Systèmes Informatiques d'aide à la décision distibués

Le modèle fédéraliste

Hubert Toussaint
htoussai@info.fundp.ac.be

Résumé Beaucoups de situations nécessitant une prise de décision rapide interviennent dans des environnements dynamiques et peu prévisibles. Les situations d'urgences médicales ou militaires en sont deux exemples. Ces environnements sont caractérisés par la dispersions des données parmi un ensemble d'intervenants repartis géographiquement. Contrairement aux outils classiques d'aide à la décision, les SIAD distribués tentent de répondre au besoin de mise en relation de ces données décentralisées.

Ce document présente une technique utilisée pour atteindre cet objectif de coordination : le modèle fédéraliste. Ensuite, nous présenterons une approche critique du modèle de coopération fédéraliste dans le domaine des SIAD distribués .

Key words : SIAD distribué, approche fédéraliste, distribution.

1 Introduction

Ce document propose une approche critique du modèle de coopération fédéraliste dans le domaine des SIAD distribués. Pour de plus amples informations sur ce modèle, le lecteur peut consulter [1] et [2].

Après avoir introduits les notions de SIAD et de SIAD distribué, nous présenterons les principaux modèles de communication ainsi que le framework basé sur le modèle fédéraliste développé dans [1]. La dernière section commentera certains points de l'architecture de ce framework.

1.1 SIAD

Le concept de système informatique d'aide à la décision (SIAD) est extrêmement vaste et sa définition dépend bien souvent du point de vue de l'auteur. Nous prendrons ici comme convention qu'un SIAD est *"un système informatique procurant une aide*

dans le processus de la prise de décision" [1].

Cette définition peut être précisée de la manière suivante : un SIAD est *"un système d'information interactif, flexible et adaptable, développé spécialement pour aider à la solution de problèmes de management peu ou non structurés. Il utilise des données, fournit une interface simple et autorise le manipulateur à exprimer ses propres opinions."* [1].

De la même manière que les définitions varient en fonction des auteurs, il n'existe pas d'architecture standard pour un SIAD. Chaque auteur reprend sous le terme de SIAD un ensemble plus ou moins étendu de composants. Ce document se base sur l'architecture en 5 composants proposée par Marakas [4] : le système de gestion des bases de données (SGBD), le système de gestion des modèles (MBMS), le moteur (KE), l'interface utilisateur et l'utilisateur (figure 1.1).

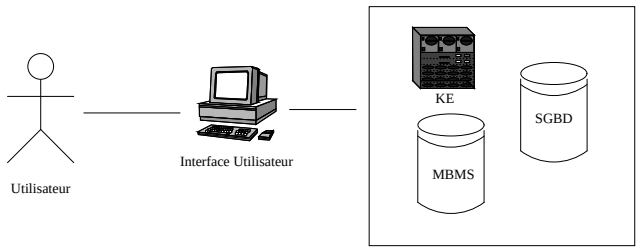

FIG. 1. Architecture d'un SIAD selon Marakas ([4]).

Les principaux composants d'un SIAD sont le système de gestion des bases de données (SGBD), le système de gestion des modèles (MBMS), le moteur (KE), l'interface utilisateur et l'utilisateur.

1.2 SIAD Distribué

La tendance actuelle à la globalisation à transformé la manière de prendre les décisions : les organisations sont maintenant bien souvent présentes sur plusieurs sites répartis à différents endroits de la planète. En conséquence, les données, les interfaces et les utilisateurs se situent bien souvent dans des lieux géographiquement distincts.

Il faut remarquer ici que le concept de SIAD n'est pas incompatible avec la distribution des connaissances et des utilisateurs, mais simplement que l'architecture classique d'un SIAD repose sur une source centrale de synchronisation (bien souvent il s'agit d'une base de données centrale). Cet élément représente un point central susceptible de mettre tout le système hors-service en cas de problème, que ce problème soit local (comme une panne matérielle) ou externe (comme une panne réseau).

L'évolution du concept de SIAD vers celui de SIAD distribué s'est principalement produite au niveau de la topologie utilisée pour connecter les différents composants. On peut définir un SIAD distribué comme *une collection de composants ou de services – hardwares ou softwares – ,organisés au sein d'un réseau dynamique et non-fiable, coopérants dans le but d'aider à la prise de décision* [2].

2 Schémas de communication

Les trois principales topologies utilisées pour la communication sont la topologie centralisée, la topologie hybride et la topologie décentralisée (figure 2). La topologie hybride représente une étape intermédiaire entre les deux autres topologies.

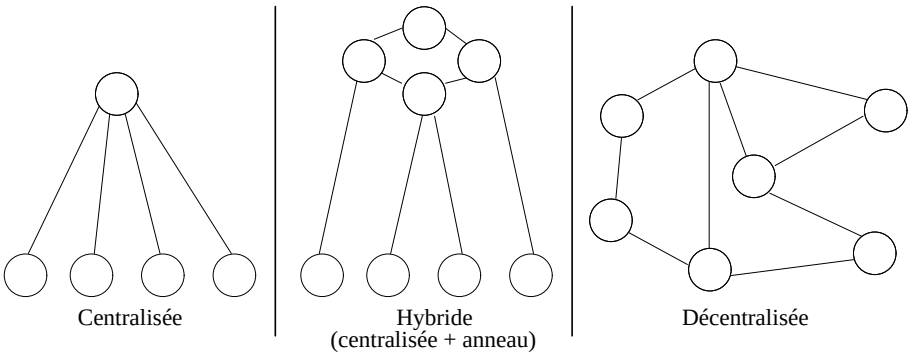

FIG. 2. Les trois principales topologies de communication

La topologie centralisée permet un contrôle aisé sur les données et les utilisateurs du SIAD. Bien souvent, le rôle de coordination centrale est joué par la base de données. Comme annoncé plus haut, ce choix de topologie rend le système particulièrement fragile face à une défaillance du composant central. En outre, l'évolutivité d'un tel système

est limitée : l'augmentation de la capacité ne peut se faire que par un remplacement du composant central par un composant plus puissant.

A l'inverse, la topologie décentralisée est tolérante aux fautes (dans une certaine mesure), et extensible. En effet, l'extension du système peut être réalisée par l'ajout de nouveaux composants sans devoir suspendre le fonctionnement de l'ensemble. De même, la réplication des composants "vitaux" entre les différents noeuds permet d'assurer une certaine capacité de tolérance aux fautes. Le principal désavantage de cette topologie réside dans la difficulté à connaître et à maîtriser l'état du système : il est ainsi relativement difficile de gérer les autorisations d'accès.

Afin de résoudre ces problèmes, la topologie hybride isole quelques composants jugés "centraux' à l'architecture dans un anneau. De cette manière ces composants sont clairement identifiés et donc plus faciles à administrer. Afin de ne pas mettre en danger l'ensemble du système en cas de panne de l'un de ces composants centraux, une certaine redondance des fonctionnalités et des connections est prévue.

3 Le modèle fédéraliste

L'un des ojectifs des SIAD est de fournir à chaque utilisateur un SIAD *sur mesure*. C'est à dire un SIAD qui soit spécifiquement adapté aux rôles et aux capacités de l'utilisateur tout en étant capable de s'adapter dynamiquement à ses demandes particulières.

Afin de répondre au mieux à ce souhait, le modèle fédéraliste utilise le concept d'*environnement de travail* [1] (figure 3). L'environnement de travail masque les différents composants du SIAD. Cette abstraction des composants physiques rend le réseau formé des environnements de travail des différents utilisateurs capable de s'auto-gérer.

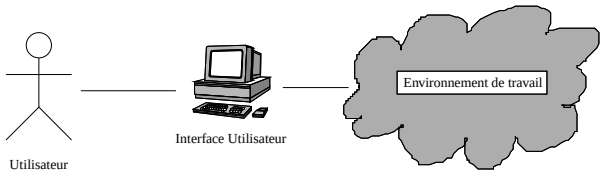

FIG. 3. Le concept d'environnement de travail.

Dans le modèle fédéraliste, chaque environnement de travail représente un noeud du réseau décentralisé : le seul pré-requis est que ce noeud doit être capable de commu-

niquer avec au moins un autre noeud du réseau. Les mécanismes internes du réseau sont alors capables d'adapter le réseau dynamiquement à l'arrivée de ce nouveau noeud. De la même manière, en cas de disparition d'un noeud, le réseau va gérer cette perte et se régénerer.

Le principe de fonctionnement d'un SIAD basé sur le modèle fédéraliste est le suivant : lorsque le premier utilisateur crée sont environnement de travail, celui-ci ne contient que les informations disponibles localement ainsi qu'un ensemble minimal de service globaux (comme un service de nommage public). A l'arrivée d'un utilisateur additionnel (figure 3), les environnements de travail des utilisateurs se mettent en rapport et s'échangent leur connaissances publiques respectives. De cette manière la connaissance globale est comuniquée à chaque utilisateur du SIAD et donc le retrait d'un utilisateur ne signifie pas la disparition des connaissances qu'il avait introduites dans le système.

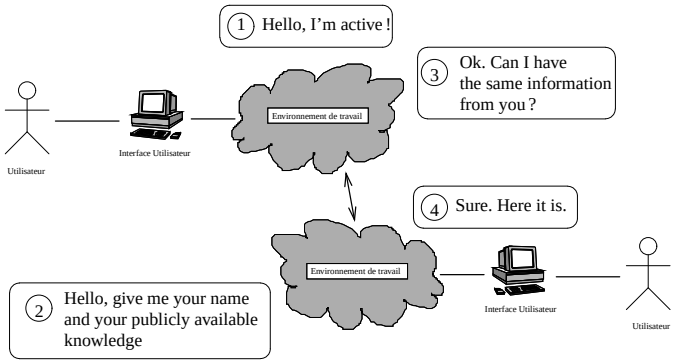

FIG. 4. Arrivée d'un nouvel utilisateur.

Les environnements de travail se mettent en relation et échangent leurs informations publiques.

La gestion des autorisations d'accès des différents utilisateurs utilise un système pyramidal : à la création du système, quelques utilisateurs (les administrateurs du système) définissent les droits des différents utilisateurs connus. Afin de ne pas fermer le système aux connaissances externes en n'autorisant que les seuls utilisateurs enregistrés à se connecter, le système de permission utilisé prévoit que certains utilisateurs possèdent le droit d'inviter des "extérieurs" à se connecter au système.

Le modèle fédéraliste repose entièrement sur la capacité des différents noeuds de s'organiser en un réseau cohérent ainsi que sur leur capacité à réagir de manière appropriée aux incidents. En effet, les utilisateurs ne disposent que d'informations concernant leur environnement de travail propre ; ils ne sont pas en mesure d'intervenir sur le réseau en lui-même.

4 Framework proposé par [1]

L'article à la base de ce document ([1]) présente une implémentation pratique du modèle fédéraliste au moyen du langage Java et de la technologie Jini en particulier. Nous nous limiterons ici a en présenter les aspects principaux :

- La séparation de la logique et de la présentation.
- L'adaptation par extension et par implémentation.
- La modularité.
- La mise en place d'une fédération Jini.
- L'utilisation d'objets distribués d'aide à la décision.
- L'utilisation d'un modèle basé sur les évènements.

La séparation de la logique et de la présentation permet de fournir une interface personnalisée à chaque utilisateur du SIAD tout en ré-utilisant les même composants logiques pour supporter ces interfaces. Ce choix évite de devoir multiplier de manière exponentielle le nombre de composants en charge de la logique du système. En outre, cette séparation permet de faire évoluer le framework dans le but de supporter des terminaux plus "exotiques" (par exemple des gsm, des pda,...).

L'adaptation par extension et par implémentation fait que la plupart des fonctionnalités du framework sont disponibles à la fois sous la forme de classes abstraites Java et sous la forme d'interface Java. De cette manière, la personne désirant ajouter une fonctionnalité au framework ne doit pas se soucier de la limitation de Java en ce qui concerne l'extension de classes.[1]

La modularité a pour but de permettre à une personne en charge de l'implémentation d'une fonctionnalité particulière de pouvoir se limiter à l'écriture d'une classe principale relativement simple ; cette classe centrale reposant en grande partie sur des modules et services déjà présents dans le framework. Ce choix apporte plusieurs bénéfices à l'ensemble du framework : d'abord, plusieurs objets clients peuvent utiliser le

[1] Une classe Java ne peut étendre qu'une seule autre classe, tandis qu'elle peut implémenter un nombre quelconque d'interfaces.

même objet du framework au même moment, réduisant par le nombre d'objets nécessaires. Ensuite une modification de la politique d'un objet/service est immédiatement répercutée sur tous les clients (par exemple une mise à jour des règles d'authentification dans le module de sécurité).

L'utilisation de la technologie Jini pour la distribution implique de devoir gérer un certains nombre d'activités en temps-réel (par exemple la recherche de composants et/ou de services). Le framework contient pour cela une classe générique encapsulant l'ensemble de ces taches de manière à en décharger au maximum le développeur. En outre, le choix de Jini permet de gérer de manière quasi-automatique l'ajout et le retrait de clients et/ou de services (auto-organisation et régénération).

L'utilisation d'objets distribués d'aide à la décision. La plupart des composants du framework utilisables pour la construction d'un SIAD spécifique sont fournis sous la forme d'objets distribués d'aide à la décision. Ce format d'objet a été développé pour être facilement gérable par les méthodes *JavaSpaces*[2]. Un *JavaSpace* peut être vu comme un repository d'objets partagés accessible au travers d'un réseau. Ces objets partagés sont dit *passifs* dans le sens où pour modifier un objet, un processus doit retirer l'objet du *JavaSpace*, le modifier puis le ré-insérer.

La distribution des messages entre les différents composants/services est une tâche fondamentale dans un système distribué. Le choix s'est ici porté sur l'utilisation d'un modèle basé sur les évènements. Le framework utilise les mécanismes de notification *JavaSpaces* pour implémenter cette distribution de la manière la plus intuitive et efficace possible.

5 Commentaires

Nos commentaires sur l'approche fédéraliste utilisée par [1] dans la réalisation de son framework des SIAD distribué suivent deux axes : le premier concerne l'applicabilité du modèle fédéraliste tandis que le second porte sur les choix architecturaux réalisés.

Concernant le modèle fédéraliste, il est intéressant de se poser la question des données erronées ou périmées. Comment un utilisateur peut-il retirer du réseau des données alors que chaque nouveau participant recoit une copie des données publiques globales ? La solution la plus simple serait de faire en sorte que les environnements de travail ne fournissent aux nouveaux arrivants que les données publiques locales. Néanmoins, cette

[2] Pour une présentation détaillée du concept de *JavaSpaces*, le lecteur peut consulter [5].

solution implique que le retrait d'un utilisateur du réseau signifie la disparition des données qu'il avait apportées et donc une diminution de la qualité de l'information fournie aux nouveaux arrivants.

De manière similaire, l'introduction au sein du réseau de connaissances volumineuses (en terme de place occupée) risque, lors de la phase de distribution de la connaissance, de saturer de manière assez rapide les moyens de communication disponibles. En outre, tous les participants ne disposent peut-être pas de la capacité nécessaire pour stocker l'ensemble de ces données volumineuses de manière locale. La persistance des ce type de données suppose alors une taille limite pour le réseau ; taille en dessous de laquelle des informations peuvent être perdues car aucun utilisateur n'est en mesure d'assurer leur conservation en totalité.

Un autre aspect intéressant concerne la capacité de ce modèle à supporter un grand nombre d'utilisateurs simultanés. Un nouvel arrivant supplémentaire doit communiquer successivement avec les espaces de travail de tous les autres utilisateurs. Cela représente une charge importante à la fois pour l'utilisateur entrant (qui va devoir attendre la fin de cette étape de synchronisation des connaissances avant de pouvoir commencer à utiliser le SIAD) et pour les utilisateurs déjà présents (dans l'hypothèse de l'arrivée continue de nouveaux utilisateurs, le flux de requêtes de synchronisation résultant risque de paralyser le réseau, tant en terme de bande passante qu'en terme de puissance de traitement).

Ces problèmes sont directement liés au choix du modèle fédéraliste. Dans un contexte de communication centralisée, ils pourraient être résolus de manière relativement élégante au moyen d'une base de donnée centrale. Le rôle de cette source centrale de données serait de conserver l'état courant des connaissances (permettant par là aux utilisateurs entrants de synchroniser leurs connaissances en une seule requête) et de décharger les environnements de travail des utilisateurs d'une partie des tâches de gestion du réseau. Cette évolution consisterait à remplacer le modèle décentralisé par un modèle hybride.

Concernant les choix architecturaux effectués par l'auteur, certains semblent étranges, voire contradictoires. Dans un premier temps, l'auteur choisi d'utiliser le modèle de Marakas [4] afin d'identifer les composants d'un SIAD. Mais la suite du développement montre que la plupart des composants ainsi identifiés sont noyés dans un objet abstrait : l'espace de travail. Seuls l'utilisateur et son interface avec le système échappent à cette abstraction. Ce choix rend l'utilisation d'un modèle distribué plus simple car il masque les objets candidats à la place de composants centraux (comme la base de données ou le gestionnaire de modèles). Mais masquer la complexité du système en fusionnant trois composants *a priori* distincts n'est probablement pas la meilleure façon de rendre ce

framework facile à utiliser et surtout à contrôler.

Le souci de modularité à guidé l'auteur tout au long de la réalisation du framework. Cependant, cet aspect a parfois été poussé à l'extrême : certains composants ont été décomposés en entités plus simples jusqu'à l'obtention de modules n'effectuant plus qu'une opération élémentaire. Ce choix risque de provoquer un impact négatif au niveau des performances ; en effet, tous ces modules sont des objets occupant chacun de la mémoire et nécessitant des opérations de gestion spécifiques (aussi bien par les machines virtuelles Java que par le framework).

Un autre élément intéressant est la manière dont les objets partagés sont utilisés aux moyens de la technologie *JavaSpaces* : pour modifier un objet, il faut le retirer du *repository*, le modifier puis le ré-insérer. Dans le cas d'objets fortement demandés et en particulier en cas de concurrence entre ces accès, ce comportement peut parfois entrer en conflit avec le comportement de régénération du réseau. Par exemple, si, pendant la durée où un objet X est retiré du *repository* afin d'être modifié, un autre utilisateur essaye d'accéder à ce même objet X, il va supposer que l'objet n'existe pas ou plus et le recréer (comportement d'auto-régénération du réseau). Cependant lorsque le premier objet X va vouloir être ré-introduit dans le *repository*, il sera refusé car considéré comme un doublon. Il existe donc un risque de perte d'information au travers de ce mécanisme.

Enfin, un aspect suprenant est la volonté de séparer l'interface du framework de la logique sous-jacente, justifiée par l'auteur *afin de permettre au framework de supporter des terminaux "exotiques" (gsm, pda, ...)* [1]. Ce choix permet en effet d'ajouter au framework de nouveaux composants d'interface permettant à un gsm d'obtenir l'accès aux informations. Mais comment un gsm sera-t-il en mesure de supporter les charges de calcul induites par la manipulation des données ? En effet, comme les composants logiques sont partagés par toutes les interfaces utilisateurs du framework, il n'est pas possible d'insérer une logique spécifique pour les terminaux "légers" ; logique qui permettrait aux calculs "lourds" d'être effectées sur une machine plus puissante (partage de la puissance de calcul en plus du partage des données).

6 Conclusion

La question centrale lors de l'utilisation d'un modèle fédéraliste (et donc distribué) pour l'auto-organisation d'un SIAD est de savoir "jusqu'à quel point un réseau dé-centralisé est-il capable de s'auto-organiser lorsqu'il s'agit de partager des informations relativement complexes ?". Car contrairement aux populaires réseaux *peer to peer* où les seules données échangées sont des fichiers, le framework doit assurer une organisation suffisament stable pour permettre le partage de modèles et de données et ce

sans avoir recours à des composants centraux. Par exemple, il n'est pas possible d'interrompre le travail d'un utilisateur du SIAD en cas de perte de connection (momentanée) avec le modèle utilisé alors que ce comportement est courant lors des échanges *peer to peer*.

Pour conclure ce document, nous pourrions nous poser la question suivante : "un framework de SIAD distribué se doit-il d'être absolument générique ?". En effet, le souci de rendre le framework complètement adaptable à toutes les situations possibles fait que la grande majorité des optimisations possibles ne peuvent pas être appliquées. On peut même ajouter que tenter d'optimiser une application pratique du framework revient à remettre en cause certaines hypothèses de départ de sa construction. Appliquer le framework consiste alors presque à le reconstruire en éliminant certains aspect non-utilisés ; cette reconstruction est facilitée par la présence d'un certains nombre de blocs "ré-utilisables", mais n'est pas aussi triviale que la simple instantiation d'un ensemble de composants.

Références

[1] A. Gachet, P. Haettenschwiler, A Jini-based software framework for developing distributed cooperative decision support systems. Soft. Pract. Exper. **33** (2003) 221–258

[2] A. Gachet, A New Vision for Distributed Decision Support Systems. proceedings of the DSIage 2002 Conference, Oak tree Press, Ireland, pp.341-352

[3] S. Goddart, S. Zhang, D. Lytle, A. Seavey A Software Architecture for Distributed Geaspatial Decision Support Systems. URL du projet : http ://nadss.unl.edu/

[4] GM. Marakas, Decision Support Systems in the Twenty-first Century. Prentice-Hall : Upper Saddle River, NJ, 1999, *xxi*, 506

[5] E. Freeman, S. Hupfer, K. Arnold, JavaSpaces Principles, Patterns, and Practice (The Jini Technology Series... From the Source) Addison-Wesley, 1999