LES MODÈLES DE RÉPARTITION

1. **Présentation des différents modèles de répartition**

Un modèle de répartition représente un ensemble d'abstractions permettant de spécifier les interactions entre composants selon une logique fonctionnelle plutôt qu'en termes de moyens techniques utilisés pour transporter l'information de l'un à l'autre. Par exemple, le modèle « objets répartis » décrit les interactions en termes d'appels de méthodes entre des objets déployés sur des nœuds différents.

Modèles de répartition : stratégies permettant aux applications réparties de communiquer.

Les modèles sont :

1.1 Communication par passage de messages

Il s'agit d'un protocole de communication de bas niveau (échange de flux non structurés d'octets) entre processus ou threads répartis, utilisé par les systèmes répartis primitifs. La sémantique qui anime cet échange est du type envoi/réception (conversationnel) : des messages indépendants unidirectionnels sont envoyés sur un réseau fiable (TCP) ou non fiable (UDP). On se situe « au ras du câble ».

La connexion par sockets symbolise le type de connexion « poste à poste » (« peer to peer » ou « d'égal à égal ») : l'envoi de message se fait au moyen de primitives spécialisées du système d'exploitation.

Le schéma du dialogue est celui de l'architecture Client/Serveur : un Serveur de socket attend des requêtes entrantes de la part d'un appelant sur un port spécifique, dès qu'il reçoit une requête, il effectue les calculs adéquats et retourne les résultats à l'appelant qui les attendait. La plupart des premières applications Client/Serveur ont été implémentée avec ce protocole.

on suppose l'existence des fonctionnalités suivantes :

Localisation des services : l'émetteur doit disposer d'une référence qui désigne le destinataire sans ambiguïté au sein de l'application répartie.

Synchronisation : l'échange de messages implique une synchronisation entre émetteur et destinataire qui doit être prévue par le programmeur

Hétérogénéité : pour faire coopérer des calculateurs hétérogènes, le programmeur doit définir lui-même une représentation commune des données, ainsi que les fonctions de conversion des représentations natives propres à chaque calculateur vers la représentation commune qui a été choisie (emballage des données), et réciproquement (déballage des données).

1.2 Appels de sous-programmes à distance (RPC : Remote Procedure Call)

Un développeur d'application peut désirer un service de communication plus structurée que l'échange de messages, afin de résoudre les problèmes récurrents de représentation des données et de synchronisation.

Le schéma Client/Serveur mis en œuvre par l'envoi de message est très similaire à un appel de sous-programme local. La sémantique qui anime cet échange est du type requête/réponse. La communication est synchrone : l'appelant attend les résultats de sa requête auprès d'un autre composant avant de poursuivre son exécution. Le but est d'étendre l'appel de sous-programme standard au cas où les composants « appelant » et « appelé » résident sur des nœuds différents d'un système réparti. On introduit alors un mécanisme d'appel de sous-programmes à distance.

on suppose l'existence des fonctionnalités suivantes :

Localisation des services : l'association d'un Client et d'un Serveur est appelée liaison (« binding »). L'information de liaison peut être figée chez le Client ou découverte au moment de l'exécution : liaison dynamique via un annuaire de services.

Hétérogénéité: comme dans le cas du passage de messages, les données manipulées doivent pouvoir être mises sous une représentation commune à tous les nœuds.

Passage de paramètres: du fait du passage de paramètres, un sous-programme à distance ne peut être appelé directement: il faut connaître sa signature (type des paramètres et des valeurs de retour). La solution est de générer automatiquement deux composants logiciels : une souche (« stub ») et un squelette (« skeleton ») de sous-programmes.

La souche est un module qui réalise l'emballage des paramètres (« marshalling ») de l'appel, et l'envoi de la requête au nœud sur lequel ce sous-programme réside. La souche se met ensuite en attente d'un message. Sur ce second nœud, le squelette, qui est en attente de tels messages demandant l'exécution de sous-programmes, déballe les paramètres « unmarshalling »), et appelle le corps du sous-programme réel. Lorsque cet appel se termine, il emballe la valeur de retour dans un nouveau message, qui est envoyé au nœud demandeur. La souche récupère et déballe alors la valeur de retour et la retourne à l'appelant, dont l'exécution se poursuit.

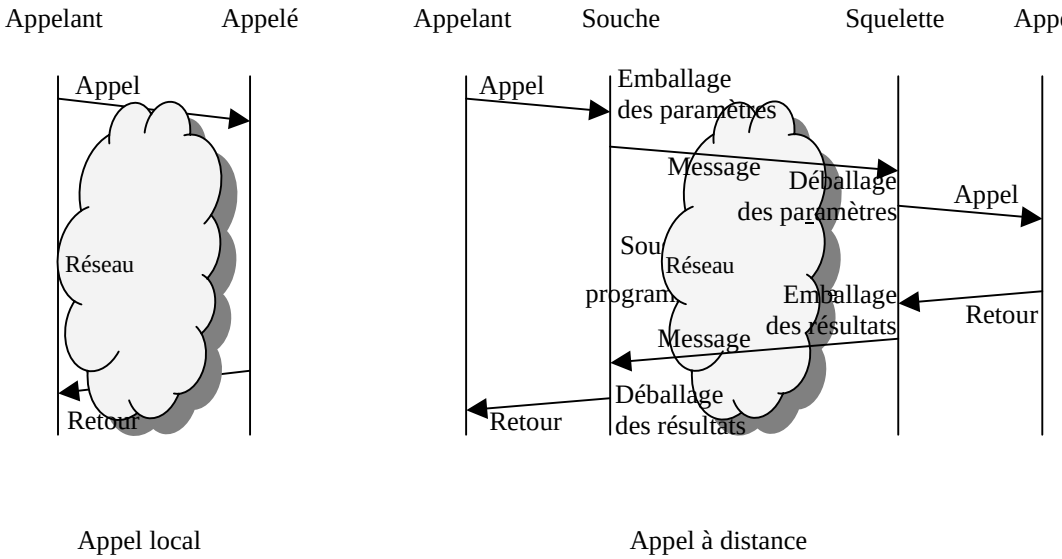

Fig.1 : Mécanisme d'appel de sous-programme

Du point de vue de l'appelant, tout se passe donc exactement comme si le sousprogramme appelé était local. De même, du point de vue de la mise en œuvre du sousprogramme, rien ne distingue un appel émanant d'un nœud distant d'un appel purement local.

et qui dispose de services évolués :

Service de nommage : enregistre les coordonnées des services distants.

Service de sécurité : fournit des fonctions d'authentification et de chiffrement.

Gestion des pannes : contrôles par timeout et RPC orientée connexion.

1.3 Les objets répartis et l'invocation de méthodes à distance (RMI : Remote Method Invocation)

Par rapport à la programmation structurée, la Programmation Orientée Objets, en introduisant les concepts d'encapsulation, de polymorphisme et d'héritage, est beaucoup mieux adaptée à la création de systèmes Client/Serveur flexibles parce que les données et la logique de fonctionnement (méthodes) sont « encapsulées » dans les objets, ce qui leur permet de s'implanter n'importe où dans un système réparti. La granularité de la distribution est améliorée.

Une application se présente désormais comme un ensemble d'objets qui s'envoient des messages pour appeler les méthodes désirées.

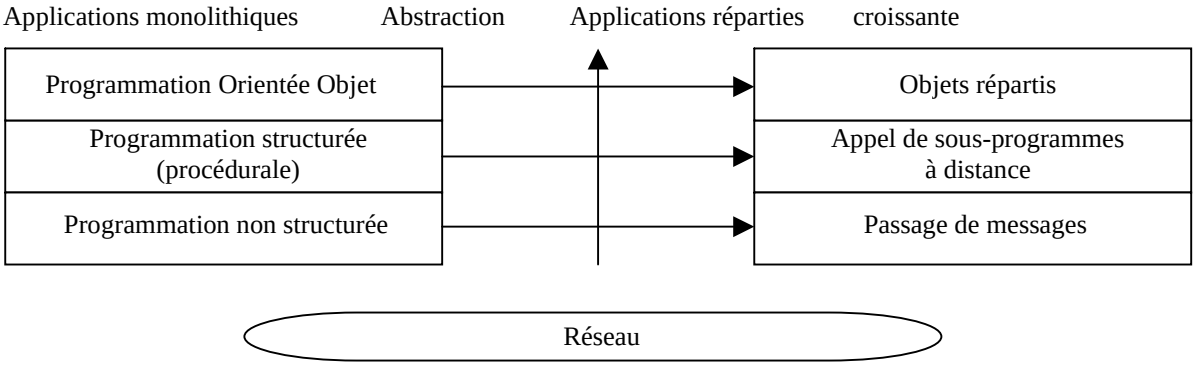

Fig.2 : corrélation entre méthodes de développement et modèles de répartition

Objets répartis :

Un objet « classique » n'a d'existence que dans un seul programme (seul le compilateur qui les crée connaît leur existence).

Un objet « réparti » est un objet qui peut vivre n'importe où sur le réseau et sur lequel il est possible d'invoquer à distance une méthode de l'objet.

Au niveau fonctionnel, les objets répartis peuvent être vus comme des composants logiciels autonomes (« boîtes noires ») : fragments de code qui ne sont pas liés à un programme, un langage, un système d'exploitation ou un réseau particuliers. Les composants sont des objets libérés des chaînes qui les liaient à un langage ou une plate-forme particulière. Une application répartie se présente alors comme une composition de composants. Dans les systèmes à objets répartis, le composant est l'unité de travail et de répartition.

Invocation de méthodes distantes RMI :

Pour effectuer un appel de méthode à distance, l'appelant doit disposer d'une référence identifiant l'objet cible. Une référence distante, désignant un objet cible dans une application répartie, est une information communicable entre nœuds de l'application qui désigne sans ambiguïté cet objet, et permet d'effectuer des appels de méthodes à distance. Un objet réparti est en fait un pointeur sur un objet distant.

L'exécution d'un appel de méthode sur un objet distant se déroule en deux temps :

1. l'appelant utilise la référence distante désignant l'objet pour déterminer sur quel nœud celui-ci réside,
2. il peut alors composer et émettre le message représentant l'appel de la méthode, comme le ferait une souche cliente dans le cas d'un appel de sous-programme à distance. En plus de l'identité du sous-programme appelé et des paramètres, le message doit contenir l'identité de l'objet destinataire.

Le modèle « objets répartis » suppose l'existence des fonctionnalités suivantes :

Localisation des services : une mise en œuvre du modèle « objets répartis » comporte donc en principe :

- Un moyen d'affecter à un objet une référence distante. Une telle référence doit identifier l'objet au sein de l'application répartie : nœud sur lequel il réside, ainsi que l'identité de l'objet.
- Un mécanisme permettant d'exécuter un appel de méthode sur un objet connu par une telle référence, au moyen d'un double aiguillage (d'abord en fonction du nœud destinataire, puis en fonction de la classe de l'objet).
- Un moyen de diffuser ces références sur le réseau.

Hétérogénéité : convenir d'un modèle de type global, permettant d'associer à chaque référence d'objet, la détermination de l'interface offerte par l'objet qu'elle désigne : des mécanismes de conversion de types doivent être offerts, permettant la conversion d'une référence d'une classe en une référence d'une autre classe (on ne parle plus de conversion entre types de données).

1.4 Mémoire partagée répartie (DSM : Distributed Shared Memory)

Une application répartie, basée sur la mémoire partagée répartie, procède selon un modèle de partage d'informations, et non plus d'échange d'informations comme dans les cas précédents. Les zones de mémoire partagées ne sont pas localisées sur un nœud particulier mais dupliquées sur chacun des nœuds qui y accèdent. Des copies des objets manipulés sont conservées sur différents nœuds (à l'opposé des modèles précédents où les abstractions du modèle sont localisées sur un nœud particulier). Les mécanismes d'échange d'informations sont dissimulés par l'exécutif de répartition qui assure la synchronisation des copies locales des informations partagées.

Existence des fonctionnalités suivantes :

Localisation des services : pour le développeur, ce modèle de répartition est transparent car il n'a pas à se préoccuper de la localité des différents composants. Tout se passe comme si ceux-ci s'exécutaient sur une même machine, et accédaient aux mêmes données.

Communication : Les échanges de messages nécessaires sont déterminés par le choix d'un modèle de cohérence : ensemble de propriétés que doivent vérifier les accès en lecture et en écriture à un objet partagé donné, et aux copies locales de cet objet dont dispose chaque nœud.

Hétérogénéité: tout se passe comme si les différents composants accédaient aux mêmes données (modèle de partage d'informations).

1.5 Files d'attente de messages (MOM : Message Oriented Middelware)

Mécanisme qui permet d'échanger des messages dans un système Client/Serveur au moyen de files d'attente de messages. Les applications communiquent sur le réseau en déposant et retirant simplement des messages dans des files d'attente (paradigme producteurs/consommateurs). Ce système permet aux Clients et au Serveur de communiquer de manière asynchrone sans être liés par

une connexion logique. Le Serveur et le Client peuvent opérer à des vitesses et à des moments différents. Un consortium MOM a été créé en 1993 pour normaliser le Middleware orienté messages.

On qui suppose l'existence des fonctionnalités suivantes :

Localisation des services : la plupart des implémentations permettent à l'émetteur de désigner le nom de la file d'attente des réponses.

Hétérogénéité: des descripteurs de format indiquent au récepteur comment interpréter les données du message.

Communication : MOM masque aux applications toute la partie communication.

Qui dispose de services évolués :

- Service de nommage: nommage hiérarchique.
- Service de sécurité: fournit des fonctions d'authentification et de chiffrement.
- Gestion des défaillances: les files d'attente persistantes (sur disque, opposées aux non persistantes en mémoire) offrent un premier niveau de tolérance aux pannes. Une forme de protection transactionnelle permet à une file d'attente de participer à un protocole de validation à deux phases. Les messages peuvent être re-routés vers des files d'attente de remplacement en cas de défaillance du réseau.

2.2. Fonctionnalités complémentaires des modèles de répartition

Dans un souci de cohérence et d'homogénéité, on pourrait s'attendre à ce qu'un certain nombre de fonctionnalités soient intégrées par les modèles de répartition Ces fonctionnalités sont généralement remplies par le Middleware mais de façon spécifique pour chacun. Il s'agit des fonctionnalités suivantes :

2.1 Gestion des exceptions

Un mécanisme d'exceptions peut être utilisé pour signaler à un composant appelant qu'un appel de sous-programme ou de méthode s'est terminé d'une façon anormale. Si le langage hôte supporte lui-même un mécanisme d'exception, ce message peut être traduit en une exception native qui sera propagée au composant applicatif appelant.

2.2 Avortement d'appels distants

L'utilisateur peut souhaiter annuler avant l'issue normale de son exécution un appel de méthode ou de sous-programme (surtout dans le cas de systèmes temps réel). Comme la levée d'exceptions, l'avortement est lié au mode d'interaction par requête/réponse, et ne s'applique pas au cas du passage de messages.

2.3 Appels unidirectionnels

L'utilisateur peut désirer poursuivre son traitement local sans attendre la confirmation de fin d'exécution de l'appel à distance. Un appel unidirectionnel est donc la construction qui, dans une application répartie basée sur l'appel de sous-programmes distants ou les objets répartis, se rapproche le plus d'un simple passage de message.

2.4 Transactions

Des ensembles d'échanges (messages, appels de sous-programmes ou de méthodes, actions sur un espace de stockage partagé) peuvent être groupés au sein de transactions, afin de préserver certaines propriétés de cohérence globale du système. Les propriétés généralement offertes par les mécanismes de transactions sont connues sous le nom d'ACID (Atomicité, Cohérence, Isolation, Durabilité).