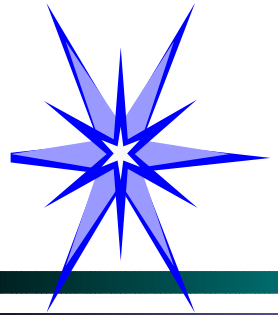


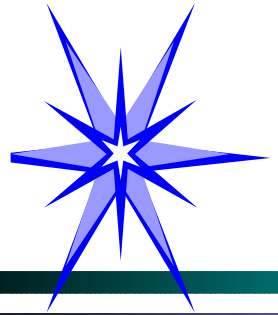
Une plate-forme dynamique pour l'appel de méthode asynchrone distant en Java

Pierre Vignéras
Équipe Systèmes et Objets Distribués
du LaBRI
LaBRI, Université Bordeaux 1, France
vigneras@labri.u-bordeaux.fr

Grappes Ile de Berder
16-18 mai 2001

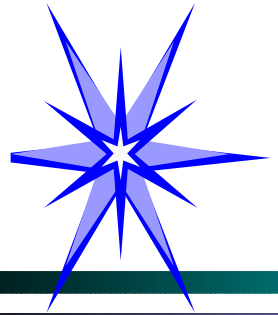
Plan

- Problématique
- Objectif dynamicité
- Objet générique : conteneur actif
- La plate-forme Jacob
  - Asynchronisme
  - Transparence
  - Semi-transparence
- Conclusion et perspectives



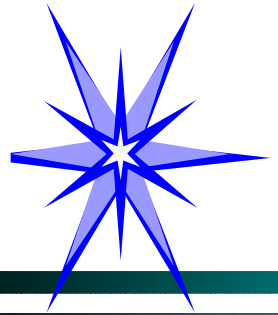
Problématique

- Les objets distants usuels requièrent
  - la déclaration d'une interface
    - IDL (CORBA), java.rmi.Remote (RMI)
  - Une compilation (besoin du code source)
    - compilateur IDL (CORBA), rmic (RMI)
- sont composés de :
    - la partie "métier" (MyObject)
    - la partie serveur (UnicastRemoteObject)
    - la partie talon (Proxy ou Stub)
    - la partie squelette (Skeleton)



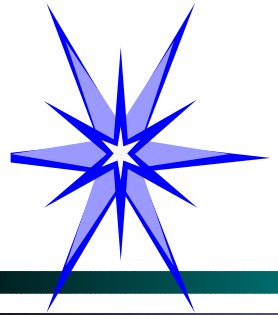
Objectif : dynamicité

- Tout objet doit pouvoir devenir distant dynamiquement et toute méthode d'un objet doit pouvoir être appelée de manière asynchrone
  - Séparation des aspects distant et asynchrone de l'implémentation de l'objet
  - Pas de déclaration ni de compilation
    - réutilisation de code
  - Pas d'héritage particulier
    - Design patterns



Objet générique : conteneur actif

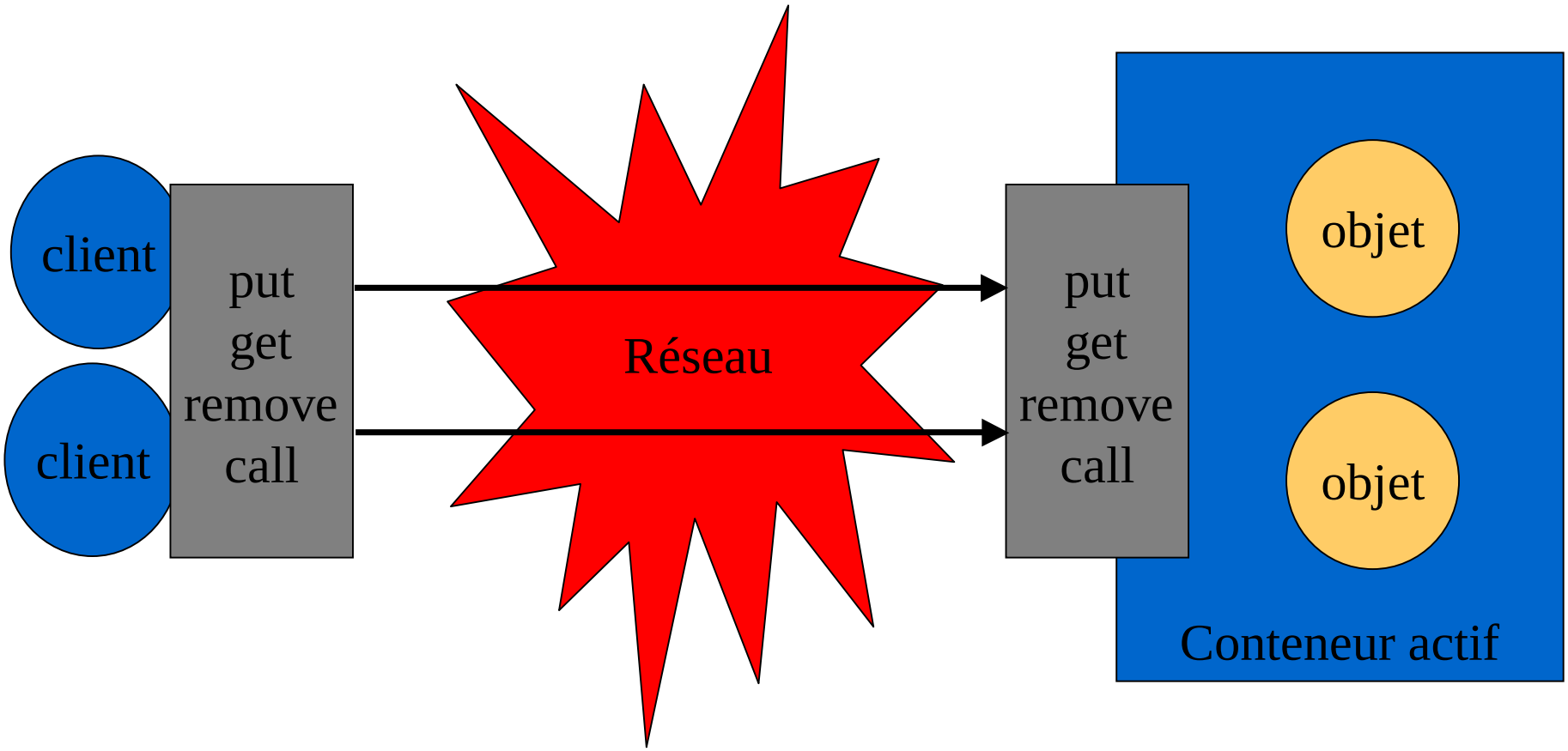
- Un conteneur
- Quatre méthodes
  - Object put(Object key, Object object)
  - Object remove(Object key)
  - Object get(Object key)
  - void call(Object key,
    String method,
    Object[] args,
    MethodResult result)
- La méthode call génère de l'activité

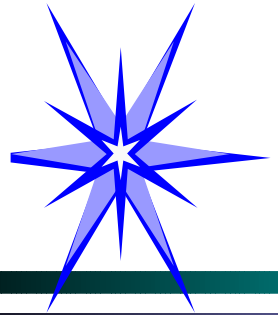


Solution : les conteneurs actifs

- Le conteneur actif est la partie serveur de tous les objets qu'il contient
- Le couple (activeContainer, key) est la référence distante d'un objet contenu dans un conteneur.
- L'utilisation de la réflexion dans le conteneur actif élimine la partie squelette.
- Rendre un objet distant revient à l'insérer dans un conteneur.

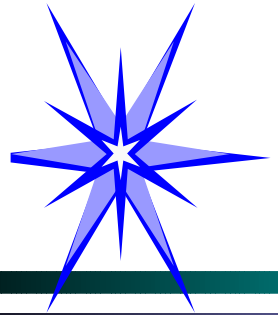
Solution : les conteneurs actif





Solution : les conteneurs actifs

- Inconvénients :
  - Le mécanisme de réflexion est lent
  - Perte du typage fort donné par le couple Stub/Compilateur
    - Utilisation des Proxy dynamiques du JDK 1.3 pour résoudre ce problème
    - Compilation d’une interface client

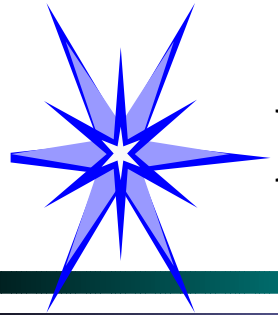


Expressivité des conteneurs actifs

- On peut exprimer une migration d'agent avec quatre primitives :
  agent.migrate(destination)
  - Agent agent = (Agent) from.get(agentID);
  - from.remove(agentID);
  - to.put(agentID, agent);
  - to.call(agentID, "onMigrationMethod", null, null);
- Formalisation en π-calcul (Mémoire de DEA)

Plan

- Problématique
- Objectif dynamicité
- Objet générique : conteneur actif
- La plate-forme Jacob
  - Asynchronisme
  - Transparence
  - Semi-transparence
- Conclusion et perspectives

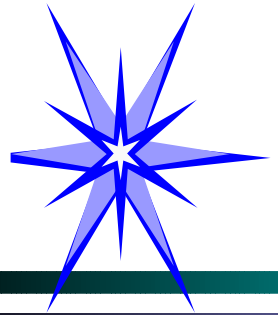


La plate-forme Jacob

- Basée sur la notion de conteneur actif
- Les conteneurs actifs sont distants
  - Tout objet peut devenir distant en étant placé dans le conteneur
- Dynamicité
- Ajout de l'asynchronisme dans la méthode call
  - parallélisme

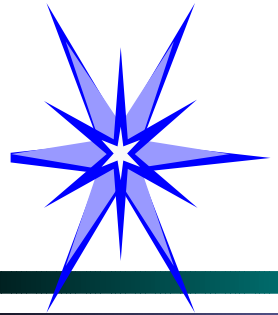
Plan

- Problématique
- Objectif dynamicité
- Objet générique : conteneur actif
- La plate-forme Jacob
  - Asynchronisme
  - Transparence
  - Semi-transparence
- Conclusion et perspectives



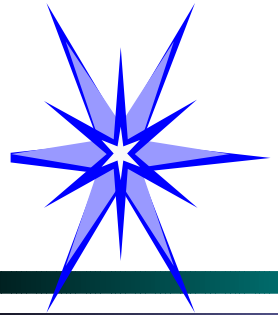
Ajout de l'asynchronisme

- void call(Object key,
  String method,
  Object[] args,
  MethodResult result)
- la méthode spécifiée de l'objet associé au couple (activeContainer, key) est exécutée par une nouvelle thread du conteneur
  - Cette thread est accessible au client
- Le résultat (exception ou retour de méthode) sera envoyé à l'objet distant MethodResult
- Asynchronisme côté serveur



Appel de méthode asynchrone distant

- L'appel précédent requiert une communication réseau
  - l'appel est bloquant jusqu'au lancement de la thread par le serveur
  - l'asynchronisme est partiel
- Le client peut utiliser une bibliothèque d'asynchronisme côté client pour l'ensemble des opérations sur les conteneurs (put, get, remove et call).
- Le client a accés aux deux threads impliquées dans son appel (thread communicante et thread d'exécution de la méthode)

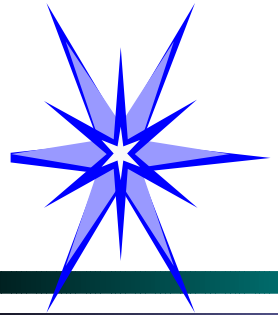


Asynchronisme et exceptions

- Problème délicat
- Deux types d'exceptions
  - Les exceptions réseau
    - gestion globale pour l'ensemble de l'application
    - Enregistrement d'un gestionnaire d'exception réseau
  - Les exceptions "métiers"
    - gestion locale par appel de méthode
    - si une exception survient durant l'appel de méthode asynchrone
      - l'appelant peut être interrompu
      - l'appelant récupère l'exception lors de la récupération du résultat

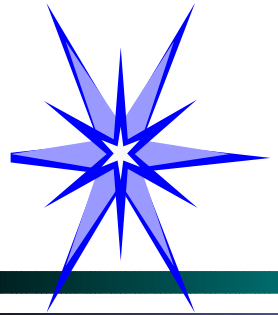
Plan

- Problématique
- Objectif dynamicité
- Objet générique : conteneur actif
- La plate-forme Jacob
  - Asynchronisme
  - Transparence
  - Semi-transparence
- Conclusion et perspectives



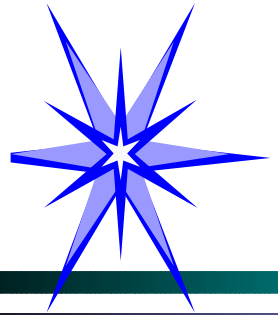
Transparence des appels distants asynchrones

- But :
  - un appel local en apparence
  - un appel asynchrone distant en réalité
- Moyen : évaluation retardée
  - la thread est bloquée à l'utilisation du résultat
  - Exemple :
    - Result result = object.method(args); // Appel asynchrone
    - computeSomething();
    - useResult(result.getValue()); // Appel bloquant !!



Transparence des appels distants asynchrones

- Par héritage
  - accès aux champs non controlés
  - mots clés 'final', 'static'
- Utilisation exclusive des interfaces
  - plus de problèmes avec les mots clés final ou static.
  - plus d'accès aux champs
    - Utilisation du mécanisme de proxy dynamique du JDK 1.3
- Limitations communes :
  - Cohérence des objets passés par copie

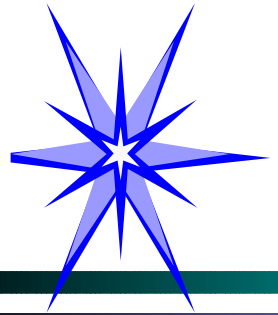


Problèmes soulevés par la transparence

- Gestion des exceptions lors d'un appel asynchrone transparent
  - Plus de try/catch
  - Rendre l'appel bloquant ne résout pas le problème des exceptions non contrôlées
- Efficacité
  - Le développeur doit garder à l'esprit l'aspect asynchrone pour obtenir des gains maximum (appel de méthode au plus tôt et utilisation du résultat au plus tard)
- Intérêt : sucre syntaxique

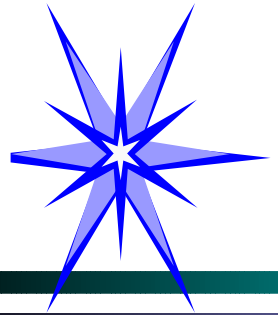
Plan

- Problématique
- Objectif dynamicité
- Objet générique : conteneur actif
- La plate-forme Jacob
  - Asynchronisme
  - Transparence
  - Semi-transparence
- Conclusion et perspectives



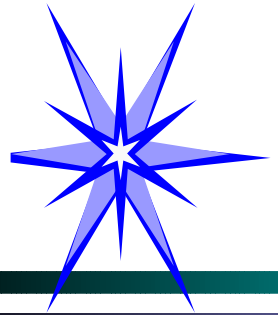
Semi-transparence

- Inconvénient de la transparence distante :
  - Passage des paramètres par copies
    - Non conforme au langage
    - Problème de cohérence des objets
  - Solutions : rester conforme au langage même en distant
    - Passage par référence
      - Remplacer o.m(p) par o.m((activeContainer, key))
        où activeContainer est le conteneur actif dans lequel réside p (automatiquement inséré s'il n'y est déjà)
        et key, et la clé de p dans activeContainer.
    - Clonage
      - Remplacer o.clone() par activeContainer.put(o);



Semi-transparence

- Inconvénients de la transparence asynchrone :
  - Prise de conscience du développeur de l'asynchronisme pour être efficace
  - Gestion des exceptions compliquée
- Solution : compilation d'une interface pour le client
  - Chaque méthode retourne un objet futur
    - Remplacer
      Result res = o.m(p);
      par
      CallTask task = o.m(p);
      try{ Result res = (Result) task.waitForResult(); }catch(Exception...



Conclusion et perspectives

- Jacob est une plate-forme opérationnelle
  - Problème du voyageur de commerce
  - Analyse spectrale distribuée de son
  - Implémentation plus efficace du protocole de communication sous-jacent
  - Distribution et parallélisation "automatique" des applications Java
- Implémentation d'objets "services" :
  - persistance, sécurité, transaction, messagerie, tolérance aux pannes
- Implémentation de la spécification MASIF (agents mobiles)
- Semi-transparence
  - asynchrone
  - distante