Autorégulation et adaptation du traitement de flux de données par une architecture multi-agent

Laurent Poligny, Guillaume Hutzler

Autorégulation et adaptation du traitement de flux de données par une architecture multi-agent

Laurent Poligny, Guillaume Hutzler

IBISC - FRE 3190, Université d'Évry Val d'Essonne
523 Place des Terrasses de l'Agora, 91000 Évry – FRANCE

Résumé : La construction de systèmes d'interaction homme-machine repose sur la notion sous-jacente d'architecture de traitement de données. Dans le contexte dynamique de l'informatique ambiante, ces architectures ne peuvent plus être conçues de manière statique mais doivent au contraire évoluer dynamiquement, dans la recherche d'un compromis entre temps de réponse et pertinence du traitement effectué par le système. Nous proposons dans ce cadre une architecture fondée sur l'encapsulation des fonctions de traitement dans des agents génériques dotés d'un modèle comportemental simple, et ajustant les interactions entre agents en fonction de leur satisfaction ou au contraire de leur frustration. Nous montrons qu'une organisation dynamique peut alors s'établir, permettant une autorégulation dynamique et adaptée au contexte de fonctionnement du système.

Mots-clés : auto-régulation, adaptation, programmation flux de données, système holonique

1 Introduction

Dans le contexte de l'informatique ambiante, décrit par Harper *et al.* (2008), la notion même d'ordinateur est profondément modifiée : elle inclut désormais tous types de dispositifs de captation, de traitement et de rendu disséminés dans l'environnement de l'utilisateur. Ces dispositifs incluent aussi ceux que l'utilisateur porte sur lui comme un téléphone portable ou un assistant personnel. Un ordinateur ne peut plus être considéré comme un dispositif unique mais comme un ensemble de dispositifs en interaction constante. Ceci change la notion d'interaction homme-machine, tant du point de vue de l'ergonomie de l'interface que du point de vue des architectures de traitement de données sous-jacentes.

Dans ce contexte, nous ne nous intéresserons qu'aux problématiques liées aux architectures de traitement. Elles ne peuvent plus être conçues de manière statique mais doivent au contraire évoluer dynamiquement. D'une part pour prendre en compte l'ajout ou la suppression de ressources (pour la captation, le traitement ou le rendu), d'autre

part pour rechercher un compromis entre temps de réponse et pertinence du traitement effectué par le système.

Pour illustrer la pertinence d'un traitement et du temps que le système utilise pour s'ajuster au contexte, considérons un scénario de suivi en vidéo conférence mobile et examinons les attentes des utilisateurs de ce service dans le contexte de l'informatique ambiante.

Alice rentre chez elle en discutant avec Bob sur son téléphone 3G. En passant dans le salon le visage de Bob apparaît sur l'écran de la télévision, pour une fois disponible puisque ses enfants ne sont pas là. Mais Alice n'y prête pas attention, ce qui pourrait indiquer au système que dans ce contexte précis l'utilisation de cet écran n'est pas pertinent. Elle se dirige dans la cuisine. Sans lâcher son téléphone, elle se prépare un thé et s'installe à la table. L'écran de contrôle du réfrigérateur (qui affiche normalement le menu pour le soir et les articles manquants) affiche l'image de Bob. Elle lève la tête et pose son téléphone. Les caméras disséminées dans la cuisine prennent le relai sur celle incluse dans le téléphone portable. Bob préférant les plans rapprochés demande au système d'Alice de satisfaire sa préférence en maintenant le zoom au maximum sur le visage d'Alice. Mais lorsque cette dernière commence à préparer à manger, Bob demande au système de la maison de la suivre en plan large. Le système va essayer d'envoyer à Bob les images d'Alice se déplaçant dans sa cuisine pour préparer un pain aux épices. Pour suivre Alice, le système calcule sa position et anticipe (si possible) ses déplacements (réfrigérateur, plan de travail, four...). Mais Alice ne facilite pas la détection de ses déplacements en plongeant sa cuisine dans la pénombre que procure les petites lumières tamisées au dessus du plan de travail. L'anticipation de la position d'Alice est difficile car les images des caméras sont trop sombres et la détection ne peut se faire que très près des caméras. Cela entraînera des latences entre l'apparition d'Alice devant une caméra et son apparition sur l'écran de Bob.

Ce scénario, qui reste relativement basique, illustre cependant un certain nombre de contraintes que nous nous efforçons d'aborder dans la conception d'architecture de traitement de données *adaptatives* et *auto-régulées* [Vallée *et al.* (2006)]. L'*adaptation* à laquelle nous faisons référence consiste, pour le système, à permettre l'ajout dynamique de nouvelles ressources, que ce soit pour la captation, le traitement ou encore le rendu. Cela consiste également à rendre le système robuste au retrait ou à la panne de certains des éléments le composant. L'*auto-régulation* correspond à la capacité, pour le système, à constamment adapter son fonctionnement en rapport avec les ressources disponibles et le contexte d'utilisation (conditions environnementales de lumière ou de bruit, personnes en interaction, etc.).

Plus précisément, les propriétés attendues du système sont les suivantes :

- le système doit supporter l'*ajout dynamique de ressources*, logicielles ou matérielles ;
- le système doit être *tolérant aux pannes* ;
- le système doit être capable de *coupler les données* lorsque celles-ci proviennent de systèmes de captation complémentaires ;
- le système doit s'*adapter au contexte* ;
- le système doit fonctionner en *temps-réel*.

Nous allons, dans la suite de ce papier, présenter un modèle d'architecture de traitement s'auto-régulant en fonction de la pertinence des informations et du contexte ainsi

que du temps de traitement. Ce modèle s'adapte potentiellement à l'ajout et à la suppression dynamique de ressources et de fonctions de traitement.

Nous présenterons dans la première partie deux grandes approches du traitement de flux, la programmation par flux de données et les systèmes holoniques, puis nous exposerons notre vision les conciliant. Nous présenterons ensuite un modèle d'agents génériques permettant une conception rapide et supportant l'autorégulation distribuée. Nous détaillerons finalement une approche d'autorégulation permettant d'obtenir un compromis dynamique entre le temps de traitement et la pertinence du rendu.

2 Entre la programmation par flux de données et les systèmes holoniques

Dans la suite, nous allons considérer plusieurs caractéristiques :

- pour un même résultat, *plusieurs traitements sont possibles* : soit parce qu'il existe pour une même information plusieurs fonctions de même nature (par exemple, en utilisant une caméra, on peut extraire la trajectoire d'un objet dans une pièce en calculant, image par image, l'ensemble des positions successives de l'objet, mais on peut aussi combiner la position courante et le vecteur accélération pour anticiper la position suivante) ; soit parce qu'il existe des capteurs différents permettant à des traitements de conduire à un même résultat (on peut détecter la présence d'une personne en l'observant avec une caméra, ou simplement en utilisant un détecteur de présence).
- on veut pouvoir *contrôler la pertinence* des différents chemins de traitement, en jugeant de l'intérêt d'une information. L'utilisateur, faisant partie du système, doit pouvoir interagir en attribuant de l'intérêt à une information que le système produit ou à la façon dont le système la produit. Par exemple, l'utilisateur peut préférer utiliser la caméra qui donne un plan large, à une caméra qui donne un plan de profil, ou bien l'utilisateur peut préférer un système proposant des changements rapides de plan, quitte à avoir des changements inutiles, à un système trop stable qui donnerait toujours la même caméra même si son interlocuteur est hors champ.
- le système doit chercher à satisfaire des *contraintes* ou *objectifs de temps* sans pour autant *empêcher les dépassements*. C'est à dire que si l'utilisateur demande au système de changer de caméra quand son interlocuteur n'est plus visible pendant 2 secondes la validité du système n'est pas remise en cause si le changement en prend 5. C'est en moyenne que le système doit satisfaire les contraintes de temps. Dans ce cas on parle généralement de temps-réel mou.

Pour satisfaire les exigences de traitement de notre scénario, le système doit ainsi être capable de traiter les flux de données (multimédia) et son architecture doit permettre de gérer l'adaptation aux changements de l'environnement, aux pannes, aux ressources insuffisantes et même à la non-satisfaction de contraintes.

2.1 Traitement d'une donnée et traitement de flux de données

Pour traiter une donnée par un système informatique, on réalise un programme. On le découpe en sous-programmes (ou fonctions de traitement) plus petits prenant en entrée un ensemble d'informations à traiter et retournant en sortie une nouvelle information, qui peut être stockée dans une variable pour une utilisation ultérieure. On parcourt le programme, contenu dans la mémoire centrale de l'ordinateur, instruction par instruction (la ligne en cours d'exécution est donnée par le compteur ordinal, caractéristique physique des machines de Von Neumann).

Lorsque le système ne traite plus les informations de manière ponctuelle dans le temps mais comme une séquence continue de données, l'approche change.

Le programme s'exécute dès que les données sont disponibles en entrée. Un programme peut alors être vu comme un graphe dirigé de fonctions de traitement dans lesquelles les données circulent. Sans nouvelle donnée le programme s'interrompt et attend. Si au contraire, plusieurs données arrivent, elles sont traitées dans l'ordre d'arrivée (algorithme de type FIFO).

Pour le traitement de flux de données, il existe deux grands paradigmes. Nous allons maintenant les détailler et les confronter à notre exemple.

2.2 Introduction à la programmation par flux de données

La programmation par flux de données [Uustalu & Vene (2005)] est née dans les années 70-80 de la nécessité de paralléliser le traitement d'informations indépendantes, de façon similaire aux circuits électroniques. L'objectif fixé à la programmation par flux de données était de supprimer l'exécution séquentielle (dirigée par un seul compteur ordinal) et l'utilisation centralisée de la mémoire [Silc *et al.* (1997)]. Le programme ne doit plus être centré sur les instructions mais sur les données. On peut généraliser les notions introduites par cette approche comme la volonté de distribuer le traitement des données et d'augmenter l'autonomie des entités qui effectuent ces traitements. On retrouve cette même notion d'autonomie de traitement dans les systèmes multi-agents [Ferber (1995)].

Du fait de l'approche compositionnelle des langages de programmation par flux de données, ces derniers présentent deux caractéristiques importantes, particulièrement intéressantes dans notre contexte : la première est la possibilité d'exploiter des librairies de fonctions de traitement, ce qui permet de tirer parti de la puissance de librairies spécialisées efficaces [Puckette (1996)] ; la seconde est la possibilité de définir une chaîne de traitements en utilisant les concepts de la programmation graphique introduits dans les années 90 [Johnston *et al.* (2004)], ce qui permet de structurer graphiquement un programme en visualisant directement les fonctions de traitement et leurs connexions.

2.3 Holons et système de production holonique

La notion de holon a été proposée par Koestler [Koestler (1967)]. Il y décrit des entités autonomes et modulaires qui ont pour objectif la manipulation d'informations et

d'entités physiques. Les systèmes de production holoniques sont définis par le modèle HMS[HMS (1994)] comme suit :

- Holon : une entité de base autonome et coopérative pour la transformation, le transport, le stockage et / ou la validation, aussi bien de l'information que des objets physiques. Le holon est généralement constitué d'une partie de traitement de l'information et d'une partie physique. Un holon peut faire partie d'un autre holon.
- Autonomie : la capacité d'une entité à créer et contrôler l'exécution de ses propres plans et/ou stratégies.
- Coopération : un processus par lequel un ensemble d'entités élabore des plans mutuellement acceptables puis les exécute.
- Holarchie : un système de holons qui peuvent coopérer pour atteindre un but ou un objectif. L'holarchie définit les règles de base pour la coopération des holons et donc les limites de leur autonomie.
- Holonic Manufacturing System (HMS) : une holarchie qui intègre l'ensemble des activités manufacturières, depuis la commande jusqu'à la commercialisation en passant par la conception et la production, de manière à rendre l'entreprise réactive.

Les holons peuvent communiquer et modifier leur organisation pour effectuer une tâche qu'ils ne pourraient effectuer individuellement. De cette interaction, on attend l'émergence d'une structure permettant la résolution d'un problème donné.

Des connexions vont se créer entre les holons, dans lesquelles les données transiteront, dirigées vers les sorties. Dans ces connexions circuleront, en sens inverse, les informations liées au contrôle du traitement (le feedback). Ce retour d'information donne aux holons une vision locale de leur participation aux traitements.

2.4 Comparaison des deux approches

2.4.1 Les points communs

Ces deux approches peuvent servir le même objectif : traiter des flux d'informations.

Pour ce faire, ces approches adoptent une démarche similaire. Les informations circulent, au travers de chemins de traitement (composés d'éléments), dirigées de la production d'informations vers la sortie du système.

Elles se basent également sur une construction récursive des éléments qui les composent, permettant ainsi l'utilisation de fonctions de traitement à différents niveaux d'abstraction.

2.4.2 Les différences

La conception d'un système permettant la production de données est très différente. Dans le premier cas, on utilise la programmation par description des entrées/sorties des fonctions de traitement et la connexion des différentes fonctions entre elles. Dans le deuxième cas, le concepteur doit créer un ensemble d'agents autonomes capables de résoudre localement différents cas de figure en négociant avec les autres agents.

Il n'y a pas de retour d'informations (feedback) dans la programmation par flux de données pour qualifier la production. Dans le cas des HMS, le retour d'informations

permet de gérer la production et l'organisation entre les différents holons. Ces informations sont cruciales pour gérer les problèmes de production.

2.4.3 Un juste milieu

En utilisant la programmation par flux de données, le concepteur aurait à anticiper tous les cas possibles du contexte (la personne hors champ, la luminosité) ce qui reste encore envisageable pour cet exemple simple, mais peut se révéler irréalisable dans le cas d'applications plus complexes. Quant à l'approche par système holonique, le concepteur n'aurait plus à prévoir tous les contextes possibles, mais devrait faire face aux performances pénalisées par des interactions et des négociations complexes entre les holons.

Nous avons brièvement montré que ces deux approches peuvent partager le même objectif mais mettent en oeuvre des stratégies différentes pour l'atteindre. Nous avons donc exploré une voie médiane pour combiner d'une part la rapidité de programmation et d'exécution propre à la programmation par flux de données, et d'autre part la capacité d'adaptation des systèmes holoniques.

2.4.4 Illustration du fonctionnement souhaité

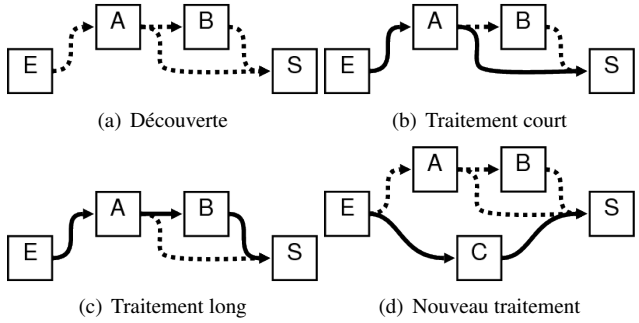
Toujours en utilisant notre scénario, nous présentons un exemple jouet extrêmement simplifié du suivi et d'identification d'un individu. Le module E représente une caméra, A un traitement d'extraction de position, B un module d'anticipation de position d'individu, C le traçage en utilisant une puce RFID et S le module choisissant la caméra à utiliser.

La figure 1(a) montre les différents traitements que peuvent subir les données sortant du module E. Ce dernier fournit des données que A peut traiter. Le module de sortie S peut traiter indifféremment les données fournies par A ou B.

On considère que les données traitées par la chaîne $EABS$ seront plus précises que les données traitées par la chaîne EAS. Par contre, le temps de traitement par $EABS$ ne satisfait pas les contraintes de temps énoncées pour le système.

Le comportement souhaité du système est alors d'alterner de longues périodes de traitement par EAS (pendant lesquelles il respecte les contraintes de temps), avec de brèves périodes de traitement avec $EABS$ (pendant lesquelles il améliore la précision de traitement), comme l'illustrent les figures 1(b) et 1(c).

Une connexion entre A et S permet d'isoler le traitement de B et de stabiliser le système sur une configuration. La connexion entre B et S permet à S de ne pas considérer les informations produites par A (si il n'existait pas de lien entre A et B). La période d'alternance des traitements doit pouvoir être ajustée par l'utilisateur. De plus, si un nouveau traitement apparaît (figure 1(d)), qu'il satisfait la contrainte de temps et qu'il est plus pertinent pour l'utilisateur, alors il doit avoir la priorité sur les autres traitements. Cette priorité n'est pas définitive puisque le contexte peut à nouveau changer, imposant au système d'ajuster son traitement.



(a) Découverte (b) Traitement court

(c) Traitement long (d) Nouveau traitement

FIG. 1 – Différents chemins de traitement

3 Description des éléments de l'architecture

Le principe général de notre approche repose sur deux idées : un développement orienté entité ainsi qu'une alternance et un ajustement dynamique des périodes de traitement par des chemins similaires.

Dans la suite, la notion de *donnée* fera référence à un couple description et valeur, et *information* à un ensemble de données.

La première caractéristique de notre système consiste à considérer que les fonctions de traitement (au sens de la programmation par flux de données) sont encapsulées dans des agents de traitement génériques. La conception de ces agents de traitement (que l'on distingue de la conception du système) consiste simplement en la description des données (incluses dans les informations) nécessaires à l'utilisation de la fonction de traitement et des informations retournées par l'agent ainsi que l'intégration de la fonction de traitement. Les agents doivent pouvoir être basés sur des bibliothèques de fonctions aussi facilement que pour la programmation par flux de données.

La seconde caractéristique, liée au traitement et permettant la régulation des agents, porte sur l'indépendance des traitements entre les agents et le fait que les agents accèdent à n'importe quelles informations rendues par les autres agents y compris eux-mêmes (via un tableau noir). Ceci permet de concentrer notre approche sur la régulation entre les agents.

L'objectif est d'abstraire les problématiques suivantes :

– Il est difficile de prédire la nécessité d'une information pour un futur *traitement ponctuel*. Dans notre approche, les agents vont systématiquement utiliser les informations qu'ils peuvent traiter. Par exemple, si le système peut identifier une personne dans la pièce sans que cela trouble les contraintes de temps du système alors il le fera même si aucun module de traitement n'utilise cette information par la suite. Si un module nécessitant l'identification apparaît ponctuellement, l'information sera disponible.
– Il n'y a *pas de service de présentation* des agents, ni d'annonce sur la disponibilité des informations. Les agents cherchent les informations disponibles et les utilisent, sans phase de négociation pour obtenir ces informations (ajout/suppression). Les

agents apprennent l'existence des autres par l'intermédiaire des informations qu'ils utilisent.
- Il n'y a pas de résolution explicite des *dépendances dans le traitement* des information. Par exemple, un module C à besoin d'une information comprenant des données a et b produites respectivement par les modules A et B qui sont totalement indépendants. Avec le traitement systématique des informations, le module A peut utiliser l'information qui contient (ou pas) la donnée b et réciproquement. Le module C trouvera des informations sans avoir à demander à A et B de travailler ensemble
- Il est difficile dans un système où le contexte évolue continuellement d'établir la durée d'un traitement a priori sans connaître les traitements nécessaires à son accomplissement. L'établissement de la *durée de traitement* a posteriori est simple, il suffit de comparer la donnée rendue aux informations utilisées à la source. Comme nous sommes dans un système de traitement continu, le temps a posteriori est représentatif en moyenne du temps de traitement.
- il est plus difficile d'estimer a priori la *pertinence d'un traitement* dans un contexte fortement dynamique. Par exemple, si Alice était allée dans le salon, utiliser l'écran de la télévision aurait été pertinent.

Pour toutes ces raisons, nous utiliserons une approche par tableau noir pour le partage des informations. Cette solution est réalisable si le système utilise une centrale « domotique » ou si les modules de traitement *broadcastent* leurs informations après leur traitement ou bien leur recherche avant leur traitement.

Nous poserons une seule hypothèse d'ordre technique liée à l'utilisation d'un tableau noir pour les agents : la recherche est peu pénalisante pour les ressources du système, ce qui permet aux agents inactifs de continuer à chercher des informations sans que les ressources du système en soient affectées.

Dans les sections suivantes, nous centrerons la présentation sur l'auto-régulation de notre système. Nous ne développerons pas davantage les problématiques d'adaptation (ajout, suppression...) dans cet article.

3.1 Les agents

Notre architecture se compose d'agents qui partagent tous la même architecture interne (figure 2). Seuls les interfaces d'entrée et de sortie, le bloc de traitement, et bien sûr les valeurs de certains paramètres internes (figure 4) sont spécifiques. Le comportement des agents est totalement indépendant des fonctions de traitement qui les composent, ce qui permet l'utilisation de bibliothèques de traitement existantes et facilite ainsi la réalisation de traitements de données. La conception des agents se base sur la description :
- des données nécessaires, et éventuellement de contraintes associées (*filtre d'entrée*),
- de la fonction de traitement (incluse dans des *bibliothèques de fonctions*) à appliquer aux données collectées sur les informations,
- de l'assemblage des nouvelles données dans de nouvelles informations (*filtre de sortie*).

L'exécution d'un agent est linéaire. L'agent cherche des informations dans l'environnement ou dans des connexions spécifiques (tubes), et vérifie leurs contraintes (*interface d'entrée*). Si tout se passe bien, l'agent effectue le *traitement des données* et en renvoie de nouvelles par l'*interface de sortie*. Dans tous les cas, il prend en compte les messages qu'il a reçus et les modifications internes.

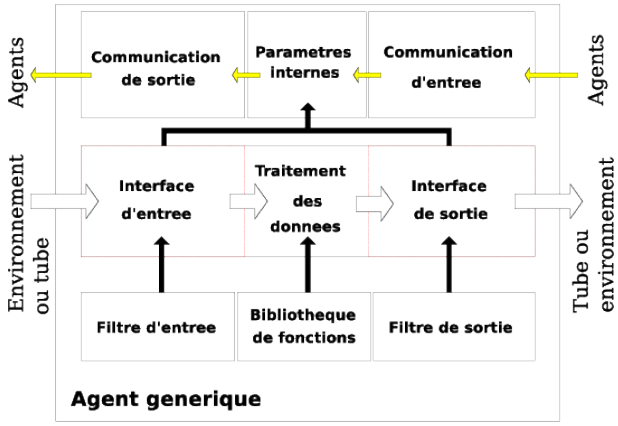


FIG. 2 – Schéma global des agents

3.2 Le partage de tubes d'information

Pour faciliter le transfert des informations entre l'agent qui les produit et celui qui les consomme, les agents peuvent utiliser des tubes nommés. On dit alors qu'ils sont *regroupés*. Ces tubes permettent aux agents consommateurs de ne pas chercher dans tout l'environnement les informations contenant les informations qui leur sont nécessaires, ce qui a pour effet de diminuer le temps de traitement (dans ce cas, les données du producteur ne sont plus accessible aux autres consommateurs). La stratégie des agents va donc les conduire à rechercher au maximum le regroupement avec les agents qui les précèdent dans la chaîne de traitement.

4 Gestion et contrôle du flux de données

Jusqu'à présent, nous avons exposé la conception et l'exécution des agents. La conception n'impose aucune façon d'organiser les agents entre eux et encore moins la façon de sélectionner un traitement plutôt qu'un autre. Cette liberté permet de différencier deux aspects : la conception des agents de traitement et la régulation des chemins de traitement.

Notre scénario initial présentait des capteurs et des traitements équivalents du point de vu de la fonctionnalité. Par exemple, les caméras de la cuisine et celle du téléphone ou bien l'extraction d'une position par rapport à son anticipation (pour la sélection des caméras, si la luminosité le permet).

Ceci permet au système de fournir différentes valeurs pour un seul contexte, et de permettre l'estimation de la pertinence du traitement. Pour améliorer l'efficacité, notre approche cherche à maximiser la pertinence des données qui sortent du système sans pour autant que le temps de traitement ne dépasse un temps fixé par l'utilisateur.

4.1 Pertinence et précision

Dans toute la suite de notre démarche, la pertinence jouera un rôle important. Elle définira le paramètre grâce auquel le traitement va se réguler. Pour ce faire, nous discutons dans cet section notre définition de la pertinence.

Dans le vocabulaire courant, la précision définit l'exactitude d'un fait alors que la pertinence est un jugement sur la qualité d'un fait.

Définir de façon objective la précision d'une donnée n'est pas toujours possible d'autant que cette précision peut changer au cours du temps surtout dans une application en temps-réel.

Nous pouvons caractériser la *fonction de précision* (qui attribue une précision à une donnée ou une situation), par la possibilité de qualifier automatiquement l'objet observé et par le caractère évolutif de l'objet observé.

Les exemples suivants donnent, pour différents cas, des indications sur la possibilité de quantifier la précision et son évolution au cours du temps :

Quantifiable et évolutif

La luminosité est une information que peut nous renvoyer une caméra. Les informations transmises par ce capteur évoluent en fonction de l'environnement extérieur. Dans ce cas on peut juger objectivement la luminosité d'une image et donc la précision avec laquelle il sera possible de distinguer des formes sur l'image.

Non quantifiable et évolutif

Les préférences de l'utilisateur varient au cours du temps et ne sont pas prévisibles. L'utilisateur peut préférer une luminosité faible à un moment puis vouloir qu'elle soit plus forte à un autre. Aucune précision ne peut être attachée à ce type de traitement, alors que l'utilisateur émet une préférence. La « fonction de jugement » de l'utilisateur évolue elle-même au cours du temps.

Quantifiable et non évolutif

Les résolutions différentes pour une même image en sont un exemple. Une image 200×200 est 4 fois plus précise qu'une autre en 100×100. Un tel cas de figure simple pourrait permettre de donner une précision aux fonctions de traitement fournissant ces résultats.

Une information peut contenir plusieurs données que l'on peut quantifier (comme la luminosité et la résolution d'une image). La comparaison des différentes précisions imposerait que la précision soit un vecteur, ce qui rend la comparaison difficile. De plus, la précision des informations doit représenter celle des traitements. Pour ce faire

il faudrait être capable de distinguer de manière décentralisée quels sont les traitements qui ajoutent le plus de précision aux informations.

Pour toutes ces raisons, nous n'utiliserons plus la notion de précision des fonctions de traitement mais la notion plus générale de *pertinence de traitement*. La fonction de pertinence est dépendante du contexte et de l'utilisateur. Une pertinence élevée n'est pas synonyme de précision forte, par exemple une image haute résolution est par définition plus précise qu'une image basse résolution, mais sur l'écran d'un téléphone portable cela n'a aucune importance. Par ailleurs, il est plus pertinent d'avoir le visage d'Alice sur une image basse résolution qu'une image précise sans Alice. De même il est plus pertinent d'avoir une image basse résolution si cela permet de l'analyser en respectant les contraintes de temps.

Cette notion de pertinence est traduite sous la forme d'un attribut attaché aux agents et aux informations traitées par eux. Les pertinences des agents et des informations évoluent dynamiquement au cours du fonctionnement du système selon des principes détaillés dans la section 5. Par ailleurs, l'utilisateur faisant partie du système, il peut à tout moment interagir et quantifier lui-même la pertinence du rendu du système.

4.2 Contrainte temporelle

Pour prendre en compte le temps, on ajoute dans chaque information traitée par le système un attribut représentant le temps écoulé depuis son arrivée dans le système. Chaque agent peut contrôler cet attribut pour évaluer l'information qu'il traite ou va traiter. Pour reprendre l'exemple de l'agent de traitement vidéo, un agent ne peut pas prendre une information qui date de plus de 20 ms (filtre d'entrée) ou ne pas rendre une information qui dépasserait 25 ms (filtre de sortie) empêchant la propagation d'informations inutilisables.

Avec une horloge commune, l'arrivée des informations dans le système est considérée comme la donnée temporelle de référence. Sans horloge commune, on peut ajouter les durées de traitement et d'attente/transition et ainsi obtenir la durée approximative totale de traitement d'une information. Chaque agent ajoute la durée pendant laquelle il utilisait l'information, l'environnement commun doit aussi ajouter la durée d'attente. Notre simulateur utilise une horloge commune.

Empreinte des informations

Chaque agent ajoute automatiquement son empreinte (une signature unique l'identifiant) sur les informations qu'il traite. Elle permet au système :

- de connaître l'ensemble des agents qui ont participé aux traitements de l'information et ainsi de remonter la pertinence de chaque agent ;
- à l'agent consommateur d'identifier le producteur d'une information (afin de lui proposer de se regrouper, ou d'identifier l'ensemble des ses producteurs pour leur envoyer des blâmes) ;

4.3 Régulation

Du fait que plusieurs chemins de traitement sont possibles pour traiter les informations, celles qui ressortent du système ont des pertinences différentes. Ces différences sont utilisées pour réguler les temps de traitement des différents chemins, plus un chemin d'agent est pertinent plus il aura du temps consécutif pour effectuer ses traitements.

4.3.1 Du point de vue des agents

Pour modéliser le comportement de nos agents nous nous basons sur une formalisation simplifiée de sentiments [Picard (1997)] [Sarmento *et al.* (2004)]. Le besoin des nos agents est simple : travailler dans de bonnes conditions. Le traitement est conditionné par deux choses : avoir des données à traiter et que les contraintes de temps soient satisfaites. Le comportement des agents s'axe principalement sur la notion de satisfaction. Quand un agent est satisfait il cherche à se regrouper avec ses agents producteurs pour améliorer la production et supprimer l'étape de recherche de données. L'insatisfaction se traduit principalement par l'envoi de blâmes à ses agents producteurs. Le comportement local des agents se résume en 4 (voir figure 4.3.1) états attachés à l'activité (actif/inactif) et à la satisfaction (satisfait/insatisfait) de l'agent.

Satisfait et actif. L'agent trouve des informations traitables. Il récupère l'empreinte du dernier producteur. Si le nombre des informations traitées par un même producteur dépasse un seuil, alors l'agent consommateur demande le regroupement. Si les informations qu'il trouve ne satisfont pas les contraintes de temps, il augmente son niveau d'insatisfaction.

Satisfait et inactif. L'agent ne trouve pas d'information traitable. Il augmente son insatisfaction. Au-delà d'une certaine durée d'inactivité, l'agent devient insatisfait.

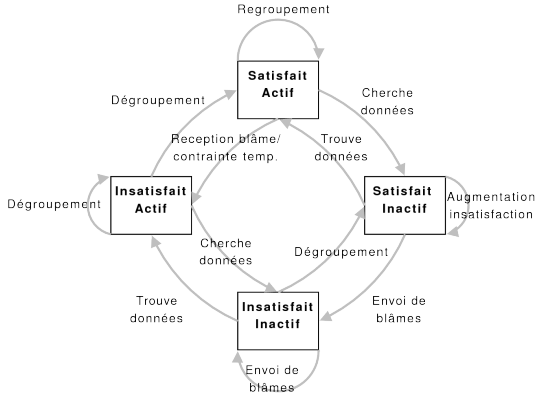


FIG. 3 – État des agents

Insatisfait et actif. L'agent trouve des informations traitables mais n'est pas satisfait. Les contraintes de temps ne sont pas correctes, les agents consommateurs sont insatisfaits et lui envoient des blâmes. Si l'agent était regroupé, il se dégroupe et réinitialise son insatisfaction (l'organisation des agents peut être la source de l'insatisfaction locale). Si c'est un agent simple, il envoie des blâmes aux agents producteurs qu'il connaît.

Insatisfait et inactif. L'agent ne trouve pas d'information et n'est pas satisfait. Il envoie des blâmes à ses agents producteurs.

Dans tous les cas, si les contraintes temporelles des informations qu'il utilise ne sont pas satisfaites, l'agent augmente son insatisfaction. De même lorsqu'un agent reçoit des blâmes, il augmente son insatisfaction.

4.3.2 Partie générique et partie programmable

Les figures 4 et 5 détaillent les différents paramètres et fonctions caractérisant l'état interne des agents et leur comportement. Nous présenterons par la suite certaines propriétés mathématiques de ces fonctions.

Var.	Nom long	Description
i	pénalité d'inactivité	quand un agent ne peut rien faire, son insatisfaction augmente de i
r	pénalité de blâme	augmente l'insatisfaction à la réception de blâmes
g	satisfaction de blâme	diminue l'insatisfaction après avoir envoyé des blâmes
p	pénalité de retard	augmente l'insatisfaction après avoir constaté un retard
k	rapport de satisfaction	rapport entre r et g
T	seuil d'insatisfaction	au dessus de cette valeur l'agent est insatisfait
d	insatisfaction	représente l'insatisfaction d'un agent
a	pertinence	représente la pertinence du traitement d'un agent
S	blâmes envoyés	nombre de blâmes émis par un agent par unité de temps
α	stabilisation de la pertinence	permet de changer la stabilité du système

FIG. 4 – Variables internes des agents

$At_i(j)$	attraction	attraction entre producteur i et consommateur j
$Sr_i(n)$	envoi de blâme	nombre de blâmes envoyés par l'agent i en fonction de sa pertinence n
$Rr_i(n, N)$	réception de blâme	nombre de blâmes N reçus par l'agent i en fonction de sa pertinence n
$Ac_i(N)$	pertinence	pertinence d'un agent i en fonction du nombres de blâmes émis N
$In_i(n)$	inactivité	augmentation de l'insatisfaction lorsque l'agent ne trouve pas de donnée

FIG. 5 – Fonctions des agents

4.3.3 Étude des paramètres

Les agents forment des chaînes de traitement, ce qui implique que lorsqu'un agent producteur ne travaille pas, les agents consommateurs qui dépendent de lui ne travaillent pas non plus. Les agents insatisfaits envoient des blâmes aux producteurs qui eux-mêmes transmettront leur insatisfaction. Nous utilisons un principe d'accumulation de blâmes le long d'une chaîne de traitement, ce qui offre la possibilité d'une gestion décentralisée le long de la chaîne.

L'objectif des fonctions détaillées dans la suite est de permettre aux agents d'avoir une représentation locale de leur pertinence qui soit représentative au niveau global. L'accumulation permet à un agent dans une chaîne de pouvoir exprimer à ses agents producteurs sa pertinence ainsi que la pertinence des agents suivants.

Lorsque l'agent ne trouve pas de données convenables, on dit qu'il est *inactif*. Son insatisfaction augmente en fonction de sa pertinence n de $In_i(n) = i\frac{k^n-1}{k-1}$. Lorsqu'un agent est insatisfait (son insatisfaction a dépassé le seuil) la réception de N blâmes entraîne une augmentation donnée par $Rr_i(n,N) = rk^{n-1}N$. Par contre quand l'agent est satisfait la réception est de $Rr_i(n,N) = rN$. On obtient le nombre de blâmes émis par l'agent i, quand il est insatisfait : $Sr_i(n,N) = \frac{k}{r}(In_i(n) + Rr_i(n,N))$.

La composition de deux agents i et j de pertinences a et b le long d'une chaîne donne :

$$\begin{aligned} Sr_i(a, Sr_j(b,0)) &= \tfrac{k}{r}\left(In_i(a) + Rr_i(a, Sr_j(b,0))\right) \\ &= Sr_i(a+b,0) \end{aligned}$$

Lorsque l'on compose des agents sous forme de chaînes, les agents producteurs interprètent le nombre de blâmes reçus par leurs agents consommateurs comme la mesure de leur pertinence.

$$Ac_i(N) = \frac{log\left(\frac{Nr(k-1)}{ik} + 1\right)}{log(k)}$$

Étude du paramètre k

Certains agents producteurs peuvent fournir des données à plusieurs agents consommateurs (comme E et A sur l'exemple 1(d)). Ces agents sont appelés *agents routeurs*. Ils jouent un rôle particulier puisqu'ils participent à plusieurs traitements. On peut montrer que le paramètre k permet de sélectionner le comportement global du système. Pour un agent routeur P, ayant n consommateurs notés C_1, jusqu'à C_n de pertinence a_1 à a_n, on a :

$$Ac\left(\sum_{j=1}^{n} Sr(a_j)\right) = \frac{log(\sum_{j=1}^{n}(k_j^a - 1) + 1)}{log(k)}$$

Étude pour $k \to 1$ En utilisant les développements en série entière nous avons simplifié la limite en 1 par :

$$\lim_{k\to 1} \quad Ac\left(\textstyle\sum_{j=1}^{n} Sr\,(a_j)\right) = \sum_{j=1}^{n} a_j$$

Le nombre de blâmes que l'agent routeur enverra à ses agents producteurs sera donné en faisant la somme de tous les agents consommateurs qui lui succèdent ($k \to 1$).

Étude pour $k \to \infty$ Soit $a_1 > a_2 > \cdots > a_n$, on conserve l'inégalité sur $k^{a_1} > k^{a_2} > \ldots > k^{a_n}$. La limite sur k donne : $\lim_{k\to\infty} \sum_{j=1}^{n} k^{a_j} = k^{a_1}$. On peut en déduire que : $\lim_{k\to\infty} Ac(\sum) = k^{a_1}$.

D'une autre façon, on peut écrire : $\lim_{k\to\infty} Ac(\sum) = max(a_j)$

Dans ce cas, le nombre de réprimandes que l'agent routeur enverra sera équivalente à la pertinence de sa chaîne de consommateurs la plus pertinente. La pertinence des autres agents consommateurs ne sera pas représentée dans celle du routeur. De manière générale, plus le paramètre k est grand et moins le système sera sensible aux chaînes de traitement peu pertinentes.

Nous ajoutons sur chaque pertinence un exposant $\alpha \in]0, \infty[$ (paramètre global) pour en modifier la prise en compte par les agents. Par exemple, deux agents de pertinence 2 et 10 voient leur pertinence ramenée à $2^{\frac{1}{20}} \approx 1,03$ et $10^{\frac{1}{20}} \approx 1,12$. Dans ce cas α facilite la découverte des différents chemins de traitement en permettant de ne plus distinguer, par la réception de blâmes, les pertinences des différents agents (quand $\alpha \to 0$). Augmenter α permet au contraire, d'accentuer la distinction entre des agents de pertinence proche (le rapport entre deux pertinences 9 et 10 vaut 1,1 et pour 9^5 et 10^5 ce rapport vaut 1,7).

Lorsque k et α sont petits, seule la topologie des chemins influence les choix des traitements. Au contraire, si k et α sont grands, le traitement sera presque exclusivement produit par le chemin le plus pertinent. La modification de ces deux paramètres peut être vue comme la température dans les algorithmes de *recuit simulé*. Lorsque l'architecture change beaucoup (ajout de nombreux agents, modifications de préférences de l'utilisateur...), la diminution de ces paramètres permet d'assouplir le choix entre les différents traitements similaires et permet de redistribuer rapidement la pertinence des agents.

Dépassement du temps de traitement Le temps joue un rôle particulier dans notre architecture. Lorsqu'un agent constate un retard sur une information, il augment son insatisfaction du paramètre p, ce qui a pour conséquence de diminuer sa durée de regroupement. Cela permet de ne pas laisser le système dans une configuration insatisfaisante.

5 Optimisation adaptée au flux de données

5.1 Principe de l'algorithme

L'objectif est d'attribuer aux agents une pertinence pendant l'exécution sans pour autant être capable de connaître la valeur exacte de chaque agent. Certains agents ont la capacité de « juger » la pertinence des informations qu'ils traitent. Ces agents sont appelés *contrôleurs*.

L'algorithme des contrôleurs est simple : pour chaque donnée que l'agent utilise, il regarde le rapport entre la « pertinence » pratique de la donnée (les agents ajoutent

aux informations qu'ils traitent leur propre pertinence) à une valeur théorique. Si cette valeur est supérieure à une tolérance, alors le contrôleur envoie à chaque agent qui a traité l'information le coefficient $pratique/theorique$. Les agents multiplient quant à eux leur pertinence par ce coefficient.

5.2 Exemple

La figure 6 reprend l'exemple donné dans la section 2.1.

On considère e_i (resp. a_i b_i c_i s_i) comme la pertinence de E (resp. A B C S) à l'instant i. Des informations transitent sur le graphe de E à S. Lorsque les informations arrivent dans S, elles peuvent avoir 3 valeurs de pertinence différentes.

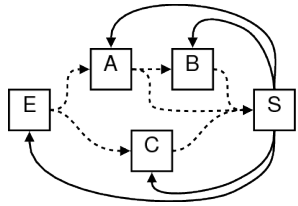


FIG. 6 – Jugement des pertinences locales

- $e_0 + a_0 + s_0 = p1_0$
- $e_0 + a_0 + b_0 + s_0 = p2_0$
- $e_0 + c_0 + s = p3_0$

S est en mesure de fournir la pertinence théorique ($t1$, $t2$ et $t3$) de chacune de ces informations. Si S obtient une donnée dont le rapport entre les pertinences théorique et pratique est supérieur au seuil de tolérance, S va envoyer aux agents qui ont participé à la création de cette donnée, un coefficient à appliquer à leur pertinence. Les valeurs p_i convergent vers celles de t_i. On peut rendre les valeurs théoriques et pratiques aussi proches que la valeur *seuil* le permet. Plus la valeur de *seuil* est petite plus la convergence des valeurs est longue.

L'étude expérimentale de cet algorithme nous a permis de constater que le besoin d'envoi de corrections augmente logarithmiquement en fonction du nombre de chemins de traitement. Le tableau suivant donne pour 1000 tests le nombre de changements nécessaires pour obtenir des valeurs pratiques à 1% des valeurs théoriques. Des valeurs théoriques sont attribuées à chaque agent entre 1 et 20. Les valeurs pratiques sont initialisées à 10. La simulation s'est faite sur 3 graphes. Entre un agent en entrée et un en sortie nous avons placé 2×2, 3×3 et 4×4 agents. Tous les agents d'une colonne sont les producteurs de la colonne suivante. Le tableau suivant donne pour un nombre d'agents, le nombre de chemins de traitement différents, la moyenne du nombre de changements nécessaires, le nombre de changements par chemin.

Notre algorithme de modification dynamique de la pertinence permet en peu de messages de faire converger la pertinence des agents vers une valeur reflétant la pertinence de leur traitement. Il permet aussi de supporter une reconfiguration en cours d'exécution.

Nb. Agent.	Nb. chemin	Moyenne	Moy. / chemin
6	4	7,7	1,9
11	27	33,0	1,2
18	256	73,6	0,3

6 Conclusion

Nous avons présenté dans cet article une approche de conception d'architectures de traitement d'informations dynamiques, comme support à des systèmes d'interaction homme-machine, notamment ceux de l'informatique ambiante. Cette approche doit permettre d'assurer d'une part un fonctionnement *adaptatif* par lequel le système peut découvrir et intégrer de nouvelles ressources, d'autre part un fonctionnement *auto-régulé*, par lequel le système peut adapter le « routage » des informations entre les différentes fonctions de traitement. Nous n'avons véritablement abordé dans cet article que la problématique d'auto-régulation, nous contentant de poser des hypothèses raisonnables pour la problématique d'adaptation, qui sera abordée dans une étude ultérieure.

Nous avons proposé d'encapsuler les fonctions de traitement dans des agents et de découpler le traitement des informations et la gestion des données de contrôle permettant la régulation du système. Cette régulation peut s'obtenir de manière décentralisée grâce à l'introduction d'une mesure de satisfaction ou de frustration chez les agents et par l'utilisation de messages permettant à ces agents de signaler aux autres une situation localement inacceptable. Cette approche permet de minimiser les ressources de calcul nécessaires à la gestion de la régulation, ce qui est particulièrement important dans un contexte de fonctionnement temps-réel. Par ailleurs, nous avons montré que certains paramètres du système permettaient de garder un contrôle assez fin du comportement dynamique du système.

Le travail s'oriente désormais dans trois directions complémentaires. Comme nous l'avons déjà souligné, la problématique d'adaptation a jusqu'à présent été laissée de côté même si différentes solutions ont d'ores et déjà été envisagées. Nous allons maintenant étudier ce problème plus précisément, notamment en liaison avec les problématiques de composition et d'adaptation de services Web. La deuxième direction de recherche concerne l'auto-régulation, pour laquelle nous nous attachons à passer d'un modèle de temps discret à un modèle de temps continu, de manière à s'approcher autant que possible, dans le modèle, de l'implémentation réelle. Enfin, pour valider complètement notre approche, cette implémentation réelle est en cours de mise en place, avec la construction d'un système matériel complet d'interaction multimodale, allant de la captation jusqu'au rendu en passant par la chaîne de traitement.

7 Bibliographie

Références

FERBER J. (1995). *Les systèmes multi-agents. Vers une intelligence collective*. Inter-Editions.

HARPER R., RODDEN T., ROGERS Y. & SELLEN A. (2008). *Being Human - Human-Computer Interaction in the Year 2020*. Microsoft Research Ltd.

HMS (1994). Hms requirements. *http ://hms.ifw.uni-hannover.de/*.

JOHNSTON W. M., HANNA J. R. P. & MILLAR R. J. (2004). Advances in dataflow programming languages. *ACM Comput. Surv.*, **36**(1), 1–34.

KOESTLER A. (1967). *The ghost in the machine*. London : Hutchinson.

PICARD R. (1997). *Affective Computing*. Cambridge, mit press edition.

PUCKETTE M. (1996). Pure data : another integrated computer music environment.

SARMENTO L., MOURA D. & OLIVEIRA E. (2004). Fighting fire with fear. *EUMAS 2004*, p. 627–634.

SILC J., ROBIC B. & UNGERER T. (1997). *Asynchrony in parallel computing : From dataflow to multithreading*. Rapport interne CSD-TR-97-4.

UUSTALU T. & VENE V. (2005). The essence of dataflow programming. In Z. AN HORVÁTH, Ed., *CEFP*, volume 4164 of *Lecture Notes in Computer Science*, p. 135–167 : Springer.

VALLÉE M., RAMPARANY F. & VERCOUTER L. (2006). Using device services and flexible composition in ambient communication environments. In *1st International Workshop on Requirements and Solutions for Pervasice Software Infrastructure (RSPSI)*, Dublin, Ireland.