Framework d'implémentation de services distribués

Eric Cariou

Master Technologies de l'Internet 1ère année

Université de Pau et des Pays de l'Adour
Département Informatique

Eric.Cariou@univ-pau.fr

Algorithmique distribuée

- But des TPs du module de Systèmes Distribués
  - Implémenter et tester les algorithmes distribués vu en cours et en TD
- Contexte de définition et validité de ces algorithmes
  - Système = ensemble de processus
    - Se connaissant tous mutuellement : identificateurs
  - Canaux de communication point à point
  - Modèle de système distribué : synchrone ou asynchrone
    - Temps de propagation des messages variés et potentiellement très longs et non bornés
  - Modèle de fautes
    - Pertes de messages
    - Crashs de processus

Algorithmique distribuée

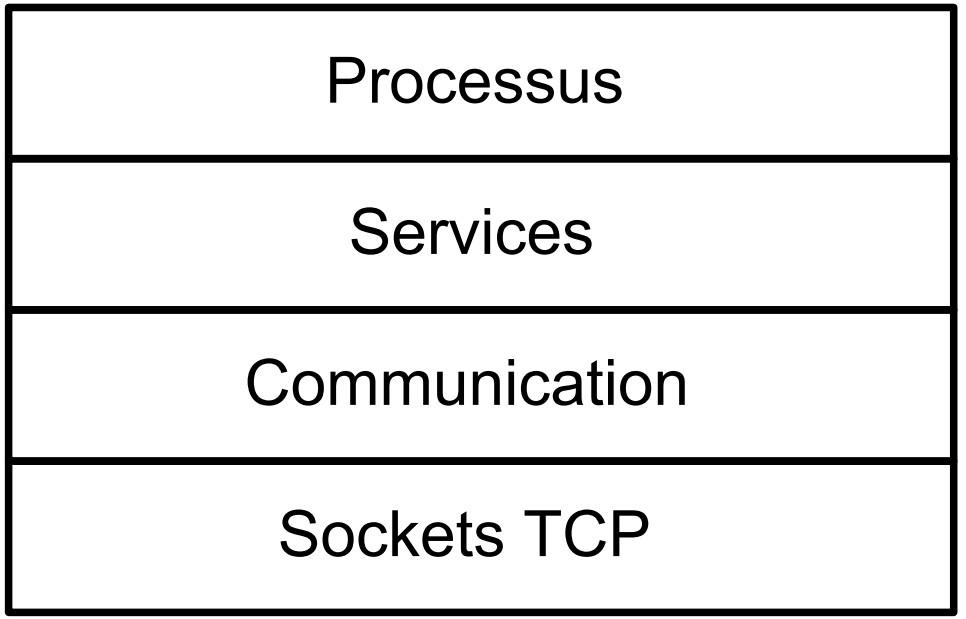
- Programmation des algorithmes : Java
- Contexte des salles de TPs
  - Réseau local rapide et fiable
    - Temps de propagation des messages très courts et bornés
    - Même en UDP non fiable, pas de perte de message
  - Pas adapté à simuler différents modèles de canaux ou de systèmes (synchrone ou asynchrone, avec ou sans fautes)
- Connaissance mutuelle des processus entre eux
  - Pas de solution native en Java pour attribuer des identificateurs automatiquement et les partager entre tous les processus
- Communication point à point
  - Possible avec sockets TCP/UDP ou Java RMI mais nécessite une encapsulation pour un usage adapté aux primitives de communicaiton et aux identificateurs de processus
- D'où l'utilisation d'un framework de communication

Framework de communication

- Implémentation en Java standard
  - Pas d'usage de librairies particulières
- Buts principaux
  - Support à la réalisation de services implémentant des algorithmes distribués
  - Basé sur une couche de communication
    - Communication point à point entre deux processus
    - Simulation d'un système particulier : gestion de délais de propagation et de perte des messages
  - Offre un service d'identification
    - Chaque processus reçoit un identificateur et connait la liste des identificateurs des autres processus

Framework de communication

- Implémentation en couches



- Communication
  - Gère l'envoi/réception de messages via des sockets TCP
  - Simule le modèle de communication : ajout de temps de propagation et de pertes de messages
- Services
  - Ensemble des services implémentant les algorithmes distribués utilisés par le processus

Fonctionnement général

- Chaque service (identification, diffusion, mutex, ...)
  - Implémente une interface Java définissant le service
    - Méthodes appelées par le processus pour utiliser le service
  - Envoie et reçoit des messages avec les autres services homologues sur d'autres processus pour réaliser l'algo
    - En utilisant la couche de communication
- Les processus peuvent aussi s'envoyer directement des messages sans passer par un service particulier
- Les messages échangés sont donc typés pour savoir à qui ils sont destinés
  - Directement d'un processus à un autre processus
  - D'un service (associé à un processus) à un service homologue (associé à un autre processus)

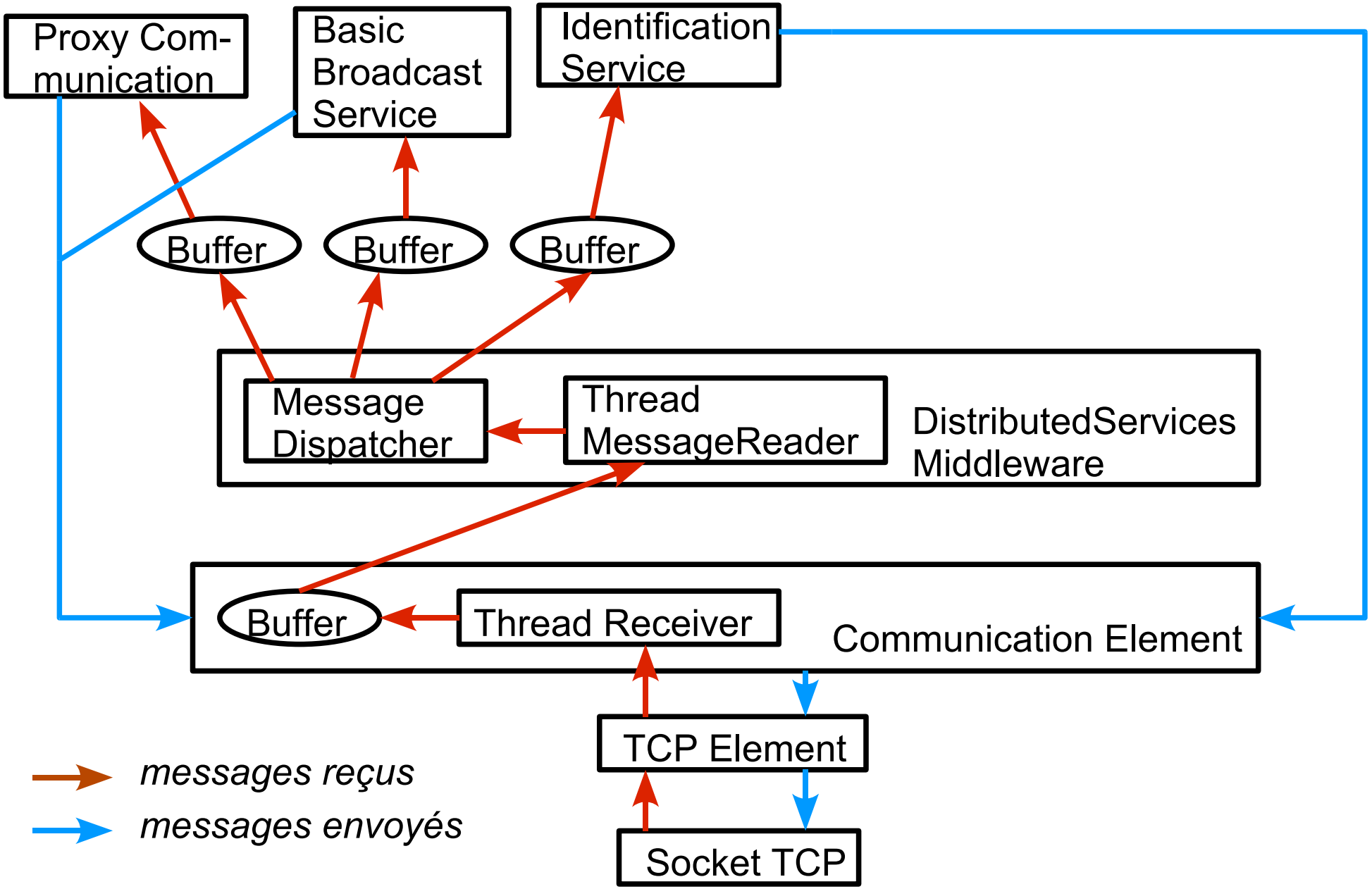
Fonctionnement général

- La couche de communication offre des primitives d'émission/réception de messages
  - En émission, on type le message à envoyer
  - Selon les critères de simulation du système
    - Attend un certain temps avant d'envoyer le message : délai de propagation
    - N'envoie pas le message : perte de message
  - En réception, on dépose le message au bon service ou au processus directement
    - Via un buffer de messages associé à chaque service ou au processus directement
- Identification d'un processus
  - Abstraite : un seul numéro
  - Concrète : extension de ce numéro avec le couple @IP/port de la socket TCP de la couche de communication

Fonctionnement général

- Envoi d'un message
  - Ouverture d'une connexion sur la socket TCP du destinataire, envoi du message puis fermeture de la connexion
  - Si problème à ce moment là
    - Le processus distant est a priori planté
    - Selon la configuration de la simulation du système : l'émetteur du message est informé ou non du problème de communication
- Réception d'un message
  - Bloquant par principe
    - Sur la socket TCP pour la couche communication ou dans le buffer associé pour un service
  - De nombreux threads sont donc utilisés pour gérer ces réceptions

Détail architectue



Packages

- communication
  - Couche de communication avec gestion des sockets TCP
- service
  - Accès général aux services
  - Ensemble des interfaces des services implémentés
- service.id
  - Implémentation du service d'identification
- service.broadcast
  - Implémentation (non complète) de services de diffusion basique et fiable

Package communication

- Identification d'un processus
  - ProcessIdentifier : numéro d'un processus
  - IPProcessIdentifier : extension avec couple @IP/port
    - @IP : celle de la machine sur laquelle est lancé le processus
    - Port : celui utilisé par la socket serveur TCP (déterminé automatiquement par défaut ou peut être précisé au besoin)
- TCPElement
  - Création socket serveur TCP associée à chaque processs
  - Envoi/réception de messages sur la socket TCP
- SynchronizedBuffer
  - Buffer générique avec possibilité d'attente bloquante si buffer vide
  - Utilisé pour stocker des messages reçus mais peut être utilisé dans n'importe quel autre contexte

Package communication

- ReliabilitySetting
  - Permet de positionner différents paramètres de fiabilité pour simuler un système distribué particulier et ses fautes
    - Temps de propagation minimal et maximal d'un message, taux de perte des messages, taux de crash d'un processus et information ou pas sur les erreurs
    - Se base sur des niveaux de fautes définis dans l'énumération FaultLevel
      - NONE, LOW, MEDIUM, HIGH, HIGHEST, FULL
  - Pour messages
    - Application automatique des paramètres à chaque envoi
  - Pour processus
    - Doit appeler explicitement une opération du communication element pour tenter de le faire crasher (crash = exécuter System.exit())

	NONE	LOW	MEDIUM	HIGH	HIGHEST	FULL
Temps prop. Minimal (ms)	0	40	320	1080	2560	5000
Temps prop. Maximal (ms)	0	50	800	4050	12800	31250
Taux perte message	Aucun	20%	40%	60%	80%	100%
Taux crash processus	Aucun	20%	40%	60%	80%	100%

Package communication

- Accès à la couche de communication
  - CommunicationElement abstrait qui est spécialisé par
    - ReliableCommElt (reliable = true pour ReliabilitySetting)
      - En cas d'erreur (réelle ou simulée) lors de l'envoi d'un message, la méthode utilisée lève une exception pour prévenir du problème
      - Doit donc attendre d'être sûr que le message est reçu : l'opération d'envoi de message dure le temps de propagation simulé du message
        - Emission synchrone
    - UnreliableCommElt (reliable = false pour ReliabilitySetting)
      - En cas d'erreur, la méthode ne lève pas d'exception
      - N'est donc pas informé de la perte du message ou de l'impossibilité de contacter l'autre élément (cas où il est planté)
      - Un thread est créé pour attendre le délai avant émission : opération d'envoi retourne immédiatement
        - Emission asynchrone
  - L'identificateur du processus est stocké dans le communication element

Package service

- DistributedServicesMiddleware
  - Point d'accès au middleware intégrant tous les services
    - Connexion (avec paramètres de simulation du système) et déconnexion du système
    - Récupération des références sur les différents services
- Définition d'un service : classe Service
  - Squelette de service à spécialiser pour implémenter un service
  - Association d'un type et d'un buffer de messages
  - Chaque service implémente une interface de service
    - La même interface peut être implémentée par deux services différents
    - Ex : diffusion basique et fiable offrent le même service de diffusion au niveau des opérations

Package service

- Interfaces définissant les services
  - ICommunication
    - Service permettant aux processus de communiquer directement entre eux (envoi/réception de messages)
    - Déjà implémentée par CommunicationElement
      - Mais ne peut pas utiliser directement CommunicationElement car en réception tous les messages ne sont pas destinés au processus mais peuvent être pour d'autres services
      - Utilisation d'un proxy : ProxyCommunication
  - IBroadcast
    - Services de diffusion : basique, fiable, atomique, ...
  - IIdentification
    - Récupération pour un processus de son identifiant
    - Identifiants de tous les processus présents dans le système

Package service

- Trois classes pour gérer les messages
  - Message : message reçu ou à envoyer contenant
    - ProcessIdentifier : id du processus émetteur ou récepteur
      - En pratique, utilisera systématiquement la spécialisation pour TCP/IP
    - Data : données du message sous la forme d'un objet Java quelconque (qui doit implémenter java.io.Serializable)
  - MessageType : énumération définissant les types d'usage des messages
    - IDENTIFICATION, BASIC_BROADCAST, RELIABLE_BROADCAST, ...
    - Processus directement : NONE
  - TypedMessage : un message avec précision de son type

Package service.id

- Implémentation du service d'identification
- Principes
  - Dès sa connexion au système, un processus doit recevoir un identifiant unique
  - Chaque processus connait la liste des identifiants des processus présents (modulo la détection de leur plantage)
  - Processus peuvent être lancés sur n'importe quelle machine et se connecter/déconnecter quand ils veulent
- Solution
  - Un serveur d'identification doit être lancé avant les processus
    - Gère l'unicité des identifiants
  - Communication via une socket multicast UDP du processus vers le serveur
    - Pas besoin de savoir sur quelle machine est lancée le serveur
  - Communication par des messages du serveur vers le processus via un communication element fiable sans erreur/délai ajoutés

Package service.id

- Service d'identification est un service particulier
  - Par principe, doit fonctionner sans erreur
    - La connaissance de l'identifiant d'un processus est obligatoire et celles des autres processus indispensable (modulo la détection des plantages)
    - Pas de perte de messages entre processus et serveur
    - Multicast UDP non fiable mais pas de risque de perte de message sur un réseau local
- Diffusion des identifiants des autres processus
  - A chaque connexion ou déconnexion d'un processus, le serveur envoie la liste des identifiants mise à jour à tous les processus
  - Implémentation basique d'un détecteur de panne
    - Toutes les 3 secondes, le serveur envoi un message à chacun des processus
    - Si problème de communication avec un processus, on le considère comme planté et le retire de la liste des identifiants
    - Envoi à tous les processus la liste des identifiants si elle a été modifiée

Package service.id

- Coté processus
  - Connaissance des identifiants des processus en mode pull et transparent
  - N'est pas informé explicitement de la connexion / déconnexion / plantage d'un processus mais mise à jour en tâche de fond de la liste des identifiants des processus présents
  - Principe à suivre : à chaque fois qu'on veut envoyer un message à des processus, récupérer la liste des identifiants via l'opération dédiée du service pour avoir la liste la plus à jour
- Classes du package
  - IdentificationServer : serveur d'identification
  - IdentificationService : service d'identification associé à chaque processus
  - XXXData : ensemble de types de données pour échange d'informations entre le serveur et les services sur les processus

Package service.broadcast

- Package implémentant les services de diffusion
  - BasicBroadcast : diffusion basique
    - Envoi d'un message à tous les autres processus présents
    - Simule le plantage d'un processus pendant la diffusion
  - ReliableBroadcast : diffusion fiable
    - Non implémentée, à faire en TP

Implémentation nouveau service

- Si nouveau type de service
  - Définir son interface dans le package service
  - Ajouter son type dans l'énumération service.MessageType
- Dans un package dédié au type de service
  - Créer une classe implémentant cette interface et spécialisant service.Service
  - Ajouter une opération dans service.IDistributedServices pour récupérer le point d'accès sur le service
  - Modifier la classe service.DistributedServicesMiddleware en prenant comme exemple la gestion des services existants
    - Ajouter une variable pour stocker l'instance du service
    - Associer à son type via le message dispatcher
    - Implémenter l'opération renvoyant l'instance du service
    - Méthodes à modifier : initServices() et le constructeur