Biographie de l'auteur

Michel Barbeau a obtenu un baccalauréat en informatique de l'Université de Sherbrooke en 1985, une maîtrise et un doctorat en informatique de l'Université de Montréal en 1987 et 1991, respectivement. Il s'est vu accorder la Médaille académique d'or du Gouverneur général du Canada pour souligner l'excellence de ses travaux au niveau doctorat. De 1987 à 1988, il a été assistant de recherche à l'INRS-Télécommunications. Depuis 1991, il est professeur au département de mathématiques et d'informatique de l'Université de Sherbrooke. Michel Barbeau est un passionné des télécommunications. Ses activités de recherche portent sur les aspects méthodologiques du développement de logiciel pour les systèmes de télécommunications.

Du modèle client-serveur vers le modèle des composants répartis

Département de mathématique et d'informatique
Faculté des sciences
Université de Sherbrooke
2500, boul. Université
Sherbrooke (Québec, CANADA) J1K 2R1

Résumé

Un système réparti permet à des utilisateurs, situés à des endroits différents, de coopérer et de mettre en commun leurs ressources. L'accès à distance aux ressources requiert un modèle d'interaction entre les utilisateurs et les ressources répartis. L'objectif de cet article est de présenter un survol de deux des grandes solutions possibles pour la réalisation de systèmes répartis. Il met en relief les caractéristiques de deux modèles: le client-serveur et les composants répartis. Le second correspond à une évolution du premier. Notre conception de chacun de ces deux modèles est décrite et comparée au moyen d'exemples, le RPC de Unix et CORBA.

Mots clés

Composants répartis, CORBA, logiciel client-serveur, programmation orientée objet, RPC, système réparti.

1. Introduction

Un système réparti est fait de ressources, de gestionnaires de ressources et d'utilisateurs de ressources. Une ressource peut correspondre à une imprimante, une fenêtre sur un logiciel ou un élément de données. Le partage de ressources par plusieurs utilisateurs géographiquement distribués est une des raisons principales pour lesquelles les systèmes répartis sont utiles. Le

partage des ressources est intéressant parce qu'il permet, entre autres, de réduire les coûts, d'échanger de l'information, de diffuser rapidement des données et de collaborer à distance.

Les réseaux de télécommunications constituent l'infrastructure sur laquelle repose les systèmes répartis. Concrètement, chaque ressource est la propriété d'un noeud du réseau et peut être utilisée à distance sur les autres noeuds grâce aux télécommunications. Un gestionnaire de ressources est un logiciel responsable de l'administration d'un type de ressources. Il possède une interface de télécommunications au travers de laquelle s'effectue l'accès et la mise à jour des ressources par les utilisateurs. Le gestionnaire met également en œuvre les politiques d'accès propres à chaque type de ressources (ex.: les imprimantes).

Tous les types de ressources ont trois besoins fondamentaux : la nomenclature, la résolution de noms et la coordination. La dimension nomenclature comprend un modèle qui détermine comment sont construits les noms des ressources. L'aspect résolution réfère au processus de traduction des noms de ressource vers des adresses de télécommunications. L'activité de coordination consiste à contrôler les accès et les mises à jour concurrents aux ressources afin d'en assurer la cohérence.

Les trois concepts de base des systèmes répartis (ressource, gestionnaire de ressources et utilisateur de ressources) peuvent être structurés et mis en relation suivant deux grands modèles, le modèle client-serveur et le modèle des composants répartis. Le modèle des composants répartis peut être vue comme une amélioration du client-serveur. L'objectif de cet article est de présenter un aperçu et de mettre en relief les caractéristiques propres à chacun de ces deux modèles.

La section deux passe en revue le modèle client-serveur. Pour faire le pont avec la section suivante, la conception par objets de logiciels client-serveur est abordée à la section trois. La section quatre discute du modèle des composants répartis. CORBA est présenté comme exemple d'environnement de développement par composants répartis. La notion d'objet d'affaire, mise de l'avant par les cadres de développement de composants répartis, est également introduite. Nous concluons à la section cinq.

2. Le modèle client-serveur

Le client-serveur est un modèle d'interaction entre processus répartis, qui sont des programmes qui s'exécutent. Il y a deux catégories de processus, les serveurs et les clients qui occupent respectivement les rôles de gestionnaires et d'utilisateurs de ressources. Un des faits marquants

de ce modèle est qu'un serveur est un fournisseur centralisé de la collection de ressources dont il a la gestion.

Un client est un processus qui exécute une tâche nécessitant l'accès à des ressources partagées. Typiquement, le client transmet des requêtes au serveur lorsqu'il a besoin d'accéder aux ressources. Pour chaque requête, si elle est correctement formulée, le serveur effectue le traitement nécessaire et retourne une réponse. Un serveur peut également jouer le rôle de client.

L'aspect centralisation du modèle client-serveur est une caractéristique qui n'est pas toujours souhaitable. Lorsque le noeud sur lequel s'exécute le serveur est inaccessible les requêtes des clients ne sont pas traitées. Pour atténuer l'impact de ce problème, on fait une distinction entre le concept de serveur et celui de service, qui peut être assuré par plusieurs serveurs. Ainsi, sur la plupart des réseaux locaux le service de résolution des noms est assuré par deux serveurs. Lorsque le primaire est non disponible celui de réserve prend la relève et assure la continuation du service. On obtient ainsi la transparence de fautes. Par ailleurs le modèle client-serveur, selon la conception susmentionnée, est bien adapté à l'accès aux ressources à distance mais ne convient pas toujours très bien pour celui aux ressources locales. Conséquemment, dans les systèmes client-serveur les programmes client utilisent deux modèles d'accès selon qu'il s'agisse de ressources locales ou non.

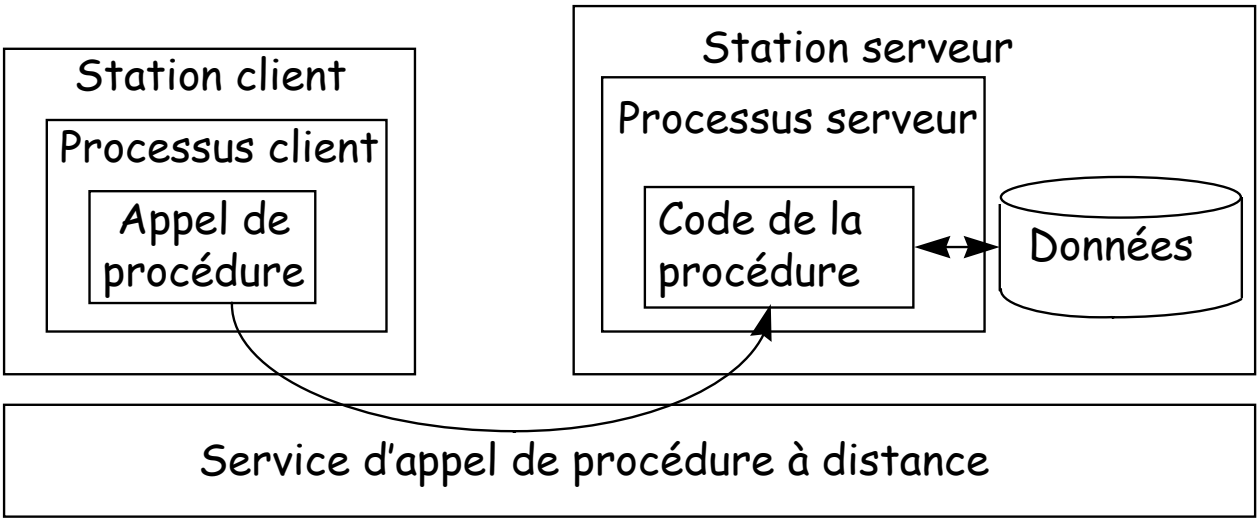

Figure 1. L'appel de procédure à distance

Unix est un système d'exploitation qui rend disponible sur le réseau les ressources d'un ordinateur selon le modèle client-serveur. Les ressources sont rendues disponibles grâce à des composants de Unix dont le *Network File System* (NFS), le *Network Information Service* (NIS) et

le *Remote Procedure Call* (RPC) [Sun 90]. Le modèle RPC est illustré à la figure 1. Le client et le serveur s'exécutent sur des machines différentes. La figure met en évidence deux faits. D'une part, les données sont logiquement séparées du processus serveur et de la procédure. D'autre part, toutes les données sont gérées par le serveur.

Les terminaux X sont également basés sur le modèle client-serveur. Un terminal X est un serveur auquel s'adresse un programme client pour transmettre des requêtes d'affichage de données. Le modèle client-serveur est évidemment très employé dans le secteur des logiciels de gestion.

3. La conception des logiciels de gestion, le client-serveur et l'approche objet

La plupart des logiciels de gestion sont conçus selon une des trois approches suivantes : la conception structurée, l'approche objet et l'approche hybride. La conception structurée et ses lacunes sont bien connues. Une bonne partie des difficultés découlent du fait que l'aspect donnée est séparé de celui des traitements. Il est maintenant largement bien admis que le modèle objet conduit à des logiciels mieux organisés et plus facilement réutilisables.

Actuellement, un bon nombre de logiciels de gestion sont réalisés avec des outils de développement par objets, par exemple Delphi [Borl 97]. Bien que ces langages soient orientés objet, ils mettent en fait en œuvre une approche hybride. Les concepts système (ex. : éléments graphiques, de communication, structure de données internes) sont effectivement modélisés par des objets alors que les concepts du domaine d'application (ex. : un client, un compte, un fournisseur) sont représentés par des tables du modèle de données relationnel. Les applications sont basées sur un modèle mi structuré mi objet. La réalisation de logiciels de gestion répartis selon un modèle objet pur est le sujet de la section suivante.

4. Le modèle des composants répartis

Le modèle des composants répartis est basé sur l'approche orientée objet [Booc 94]. Comme un objet, un composant est une entité logique qui contient des données et qui est capable d'exécuter des opérations sur celles-ci. Une partie des opérations est accessible à l'environnement du composant et constitue son interface. L'appel d'une opération par des utilisateurs du composant, des programmes ou d'autres composants de l'environnement, se fait par la transmission d'un message qui est pris en charge par l'interface qui se charge d'aiguiller la requête à la procédure associée à l'opération demandée. La procédure retourne éventuellement un message de réponse à l'appelant. Les composants sont définis par des spécifications de classes. La notion de classe

correspond également à une collection de composants apparentés. Une des forces du modèle objet, et du modèle des composants, est le concept de polymorphisme qui permet de traiter de la même façon des composants différents mais qui offrent des service similaires.

Maintenant, qu'est-ce que les composants ont de plus que les objets ? Pourquoi un nouveau terme ? Un objet est une unité logiciel réutilisable, sans peine, en autant qu'une ou plusieurs des conditions suivantes soient satisfaites : le programme d'accueil est écrit dans le même langage (ex. : C++), est sur la même plate-forme (ex. : Unix) et localisé sur le même noeud que l'objet. L'approche par composants répartis fournit une infrastructure qui facilite la réutilisation d'unités logiciel, appelées composants, d'un langage à un autre (ex . : C++ à Java), d'une plate-forme à une autre (ex. : Windows à Unix) et d'un noeud du réseau à un autre.

Dans le modèle des composants répartis toutes les ressources, locales ou non, sont représentées par des composants. Une syntaxe unique est utilisée pour les appels aux composants, qu'ils soient dans le même programme ou dans des programmes, des processus ou des noeuds différents. C'est la transparence d'accès. Deux autres aspects des composants méritent d'être mis en évidence, la nomenclature et la gestion. Concernant la nomenclature, chaque composant possède son identité propre. De plus, les composants répartis sont mobiles. Un composant peut changer de noeud d'accueil, pour améliorer la performance ou la tolérance aux fautes. Lorsqu'un composant change de position, son identité ne change pas. C'est la transparence de migration. Par ailleurs, contrairement au modèle client-serveur le modèle de nomenclature ne varie pas d'un type de ressources à un autre. Il est commun à tous les composants répartis.

L'entité qui est responsable de la gestion d'un composant réparti s'appelle un gestionnaire de composant. Il y a un gestionnaire de composant propre à chaque classe. Une copie du gestionnaire de composant de la classe correspondante est localisée avec chaque composant sur son noeud d'accueil.

Comme dans le modèle client-serveur, la transparence de fautes est assurée par la notion de service. Une collection de composants répartis sur des noeuds différents peuvent tous fournir les mêmes services. Les utilisateurs choisissent n'importe quel d'entre eux.

4.1 CORBA

Le *Common Objet Request Broker Architecture* (CORBA) est une réalisation du modèle des composants répartis [Bake 97, Mowb 95, Sieg 96]. Il est le fruit de la mise en commun des

efforts de plusieurs grandes entreprises de l'informatique, sauf Microsoft qui a décidé de développer son propre système, le *Distributed Component Object Model* (DCOM) [Brow 98].

CORBA fait intervenir pour la communication entre les utilisateurs de composants et les composants un concept nommé le *Object Request Broker* (ORB), voir la figure 2. Il transmet les requêtes des utilisateurs, les appels d'opération, aux composants (appelés simplement objets dans CORBA). Les utilisateurs et les composants peuvent être sur des noeuds différents, tourner sur des systèmes d'exploitation différents et être programmés dans des langages différents. Le ORB a la capacité d'acheminer des requêtes sur un réseau, d'un système d'exploitation à un autre et d'un langage de programmation à un autre.

Lorsque l'utilisateur et le composant ne résident pas sur le même noeud, il y a deux ORB qui interviennent. Celui sur le noeud de l'utilisateur qui transmet la requête à celui du noeud de du composant appelé. La communication entre les ORB se fait au moyen du protocole *Internet Inter-ORB Protocol* (IIOP). Une requête est transmise d'un site à un autre dans un paquet qui contient l'identité de le composant ciblé, le nom de l'opération appelée et des paramètres. Notez que c'est le composant qui est appelé par l'utilisateur et non le ORB qui joue tout simplement le rôle de intergiciel (*middleware*). Ce protocole repose évidemment sur TCP/IP. Une version avec contrôle d'accès et chiffrement est disponible.

La figure 2 met également en évidence le fait que les données et l'opération font partie d'un même composant et que le composant gère des données qui lui sont spécifiques. Concrètement, un composant peut s'exécuter à l'intérieur d'un *thread* ou d'un processus, qui peut contenir plusieurs composants. Un composant peut veiller et être toujours actif ou être activé uniquement au moment de son appel.

CORBA sépare clairement les notions d'interface et d'implantation derrière l'interface. L'implantation est interchangeable et derrière une interface peut se cacher une variété d'implantations. Ceci donne une certaine flexibilité. L'interface est spécifiée avec un langage propre à CORBA appelé *l'Interface Definition Language* (IDL). Les implantations peuvent être programmées dans différents langages dont le C, le C++ et le Java. Le développement et l'implantation d'un composant s'effectue comme suit. L'interface du composant est décrite en IDL. Cette description est passée à un processeur qui traduit l'interface dans le langage d'implantation. Le programmeur écrit dans le langage de programmation les procédures associées

aux opérations de l'interface. Le composant est compilé. Le code du composant contient les éléments nécessaires afin qu'il puisse se faire connaître au ORB au moment de son lancement.

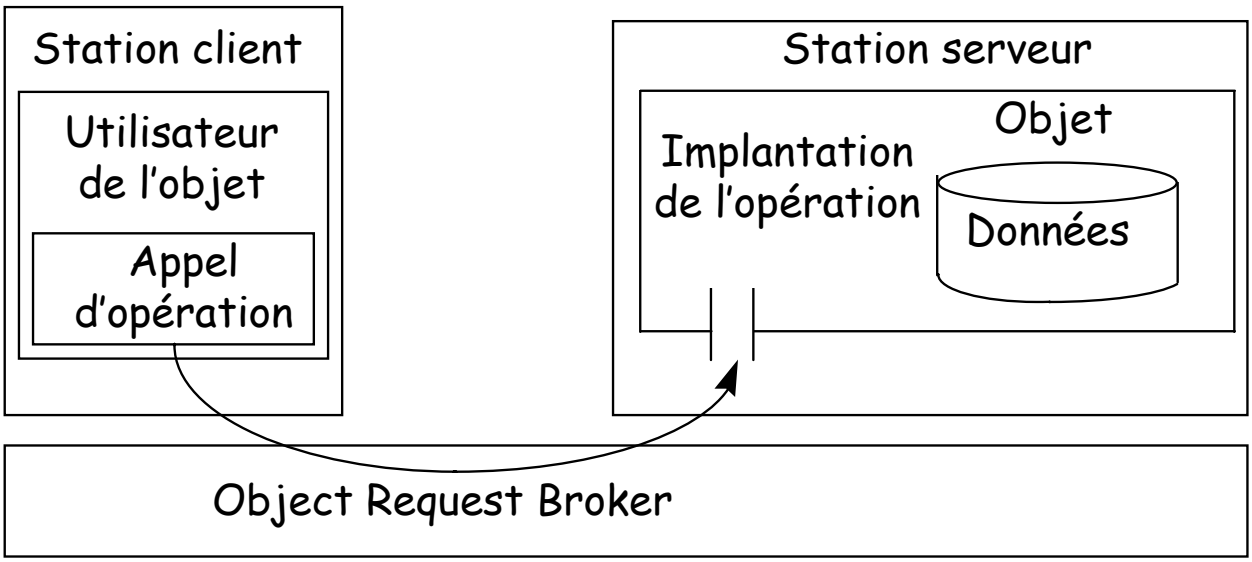

Figure 2. Le modèle CORBA

Notez que la notion d'IDL existe également avec le RPC pour la spécification des signatures d'opérations. Cependant, sous Unix par exemple, uniquement le langage C est disponible pour la programmation de l'implantation.

Des ORB CORBA sont disponibles pour un grand nombre de plates-formes. CORBA touche à un des points faibles du RPC : l'absence d'un standard de marché fort. Des traducteurs d'IDL sont disponibles pour un bon nombre de langages de programmation. Avec l'IDL, CORBA touche à une des faiblesses de la programmation par objets dans un langage comme le C++ : la communication avec des interfaces ad hoc (ex. : socket Unix ou Internet) entre objets qui sont dans des programmes différents. En effet, dans ces langages il y a un modèle de communication entre objets localisés dans le même programme, l'appel d'opération, alors que pour la communication à distance on utilise une interface de programmation, qui n'a rien à voir avec l'orientation objet, et qui est utilisée comme intermédiaire pour véhiculer les appels d'opération

Il est intéressant de noter que le développement par composants répartis en CORBA peut avoir ses particularités parce que les communications passent par le réseau. En orienté objet, on définit normalement une opération pour accéder à chaque attribut de l'objet. Lorsque plusieurs valeurs d'attribut sont requises, chacune des opérations correspondant à ces attributs est appelée. Afin de réduire le trafic et d'accroître la performance, il est intéressant de définir des opérations

qui extraient des groupes de valeurs d'attributs de composants répartis. Ce qui ne se fait normalement pas en programmation orientée non répartie.

La norme CORBA a été adoptée en 1991. Des implantations sont disponibles depuis 1993. Une implantation de CORBA est constituée au moins d'un ORB et d'un ou plusieurs traducteurs d'IDL. Certaines implantations fournissent également des services et des facilités normalisées mais optionnelles dans les implantations. Parmi ceux-ci se retrouvent des services à caractère général ; par exemple des services de noms, de sécurité, de rapport d'événement et de recherche de composant ; et des services plus spécifiques ; par exemple des services de transactions, de persistance.

4.2 Les objets d'affaire

Il est important de souligner que CORBA est plus qu'un outil de communication entre éléments de logiciel. En effet, CORBA aspire à la mise en œuvre de la notion d'objet d'affaire [OMG 94]. Un objet d'affaire est un modèle d'un concept de l'univers du discours des applications (ex. : un client, une facture). L'objectif est de faire de l'objet un composant réutilisable dans plusieurs applications. On le veut général et indépendant d'une application particulière. Cependant, un objet d'affaire a la capacité de coopérer avec d'autres objets d'affaire pour accomplir des tâches.

Lors du développement d'un système, les objets d'affaire apparaissent à l'étape d'analyse. Celle-ci conduit à la découverte de classes d'objets qui sont indépendants de l'implantation et qui correspondent à des concepts du domaine (ex. : un compte, une demande d'achat). Ceux-ci sont représentés par des composants CORBA. Ces composants deviennent ainsi disponibles aux utilisateurs. Par ailleurs, l'étape de conception d'un système conduit à la création d'artefacts (ex. : des listes chaînées) qui dépendent de la stratégie d'implantation adoptée. Par conséquent, ceux-ci ne sont pas forcément implantés par des composants CORBA. Les programmes utilisateurs peuvent contenir ou être des composants CORBA. Ce qui permet aux composants d'affaires de rapporter des événements aux utilisateurs par appels d'opération.

5. Conclusion

Le modèle client-serveur centralise l'accès aux ressources. La notion de service permet d'obtenir la transparence de fautes. Le modèle des composants répartis décentralise l'accès aux ressources et donne la transparence de fautes, la transparence d'accès, la mobilité, la transparence de migration et la nomenclature uniforme.

À la fois dans le modèle client-serveur et celui des composants répartis, les interfaces et les implantations sont séparées grâce à la notion d'IDL. Toutefois, en composants répartis (ex. : CORBA) un grand choix de langages d'implantation est offert parce qu'il existe un grand nombre de traducteurs d'IDL pour différents langages.

CORBA a été implanté par plusieurs entreprises de logiciels et certaines d'entre elles développent des implantations pour plusieurs architectures d'ordinateur. Toutefois, la réalisation du modèle des composants pur n'a pas encore atteint un niveau de maturité assez grand pour conduire à des logiciels offrant des performances comparables à celles obtenues avec l'approche client-serveur. L'approche hybride offre une bonne partie des avantages du modèle des composants et mène à des logiciels qui ont des niveaux de performance comparables à ceux obtenus avec des outils plus classiques, en particulier pour les applications de base de données.

L'approche de développement de logiciels au moyen d'objets d'affaire est certainement très attrayante mais fait actuellement l'objet de recherches et n'est pas encore tout à fait au point. Il s'agit toutefois d'un domaine qui progresse très rapidement.

Références

[Bake 97] Baker, S., CORBA Distributed Objects Using Orbix, ACM Press et Addison-Wesley, 1997.

[Booc 94] Booch, G., Object-Oriented Analysis and Design with Applications, Second Edition, Benjamin/Cummings, 1994.

[Borl 97] Delphi for Windows 95 & Windows NT - User's Guide, Borland International, Inc., 1997.

[Brow 98] Brown, N. and Kindel, C., Distributed Component Object Model Protocol -- DCOM/1.0, Microsoft Corporation, Redmond. Washington, 1998 (http://www.microsoft.com/oledev/).

[Sun 90] Sun Microsystems Inc., Network Programming, Sun Microsystems, Mountain View, CA, 1990.

[Mowb 95] Mowbray, T.J. and Zahvi, R., The Essential CORBA, John Wiley & Sons, New York, 1995. (voir également http ://www.omg.org).

[OMG 94] Object Management Group, Common Business Objects and Business Object Facility, OMG Document CF/96-01-04, Octobre 1994.

[Sieg 96] Siegel, J., CORBA Fundamentals and Programming, Wiley, New York, 1996.