers {                              e11
            stream_append $u $argv
        }
        if { $totalCount == 50 } {                       e12
            stream_relocate \
                    [stream_location \
                        [most_talkative $userCount]]
        }
    }
    on_stream_relocation {                               e13
        foreach u $bUsers {
            set userCount($u) 0
        }
        set totalCount 0
    }
}
```

Figure 1.e. Distributeur central

Pour simplifier nous supposons qu'il n'y a que 3 participants appelés 1, 2 et 3. Nous allons créer un *Mstream* U_i pour chaque utilisateur. Ces *Mstreams* seront positionnés sur les machines de leurs utilisateurs respectifs. Ce *Mstream* affichera les messages du forum à son utilisateur. Les messages postés seront en fait envoyés au *Mstream* “distributeur”, *D*, qui effectuera la distribution vers les utilisateurs. En d'autres termes *D* est le serveur de ce forum. La figure 1.a résume cet agencement.

Détaillons maintenant les éléments à programmer. Après avoir créé une *location* sur chacune des machines des utilisateurs, on peut créer la *Mstream* correspondant au distributeur central D en exécutant le code de la figure 1.b depuis n'importe quelle *location*.

Sur chaque machine utilisateur il faudra ensuite exécuter un script formé du fragment TCL de la figure 1.c puis de celui de la figure 1.d. La ligne (#c.1) crée le *Mstream* de l'utilisateur (on suppose que la fonction [`get_uid`] retourne le numéro de l'utilisateur), puis la ligne (#c.2) assure son positionnement sur la location où s'exécute le script. La ligne (#c.3) ajoute une action à exécuter chaque fois que le *Mstream* reçoit un message. Ici c'est la simple ligne (#c.4) affichant le dernier message reçu. Celui-ci est placé dans la variable “conventionnelle ” `argv` par la plate-forme. La ligne (#d.1) indique que l'on va envoyer des messages à D, en d'autres termes que l'on veut utiliser le serveur de forum *D*. La ligne (#d.2) envoie une chaîne lue au clavier vers D.

Bien entendu, il faut démarrer le serveur. Pour cela il suffit d'exécuter le script de la figure 1.e sur la location de son choix. Ce script ouvre d'abord le *Mstream* D représentant le distributeur central (#e.1), puis y enregistre un agent répondant à 2 types d'événement (#e.2) `on_stream_append` et `on_stream_relocation`. Dans cet exemple un seul agent est attaché à *D*, d'autres pourraient l'être.

Les lignes #e.3 à #e. 7 représentent la création d'un interprète TCL réservé à l'agent D qui pourra accéder à la liste des utilisateurs passée dans la variable prédéfinie `argv` (#e.2). La ligne #e.3 indique à la plate-forme que la variable `bUsers` doit se déplacer en même temps que l'agent. `bUser` est ensuite initialisée par la liste des utilisateurs précédemment passée en paramètre (#e.4). Les lignes suivantes définissent et initialisent des variables locales à l'interprète utilisées pour stocker les informations nécessaires pour décider de la relocation de *D*. Le tableau `userCount` comptabilise le nombre de messages envoyés par chaque utilisateur depuis la dernière relocation, l'index est en fait le *Mstream* associé à l'utilisateur (#e.5). La variable `totalCount` (#e.7) est la somme des messages reçus par le distributeur depuis sa dernière relocation. Pendant cette phase d'initialisation l'interprète déclare aussi qu'il va certainement communiquer avec les agents utilisateurs (#e.6).

La deuxième partie (#e.8) de la définition de l'agent du distributeur est la définition du traitement des messages reçus déjà présenté dans le cas des agents utilisateurs. Les compteurs y sont mis à jour (#e.9 et #e.10). Le message est distribué vers chacun des utilisateurs, y compris l'émetteur (#e.11). Tous les 50 messages une action de reconfiguration est faite : une demande de relocation du distributeur sur la *location* de l'utilisateur qui a envoyé le plus de messages est faite (#e.12). Le code très simple de la fonction `most_talkative` qui retourne l'index de l'élément le plus grand du tableau passé en paramètre n'est pas explicité. La fonction `stream_location` est fournie par la plate-forme et retourne la *location* courante d'un *Mstream*. La dernière partie de la définition de l'agent pour *D* (#e.13) permet de préciser les actions à faire lorsqu'un agent est relogé sur une *location*. Le fragment de code fourni après `on_stream_relocation` est exécuté chaque fois que l'agent est remis sous le contrôle d'un interprète. Ici le code remet à 0 les variables locales à l'interprète.

3. Vue détaillée du système et son utilisation

3.1. Architecture d'exécution

L'élément de base pour l'exécution de la plate-forme AGNI est un démon à lancer sur chaque hôte participant à l'application distribuée : le **démon d'agents**. Il correspond à l'implantation de la notion de *location* définie dans l'introduction. Dans l'implantation naïve actuelle un autre démon, **le contrôleur du système,** doit être lancé en premier sur une machine et un port tcp donné. Ce contrôleur maintient un certain nombre d'informations consultées en dernier ressort par les démons d'agents : principalement les positions des *Mstreams*. Le couple *<adresse IP, n° de port>* correspondant à un contrôleur identifie une occurrence de la plate-forme et est donc un paramètre à fournir au lancement d'un démon d'agents. Une fois lancé un démon d'agents ajoute une nouvelle *location* dans le système, cette *location* se voit attribuer un identifiant unique (un entier).

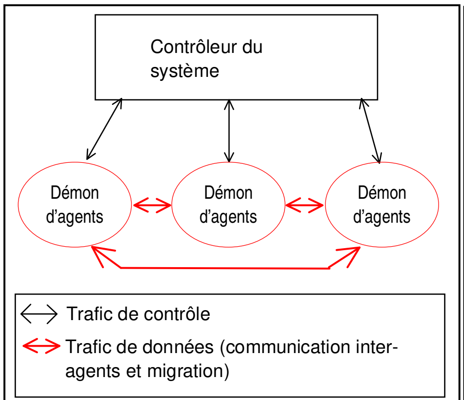

Figure 2. Organisation du système

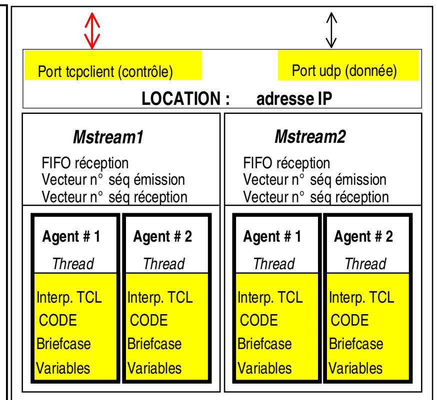

Figure 3. Organisation d'un démon d'agents

Un démon d'agents instancie un interprète et son thread d'exécution pour chaque agent s'étant exécuté au moins une fois sur la *location* qu'il représente. Cet interprète contient toutes les variables mobiles ou non et le code utilisé par un agent. Il maintient aussi une connexion tcp vers son contrôleur pour y router le trafic de contrôle (indication de déplacement, instanciation d'agent…). Il maintient également des informations relatives aux communications inter-agents qui ne passent évidemment pas par le contrôleur, l'idée étant d'obtenir l'équivalent d'une demi-connexion tcp entre un émetteur et un *Mstream* récepteur quand ceux-ci sont immobiles suffisamment longtemps. Un cache du code des agents est aussi implanté afin que la migration d'un nom se résume le plus possible au transfert des *briefcases* et à la mise à jour des informations de position.

La figure 2 résume la disposition des différents démons et la figure 3 l'organisation interne d'un démon d'agents.

3.2. Utilisation et fonctionnalités

Nous allons maintenant détailler certaines fonctionnalités et leur usage du point de vue du programmeur.

3.2.1. Mstreams et agents

Pour créer un *Mstream* l'utilisateur doit avant tout utiliser la commande :

```
stream_create stream_name [-authscript resctl_script]
```

L'option –`authscript` sera discutée plus loin.

Pour déclarer son intention d'utiliser un *Mstream*, le programmeur appelera la commande :

```
stream_open stream_name [-ordering_guarantee FIFO|UNORDERED]
```

L'option –`ordering_guarantee` permet de relâcher la propriété d'ordre des messages explicitée dans l'introduction (section 1).

Pour ajouter un nouvel agent sur un *Mstream*, préalablement ouvert, l'utilisateur dispose de la commande :

```
register_agent stream_name arg {script
    [on_stream_append script]  [on_stream_relocation script]
    [on_stream_failure script] [on_agent_exit script] }
```

Cette commande retourne en cas de succès un entier qui identifie le nouvel agent dans le contexte du *Mstream* auquel il est attaché. Elle peut être utilisée plusieurs fois pour un même *Mstream*. On voit bien que finalement un agent est un jeu de 5 scripts TCL déclenchés sur 5 événements. Les 3 premiers, les plus utiles, ont été présentés dans l'exemple introductif (section 2). L'événement lié à `on_stream_failure` correspond à la relocation automatique d'un *Mstream* en cas d'arrêt de la location courante[1]. L'événement `on_agent_exit` permet à un agent de relâcher les ressources locales qu'il a éventuellement allouées sur les différentes *locations* qu'il a visitées (fichiers temporaires, processus externes…) lorsque qu'il est détruit. Ces 5 scripts sont le seul moyen de spécifier le comportement d'un agent qui est donc un **objet purement réactif à des événements externes**. Pour leur écriture l'utilisateur dispose des fonctions habituelles de TCL y compris la possibilité de charger des extensions, ainsi que des appels liés à AGNI. Ces derniers ne peuvent être disponibles que pour un événement donné. Nous avons déjà vu `add_briefcase, stream_append, stream_location, stream_relocate`, et `stream_sender_location` dans l'exemple introductif ; donnons en quelques autres : `agent_exit` permet à un agent de se suicider, `stream_name` permet à un agent de connaître le nom de son *Mstream* propriétaire, `get_peer_map` retourne la liste des locations existantes. Il faut noter que `stream_create`, `stream_open` et `register_agent` sont parfaitement utilisables dans les scripts d'événements (sauf pour `on_agent_exit`) : un agent peut créer nouveau *Mstream* et y attacher un agent.

Il y a moyen de modifier la manière dont un *Mstream* donné répercute les 5 types d'événements vus précédemment vers ses agents. C'est le rôle du script qui peut être attaché à un *Mstream* lors de sa création grâce à l'option `-authscript`[2]. Ce script permet de filtrer des événements ou opérations relatifs au *Mstream* lui-même : ouverture, relocation, ajout d'un agent... La structure de ce script ressemble

---

[1] Un service de sauvegarde persistante de la *briefcase* limite également les effets d'une *failure*.

[2] Pour *authentication script*, en effet ce script a initialement été introduit pour contrôler quelle était la source d'un événement, puis éventuellement de le supprimer avant toute action.

à celle des paramètres de `register_agent` : un certain nombre de couples <*nom d'événement, actions TCL à exécuter*> (10 possibilités au total). Il existe en fait un thread/interprète supplémentaire par *Mstream* et par *location* pour exécuter ces actions de contrôle. Il peut être vu comme celui d'un agent "caché" que nous appellerons ***Mstream contrôleur*** dans la suite.

L'utilisateur peut entre autres ajouter un fragment TCL exécuté chaque fois qu'un `stream_open` est opéré quelque part sur la plate-forme pour le *Mstream* concerné. Dans ce cas le code TCL doit signifier son acceptation ou son refus au *Mstream contrôleur*.

Un cas courant est le contrôle **avant propagation** des événements `on_stream_append`, `on_stream_relocation` et `on_stream_failure` aux agents "réguliers". La commande AGNI `schedule_agent` *agent_id* permet au *Mstream contrôleur* de décider quels agents réguliers verront effectivement l'événement courant.

Il est même possible d'effectuer des actions **après** déclenchement des scripts des agents réguliers en associant des actions à l'événement `post_process`. Pour l'écriture de ce code l'utilisateur dispose de la fonction `get_current_event` pour connaître quelque type d'événement vient d'être traité par les agents réguliers.

La figure 4 donne une idée de l'application de ce contrôleur dans le cas de la réception d'un message pour un *Mstream* donné.

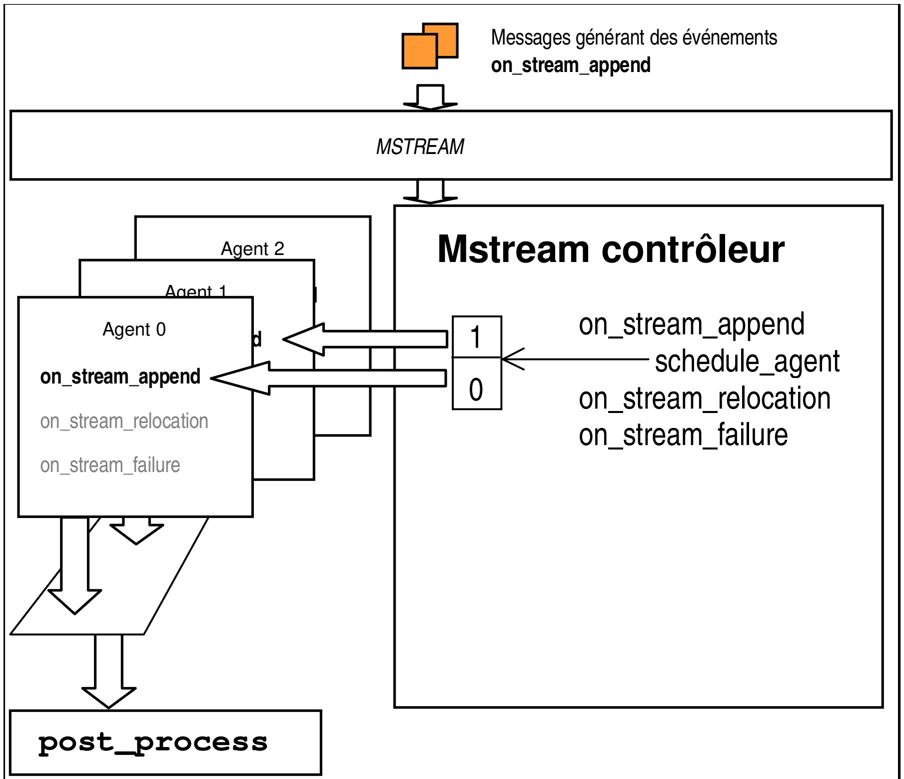

Figure 4. *Mstream contrôleur* et traitement d'un message

3.2.2. Contrôle des locations

Il est possible de contrôler le comportement d'une *location* ou en d'autres termes de paramétrer les démons d'agents en spécifiant des scripts qui peuvent par exemple : implanter des politiques d'acceptation des agents (refus d'initialiser un nouvel interprète, ou désactivation de ses commandes « dangereuses »…), contrôler l'arrivée de *Mstreams* sur la *location*… Ces scripts sont exécutés par un thread/interprète dedié de chaque démon d'agents. En pratique ces définitions sont mises dans un fichier dont le nom est passé en paramètre lors du démarrage d'un démon d'agents.

3.2.3. Contrôle de la plate-forme

Suivant la même idée que précédemment il y a moyen de contrôler le fonctionnement général de la plate-forme. Pour être précis 9 événements sont pris en compte : la création ou destruction d'une *location*, d'un *Mstream* ou d'un agent, déplacement, ouverture et fermeture d'un *Mstream*. Dans tous ces cas le programmeur peut associer un fragment TCL qui autorise ou rejette ces événements. Dans l'implantation actuelle ces fragments de code sont placés dans un fichier dont le nom est passé en paramètre lors du démarrage du contrôleur du système. C'est en effet un thread/interprète dédié de ce démon qui a la charge de vérifier des événements "globaux" de la plate-forme.

3.3. Quelques détails et discussions sur l'implantation

3.3.1. Modèle et fonctionnalités

Globalement on constate que le programmeur est face à une programmation par événements. Il doit implanter un certain nombre de réponses à des événements fournis par différentes boucles (*main event loop*) : une par *Mstream*, une par *location* et une par plate-forme. Ceci n'introduit donc aucun nouveau concept sophistiqué, un programmeur codant des interfaces graphiques ne devrait par être spécialement dépaysé. De plus l'ensemble des *locations*, puis des *Mstream*s d'une application peuvent s'instancier[3] et se contrôler d'un point unique, éventuellement intéractif (shell TCL). Cette facilité masque les problèmes de distribution du code à l'utilisateur.

3.3.2. Implémentation

L'implémentation actuelle est plutôt naïve. Elle a deux buts principaux. Le premier est d'offrir les services escomptés au programmeur afin de valider, à l'usage, nos idées. L'ensemble des fonctionnalités présentées à la section 3.2 n'a pas été imaginé en une seule fois. Il est le fruit d'une suite de développements au-dessus du système suivis d'améliorations. Le deuxième est la mise au point du contrôleur des *locations* et du contrôleur du système le problème étant de faire cohabiter plusieurs interprètes TCL muni de leur thread, et les différents caches nécessaires (code, position d'un nom).

Même si la plupart des choix d'implantation sont simples nous avons mis l'accent sur un point important : les communications entre deux *Mstream*s "suffisamment" fixes doivent être des communications **point à point effectives**. Nous avons volontairement exclu une solution basée sur un serveur/routeur central. Ce choix est dû aux applications tests visées manipulant plutôt des flux d'événements que des services de livraison de messages.

Dans une première version toutes les communications étaient implantées via TCP, ce qui nécessite la fermeture concertée et la réouverture d'une ou plusieurs connexions TCP lors du déplacement d'un *Mstream*. En effet ce *Mstream* doit rétablir les connexions correspondant aux `stream_open` que ses agents ont effectués. Les *Mstream*s ayant fait des `stream_open` sur celui qui vient de bouger

[3] Seuls les *Mstreams* peuvent être instanciés à partir d'un point unique.

doivent aussi rétablir une connexion. Ceci a évidemment un impact sur les performances du déplacement d'un *Mstream* : il faut stopper temporairement les émetteurs visant le *Mstream* désirant bouger, fermer les connexions, les rouvrir (8 paquets ip), débloquer les émetteurs, et subir le démarrage "lent " de la fenêtre TCP (*TCP slow start* [STE 97]) sur chacune des nouvelles connexions. D'après les quelques mesures effectuées le coût de déplacement d'un *Mstream* est du même ordre que le déplacement d'un agent TCL dans le système *TACOMA* [JVS 95][4], mais reste plus de 50% plus lent qu'une implantation "à la main " du passage (via l'interface TCL des sockets) et de l'exécution d'un script entre des interprètes TCL standards distribués sur plusieurs hôtes.

Nous avons choisi d'implanter une première version d'un protocole adhoc au-dessus d'udp pour les communications entre deux *Mstream*s. Vues les contraintes d'ordre désirées la base du protocole est un protocole à fenêtre à la TCP. L'acquittement correspond à la fin de l'exécution des scripts associés à `on_stream_append` pour obtenir un contrôle de flux au niveau application. Il y a deux problèmes à traiter : - implanter la fonction permettant de connaître la *location* courante d'un *Mstream*, - assurer la continuité des messages et numéros de séquence lors du déplacement d'un *Mstream*.

Le premier problème est actuellement résolu par la gestion sur chaque *location* d'un cache des positions des agents déjà contactés. La mise à jour du cache se fait en deux temps : réception d'un acquittement négatif dédié "agent inconnu sur cette *location*", puis consultation du contrôleur du système. Ce mécanisme simple nous permet d'avoir une bonne performance quand les *Mstream*s bougent peu, et de limiter en général la recherche de la position d'un agent à l'échange de 4 messages.

Le problème de la continuité des messages présente deux aspects : celui des messages **destinés** au *Mstream* à bouger et celui des messages restant dans les fenêtres d'émission de ce même *Mstream*[5]. Nous avons décidé de ne pas utiliser de méthodes à base de proxy, elles nous semblaient être un obstacle au support de la défaillance des *locations*.

Quand un *Mstream* bouge il abandonne tous les messages reçus mais non acquittés. Le premier aspect sera donc assuré par des réémissions, sur *timeout* ou acquittement négatif dédié par les *Mstream* **émetteurs**. On voit donc qu'un *Mstream* doit conserver dans son état mobile **le numéro de séquence du dernier message traité** pour chaque *Mstream* lui ayant envoyé des messages. Nous avons réglé le cas des fenêtres d'émission de manière drastique : avant de bouger, un *Mstream* attend le passage à zéro de ses fenêtres d'émission. Il faut noter que lors de la phase de déplacement d'un *Mstream* aucun message entrant n'est traité, donc aucun script `on_stream_append` n'est exécuté, aucune commande d'émission

---

[4] AGNI est en fait un peu meilleur grâce à un mécanisme de cache du code des agents.

[5] Contrairement à la mobilité envisagée dans Ipv6 [CIZ 98] la pile de communication ne bouge pas : nous traitons de la mobilité d'objets logiciels et non de celle d'équipements physiques.

`stream_append` en particulier. L'attente de ces passages ne dépend donc que des récepteurs. L'état mobile d'un *Mstream* doit donc inclure le prochain **numéro de séquence en émission** pour chaque *Mstream* avec qui il communique, en d'autres termes pour chaque *Mstream* sur lequel un de ses scripts a exécuté `stream_open`.

L'implantation actuelle nous permet de supporter les applications décrites dans la section suivante de manière acceptable. Des défauts demeurent : il y a par exemple des problèmes de *deadlock* évidents (cas du déplacement d'un *Mstream* envoyant des messages à lui-même) actuellement résolus de manière empirique. Il nous faut aussi faire des mesures de performances pour remettre en cause les choix faits pour l'implantation du protocole inter *Mstream*.

4. Applications

4.1. Partage d'applications Tcl/Tk

Notre première application pour tester AGNI est une boîte à outils pour partager des applications graphiques non modifiées écrites en TCL/Tk. Le but de cette boîte à outils est de permettre à plusieurs utilisateurs d'exécuter localement un même programme utilisant une interface graphique tout en maintenant le paradigme "**ce que je vois est ce que les autres voient**" (verront ou ont vu, en fait) pour tous. L'application partagée peut être par exemple *Tk Draw* [T 98] afin de réaliser un "tableau blanc". Les messages venant d'un utilisateur sont ici les événements graphiques "volés" au programme par du code adapté de *Tk Replay* [CRO 95].

4.2. Cartographie d'un réseau IP

La gestion des réseaux et services est un domaine où les agents mobiles semblent avoir un avenir possible [LKP 98, BGP 98, SM 98]. Les agents mobiles sont une extension naturelle de la gestion par délégation et le but principal est d'alléger la charge de la station de supervision aussi bien en ressource de calcul qu'en ressource réseau. Nous avons fait un premier test sur un utilitaire de cartographie d'un réseau Ipv4 du même type que [FPV 98,MKM 98]. Il prend en entrée une liste de réseaux de classe C à cartographier et rend la liste des nœuds qui répondent sur ces réseaux. Le résultat est délivré à la location où ont été spécifiés les réseaux cibles.

Nous avons rendu disponible l'extension dynamiquement chargeable *TMN* de *Scotty* [SCH] dans les démons d'agents. Ceci permet à un agent AGNI d'effectuer des requêtes ICMP echo [BRA 89, page 42] et donc de tester l'activité d'une adresse IP donnée.

Trois méthodes ont été testées pour la cartographie de deux réseaux IP reliés par deux autres réseaux dont le contenu ne nous concerne pas. Cette topologie très simple peut correspondre à l'interconnexion de deux réseaux locaux via un transporteur payé. La première solution, “de référence” est purement centralisée : toutes les adresses possibles sont déterminées puis probées via ICMP (Scotty fournit un mécanisme de fenêtre d’émission/acquittement/*timeout* des requêtes ICMP). C’est le schéma habituellement suivi pour les moniteurs IP. La deuxième met en cause un *Mstream*+agent mobile unique essayant de trouver une *location* AGNI sur un réseau cible, puis s’y déplaçant. L’idée est bien entendu d’effectuer les requêtes ICMP écho à partir de et en restant dans un réseau cible grâce à l’agent AGNI. Une fois un réseau cible testé, les adresses qui répondent sont placées dans la *briefcase* et l’agent passe à un autre réseau cible. Après avoir visité chaque réseau l’agent revient à la *location* de départ et affiche la liste des adresses trouvées. La *location* de départ, la liste des réseaux à explorer, et l’adresse du prochain réseau à traiter sont des informations elles aussi placées dans la *briefcase*. Les réseaux ne pouvant accepter de *Mstream* sont testés à distance comme dans la version centralisée. On peut noter que cette solution n’a pas besoin des possibilités de communications inter-agents d’AGNI. La troisième solution utilise deux types d’agent : un agent “superviseur ” attaché à un *Mstream* de même nom et des agents cartographes. Le superviseur est créé avec la liste des réseaux cibles en paramètres et affiche les résultats. Il essaye d’envoyer un agent cartographe par réseau cible. Un agent cartographe ressemble à l’agent utilisé dans la deuxième solution sauf que ses résultats locaux sont envoyés en une fois par un `stream_append` sur le superviseur. Le superviseur teste de manière centralisée les réseaux n’offrant pas de location AGNI. Cette fois des communications inter agents sont utilisées, mais les agents cartographes ne sont mobiles que pour être déployés. Par contre ils pourraient aisément communiquer ultérieurement des modifications locales au superviseur qui lui peut bouger : on peut aisément séparer la station d’affichage de l’application elle-même.

Les résultats obtenus sur des exemples limités sont encourageants : le trafic global (somme de toutes les occurrences de paquets IP) baisse d'environ 30% entre la version centralisée et les deux autres. La diminution du trafic sur les réseaux non cibles (interconnexion) est encore plus importante (50%). Par contre, la solution centralisée reste plus de 3 fois plus rapide que les deux autres.

5. Travaux relatifs

Nous n’avons pas l’ambition de faire une étude complète des systèmes ou applications à agents mobiles existants, le lecteur peut se reporter à l’édition spéciale de IEEE transaction on software engineering de mai 1998 [TSE 98] ou à celle de Communications Magazine de juillet 1998 [CMA 98]. Nous allons simplement citer les travaux qui ont directement influencé notre travail. Notre système a été conçu pour incorporer les besoins spécifiques d’applications de contrôle, collaboration ou test distribuées qui s’adaptent bien à un modèle par événements. Le Meta toolkit [WM 91] qui adopte ce modèle a influencé ce travail. Meta n’offre pas de possibilité de mobilité ou de reconfiguration souple comme dans AGNI.

Une réalisation précédente de mobilité dans Java : le système SUMATRA [RASS 97] a eu une influence certaine dans certains choix simplificateurs faits dans AGNI. D'autres travaux [RAS 98] nous ont encouragés à utiliser un nommage global non lié à la localisation. Bien entendu nous avons étudié des systèmes offrant une certaine mobilité comme *TACOMA* [JVS 95], *Voyager* [OSVoy] , *Aglets* [VEN 97] , *Odyssey* [GMOd], *ARA* [PEI 98] et MOLE [SBH 96] ainsi que des travaux plus anciens comme *Emerald* [JLHB 88] et *Obliq* [CAR 95]. Nous pensons avoir innové par les différentes possibilités de contrôler une partie du fonctionnement de la plate-forme (niveau système, *location* et *Mstream*/agent). Une autre caractéristique est la séparation nette du nommage (*Mstream*, *location*) des aspects opérationnels (agents). Un autre point important est l'utilisation aisée des communications inter-agents.

AGNI a définitivement opté pour une mobilité de serveur comme *Aglets, Voyager, TACOMA* contrairement à des systèmes à mobilité de *thread* comme [RASS 97, GRA 96, PEI 98].

6. Conclusions

Nous avons présenté un système à agents mobiles à mobilité restreinte, introduisant deux séparations nettes : l'une entre points de communication et *locations*, l'autre entre points de communication et traitements (agents). La flexibilité obtenue est agréable à l'usage. Cet usage est d'ailleurs aisé grâce à la possibilité de contrôler et construire le système d'un point central, à l'aide de commandes en ligne ou d'un outil graphique. AGNI offre donc le même aspect qu'un *shell* d'un système d'exploitation centralisé. Pour le programmeur, la prise en main est facilitée par la simplicité des scripts à écrire et la structure des messages : de simples chaînes de caractères. L'utilisation de serveurs *multithreads*, la légèreté des interprètes TCL, et le développement de protocoles adhoc nous permettent de nous assurer que le prototype développé supporte les applications visées. La pleine maîtrise du prototype nous permet d'envisager des développements et des tests de performances précis. La mobilité restreinte retenue ne gène en rien l'écriture des applications et allège considérablement l'implantation de la plate-forme et l'exécution du déplacement d'un *Mstream*. Par contre les propriétés d'ordre des communications sur un *Mstream* effectivement utilisées dans nos applications compliquent l'implantation de ces déplacements.

Bien entendu les arguments positifs précédents peuvent être repris de manière négative. L'utilisation d'un langage de script et de communications non structurées rendent difficiles la réutilisation du code (même en utilisant des artifices comme le macroprocesseur m4). Actuellement AGNI n'est pas adapté au développement de grosses applications. Il est clair que des systèmes comme *Aglet*, *Odyssey*, ou *Voyager*, où un agent est un objet d'un langage de programmation offrent plus d'attrait au programmeur expérimenté. Le développement de protocoles adhoc dans une plate-forme complètement autonome est certainement une solution limitant l'intégration d'éléments existants dans une application utilisant des agents mobiles.

Les travaux actuels de *l'Object Management Group* [OMG 97] tendent à proposer un service de mobilité au-dessus de CORBA où différents constructeurs pourront intégrer leur plate-forme à agents mobiles. Il est aussi clair que TCL/C n'est peut-être pas la meilleure des solutions pour faciliter l'implantation d'AGNI sur des architectures variées.

Une dernière réflexion subjective doit aussi être faite[6] : les applications basées sur des agents mobiles sont des applications distribuées et bien sûr elles pourraient être implantées sans agents mobiles à l'aide de protocoles applicatifs, d'appels de procédures à distance ou tous autres mécanismes classiques de ce domaine. Nous pensons en tout état de cause que le concept d'agents mobiles, de *Mstream* dans notre cas est un moyen agréable pour concevoir des applications.

7. Perspectives

L'évolution d'AGNI devra être guidée par trois lignes principales :

1. L'ajout de fonctions supplémentaires pour supporter les applications visées comme un service de distribution de données "en vrac" (*bulk data*) ou un contrôle d'accès sur les noms. Il est clair qu'il nous faut envisager de fournir une notion de groupe nommé de *Mstreams* afin d'évoluer de communications 1-1 vers des communications 1-n (*multicast*) entre agents. Aucun travail sur la sécurité des accès n'a actuellement été entrepris.
2. L'amélioration des principes de base. Il faut améliorer la résistance de la plate-forme face à une partition du réseau. Le contrôleur du système doit donc être distribué ou du moins répliqué, avec les problèmes de synchronisation et cohérence qui en découlent. Notre modèle supporterait très mal la duplication d'un nom.
3. Test et amélioration de l'implantation de l'existant. Le problème du passage à l'échelle se pose. Nous allons porter AGNI au-dessus d'un simulateur afin de pouvoir faire des tests sur un grand nombre de locations. La fonction de location d'un *Mstream* pourra utiliser les résultats de [SHHT 98].

8. Bibliographie

[BGP 98] BALDI M., GAI S., et PICCO G.P., " Exploiting code mobility in decentralized and flexible network management ", *Proceeding of the 1st Mobile Agent Workshop*, 1997.

[BRA 89] BRADEN R.T., " Requirements for Internet hosts - communication layers..", RFC 1122, 1989 http://www.pasteur.fr/other/computer/RFC/11xx/1122

[CAR 95] CARDELLI L., " A Language with Distributed Scope ", *proc 22nd SIGPLAN-SIGACT Symposium on the principles of Programming Languages,* janvier 1995.

[CMA 98] *IEEE Communications Magazine*, vol 36, n° 7, juillet 1998.

[6] Les auteurs se l'ont souvent faite durant ce projet.

[CIZ 98] CIZAULT G ., *Ipv6, théorie et pratique*, O'Reilly, 1998. ISBN 2-84177-052-4

[CRO 95] CROWLEY C., " Tk-Replay: Record and Replay in Tk ", *USENIX Third Annual Tcl/Tk Workshop*, 1995.

[FPV 98] FUGGETTA A., PICCO G.P., et VIGNA G., " Understanding code mobility ", *IEEE Transaction on software engineering,* vol 24, n°5, mai 1998

[GRA 96] GRAY R.S., " Agent Tcl: A Flexible and Secure Mobile-Agent System ", *Proceedings of the Fourth Annual Tcl/Tk Workshop,* 1996.

[GMOd] GENERAL MAGIC CORP., " Odyssey web-page ", www.genmagic.com/agents.

[JVS 95] JOHANSEN D., VAN-RENESSE R., et SCHNIEDER F.B., " An introduction to the TACOMA Distributed System ", Technical Report 95-23, Dept of Computer Science, University of Tromso, Norway, 1995.

[JLHB 88] JUL E., LEVY H., HUTCHINSON N., et BLACK A., " Fine-grained mobility in the Emerald System ", *ACM Transactions on Computer Systems,* vol 6, n°2, Février, 1988.

[LKP 98] LIOTTA A., KNIGHT G., et PAVLOU G., " Modelling network and system monitoring over the internet with mobile agents ", *NOMS'98*, 1998, vol 2, page 303-312.

[MKM 98] MINAR N., HULTMAN KRAMER K., et MAES P., " Cooperating mobile agents for mapping networks ", *Proceedings of the first Hungarian National Conference on Agent Based Computing*, mai 1998. Voir aussi www.media.mit.edu/~nelson/research/routes-coopagents.

[OMG 97] OMG (IBM, GMD Fokus, et General Magic), " CORBA facilities : Mobile Agent Facility Submission ". www.omg.org.

[PEI 98] PEINE H., " Agents for Remote Access ", *Mobile Agents*, Editeur : Cockayne W.R. et Zyada M., Manning Publishers, ISBN 1-884777-36-8, 1998.

[RAS 98] RANGANATHAN M., ACHARYA A., et SALTZ J., " Adaptive Data Combination over Wide-area networks ", *International Conference On Distributed Computing Systems (ICDCS*), 1998, p. 498-506.

[RASS 97] RANGANATHAN M., ACHARYA A., SHARMA S.D., et SALTZ J., " Network-aware Mobile Programs ", *USENIX*, 1997.

[OSVoy] Voyager White Paper, www.objectspace.com/voyager .

[S] " Tcl home page ", www.scriptics.com.

[STE 97] STEVENS W. , " TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms ", RFC 2001., www.pasteur.fr/other/computer/RFC/20xx/2001.

[SBH 96] SRAβER M., BAUMANN J., et HOHL F., " Mole – a Java-based mobile agent system ", *2ème ECOOP Workshop on Mobile Object Systems*, Linz, Austria, 1996.

[SCH] SCHÖNÄLDER J., " The SCOTTY Home Page ", wwwhome.cs.utwente.nl/~schoenw/scotty.

[SHHT 98] VAN STEEN M., HAUCK F.J., HOMBURG P., et TANENBAUM A.S., “ Locating Objects in Wide-Area Systems ”, *IEEE Communications Magazine*, January 1998, p. 104-109, ftp.cs.vu.nl:/pub/papers/globe/commag.98.pdf.

[SM 98] SAHAI A. et MORIN C., “ Towards distributed and dynamic network management ”, *NOMS'98*, 1998, vol. 2, p. 455-464.

[T 98] “ Tk-Draw web page ”, http://www.inf-technik.tu-ilmenau.de/~silvio/research/soft.htm.

[TSE 98] *IEEE Transaction on software engineering, Special issue on : Mobility and network aware computing*. vol 24, n°5, mai 1998.

[VEN 97] VENNERS B., “ Under the Hood: The architecture of Aglets ”, *Java World,* avril 1997.

[WM 91] Wood M., et Marzullo K., “ The Design and Implementation of Meta ”, *dans Reliable distributed computing with the ISIS toolkit*, Editeur : Berman K. et van Renesse R., IEEE Computer Society Press, ISBN 0818653425, 1994.