

*Journées « Composants adaptables »
17 et 18 octobre 2002
Grenoble*



Gestion de ressources pour composants parallèles adaptables

Luc Courtrai, Frédéric Guidec, Yves Mahéo

Equipe Orcade / Laboratoire VALORIA / Université de Bretagne-Sud (Vannes)

Email: {Prenom.Nom}@univ-ubs.fr

Web: <http://www.univ-ubs.fr/valoria/Orcade>

Contexte

- Plates-formes de déploiement pour applications réparties (~ grappes)
 - de plus en plus hétérogènes (matériel et réseau)
 - souvent non dédiées (eg stations de travail sur LANs performants → applications et utilisateurs multiples)
- Approche par composants
 - Maîtrise de la complexité
 - Réutilisation, portabilité

Composants parallèles adaptatifs

- Composant « parallèle »
 - Composant \equiv unité de déploiement
- Composant « adaptatif » (plutôt qu'adaptable)
 - Adaptation...
 - lors du déploiement
 - en cours d'exécution
 - ... en fonction de l'environnement
 - 👉 Environnement \equiv ensemble de ressources dont l'état peut être observé en cours d'exécution

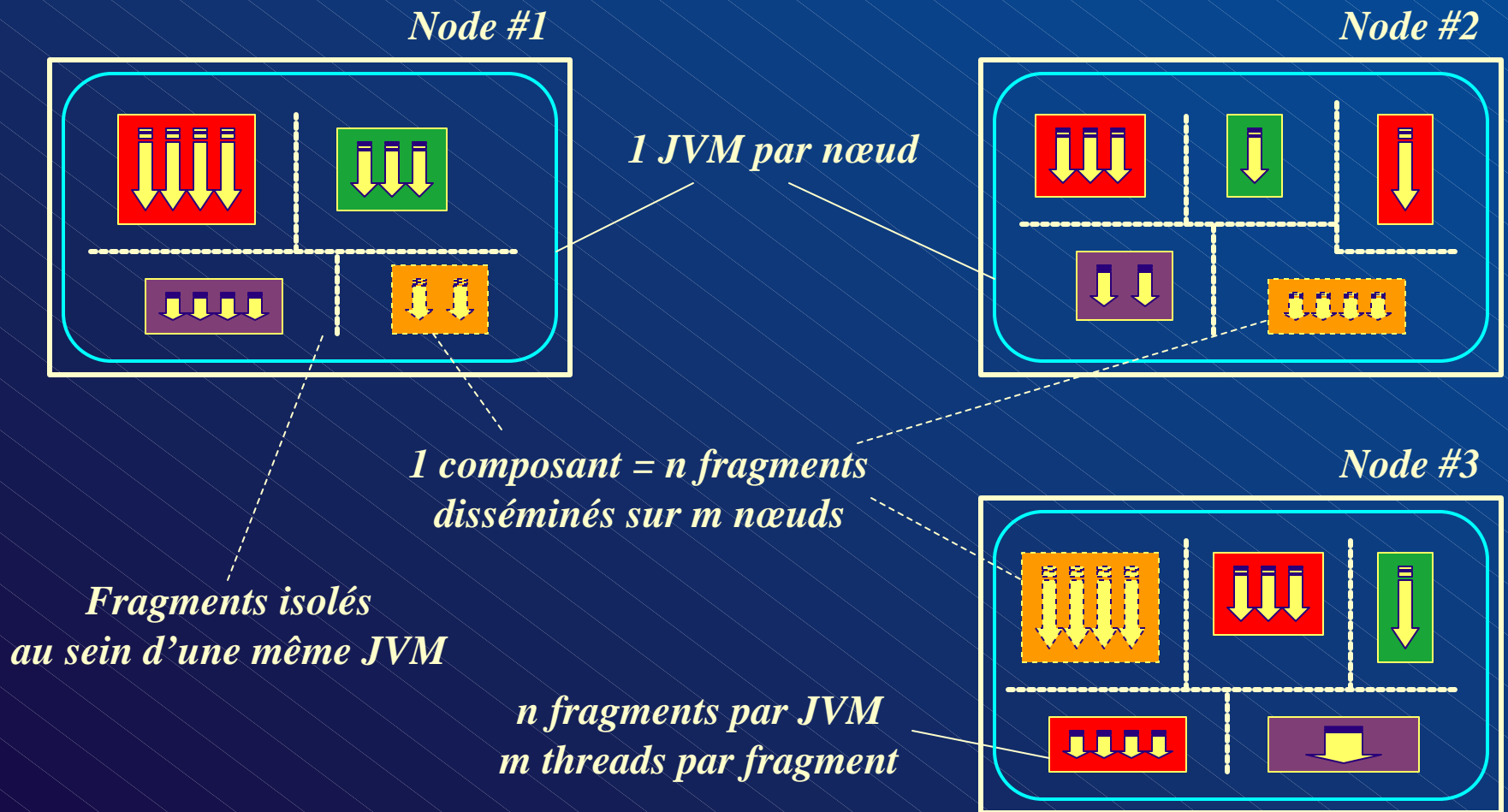
Axes du projet

- Objectifs
 - Définition d'un modèle basique de composant parallèle
 - Modélisation de l'environnement en termes de ressources utilisées (ou utilisables) par un composant
 - ↳ Définition de schémas d'observation de ces ressources
- Réalisation
 - Plate-forme de déploiement de composants parallèles adaptatifs

Composant parallèle

- Structure
 - Composant \equiv ensemble de *threads*
 - Regroupement des *threads* par fragments
 - Fragment \equiv unité de placement
 - Les *threads* d'un même fragment peuvent partager des objets
 - Les *threads* de fragments différents communiquent par les moyens habituels (*sockets*, RMI, JMS,...)
- Nommage
 - Identification en tant que ressource
 - Noms fournis par la partie métier (RMI, etc.)

Déploiement de composants sur la plate-forme Concerto



Déploiement

- Descripteur XML
- Structure
 - Découpage en termes de fragments et *threads*
- Directives de placement des fragments
 - Duplication
 - Placement sur un nœud spécifique
- Contraintes de déploiement
 - Exigences vis-à-vis de la plate-forme (eg présence d'un *RMI registry*)

Un composant parallèle → trois interfaces

- Interface « métier »

 Responsabilité du programmeur

- Interface RMI, client-serveur via TCP/UDP, etc.

- Interface « de cycle de vie »

– Déploiement, lancement, terminaison...

- Interface « ressource »

– Composant \equiv ressource (observable)

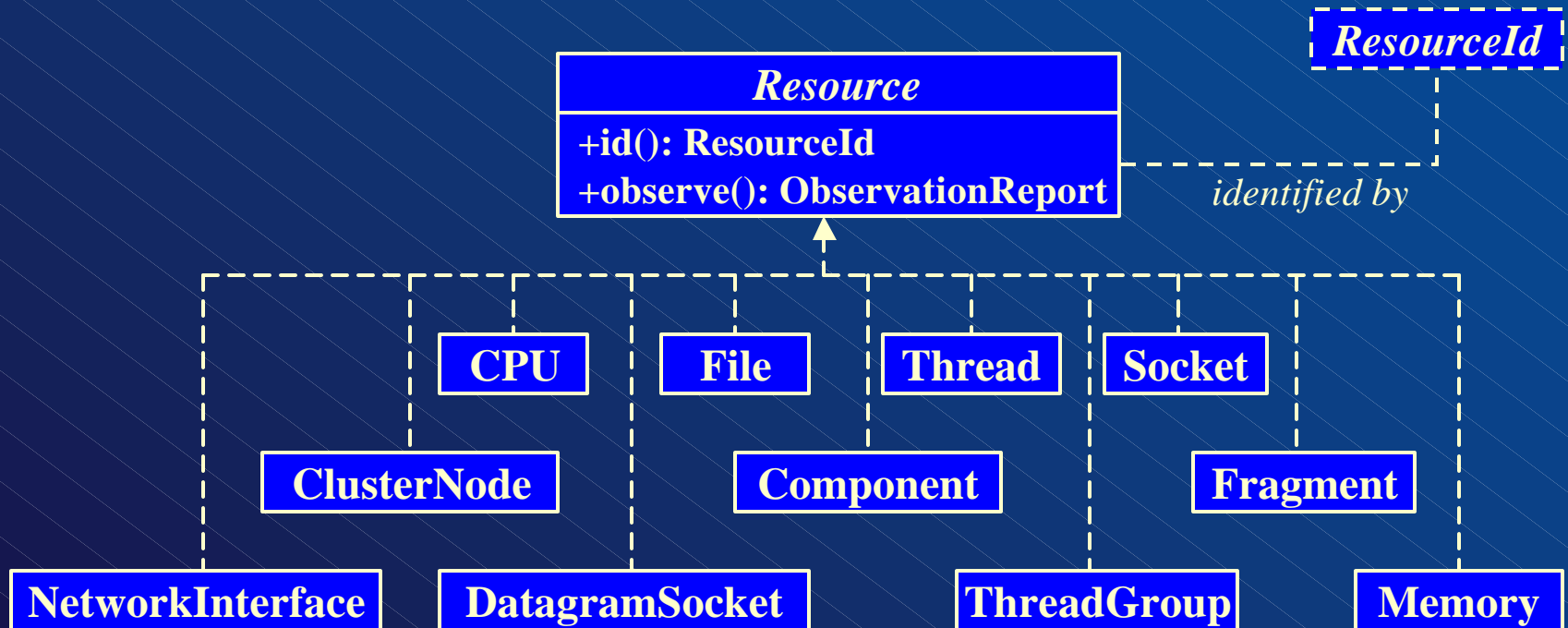
Prise en compte des ressources

- Services offerts aux composants
 - Découverte de ressources
 - Observation de l'état d'une ressource
 - Notification sur changement d'état
- Démarche
 - Modélisation et observation des ressources locales à chaque nœud
 - Prise en compte de la distribution des ressources au sein de la grappe
 - Modélisation des ressources « globales »

Modélisation des ressources

- Ressources « système »
 - Caractérisent la plate-forme matérielle (ie chaque nœud de la grappe)
 - eg CPU, mémoire, swap, interfaces réseau, disques durs
- Ressources « conceptuelles »
 - Caractérisent des ressources applicatives...
 - eg *threads*, *sockets*, répertoires, fichier
 - ... ou fournissent des services au niveau applicatif
 - eg *RMI registry*, bibliothèque de communication
 - ... ou participent au modèle de déploiement
 - eg composant, fragment

Modélisation des ressources

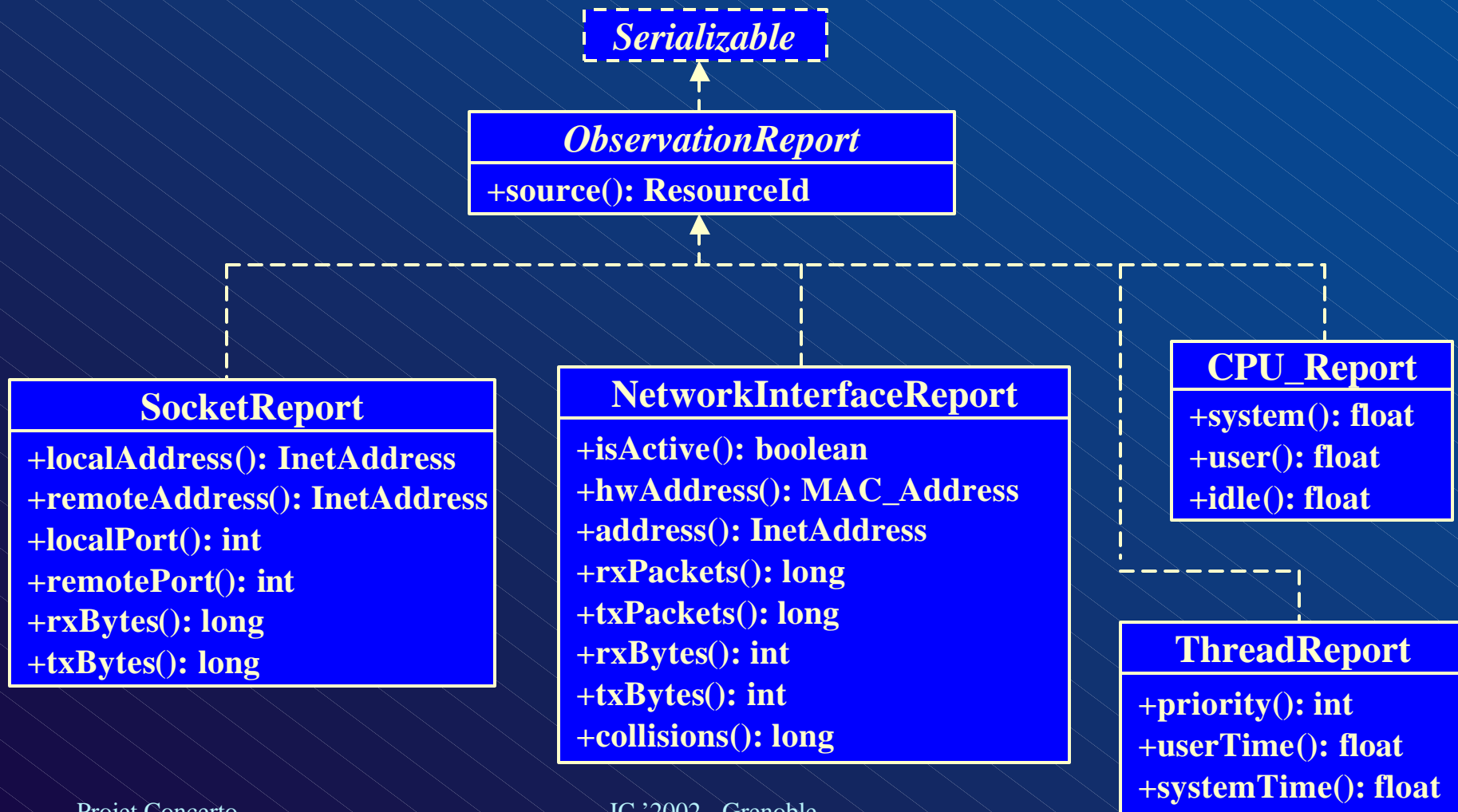


- ☞ A chaque type de ressource connu correspond une classe Java dans Concerto
- ☞ De nouveaux types de ressources peuvent être intégrés à tout instant dans le système

Perception des ressources réparties dans la plate-forme Concerto

- Toute ressource modélisée sous forme d'un objet Java dans Concerto...
 - ... porte un identifiant unique
 - ↳ On peut désigner une ressource sans ambiguïté, où qu'elle soit dans la grappe
 - ... est « observable »
 - ↳ On peut consulter l'état d'une ressource, exprimé sous la forme d'un « rapport d'observation »
 - ↳ A chaque type de ressource correspond un type de rapport spécifique

Modélisation des rapports d'observation



Le gestionnaire de ressources (*ResourceManager*)

- Une instance créée sur chaque nœud de la grappe
 - Connait toutes les ressources locales à ce nœud, et peut en observer l'état à tout instant
 - 👉 Le gestionnaire de ressources présent sur un nœud est informé de toute création ou destruction de ressource sur ce nœud
- Les diverses instances du gestionnaire de ressources coopèrent pour fournir un service uniforme au sein de la grappe
 - Identification, localisation, et consultation de l'état des ressources (qu'elles soient locales ou distantes)

Contraintes et choix de mise en œuvre

- Plate-forme de déploiement
 - Linux
 - JVM Kaffe 1.0.6 modifiée
 - *Threads* Java = *threads* natifs (→ suivi consommation CPU)
 - Instrumentation de la consommation mémoire
- Ressources modélisées en Java
 - Ressources « système »
 - Classes d'objets Java, avec code natif (JNI) pour extraire les informations de l'OS sous-jacent
 - Ressources « conceptuelles »
 - Classes standard du JDK (ou classes ad hoc) instrumentées afin de permettre l'observation des ressources considérées

Tableau de bord de la plate-forme Concerto

The screenshot shows the Concerto platform dashboard with several key areas highlighted by red boxes and arrows:

- Chargement de nouveaux composants**: Points to the top toolbar containing icons for file operations and component management.
- Visualisation du « contenu » d'un composant**: Points to the 'Contents' section of the 'modele2.jar' component, listing files like META-INF, modele2.xml, and Th1.class through Th3.class and ThreadC.class.
- Répartition des fragments d'un composant sur la grappe**: Points to the 'Components' list on the left, showing fragments like 'Frag2:wyre', 'Frag2:rousay', 'Frag1:hoy', 'Frag2:shapinsay', and 'Frag2:hoy'.
- Hôtes constituant la grappe cible**: Points to the 'Server' list at the bottom left, showing nodes 'shapinsay', 'hoy', 'rousay', and 'wyre'.
- Visualisation de l'activité sur la grappe**: Points to the 'Result' section at the bottom right, displaying the output 'resultat renvoyé par le serveur frontal..... FIN'.

Tableau de bord de la plate-forme Concerto

The screenshot displays the ConcertoTools interface with the following sections:

- File Descriptor:** Contains files `modele1.jar` and `modele2.jar`.
- Components:** A tree view showing `Component1`, `Component2`, and three fragments: `Frag3:shapinsay`, `Frag2:wyre`, and `Frag1:hoy`, each with associated resource IDs.
- Descriptor Component / Descriptor File:** Shows `Name : Component2` and a **State of threads :** section listing three unstarted threads: `Frag3:shapinsay/ResourceId(shapinsay:Resource#29')`, `Frag2:shapinsay/ResourceId(wyre:Resource#29')`, and `Frag1:shapinsay/ResourceId(hoy:Resource#90')`.
- Server:** Lists servers `shapinsay`, `hoy`, `rousay`, and `wyre`.
- Result:** Displays the message `resultat renvoyé par le serveur frontal..... FIN`.

Two red boxes with arrows highlight specific features:

- A box containing the text *Visualisation de l'état des threads d'un composant* points to the "State of threads" section.
- A box containing the text *Visualisation de l'état des ressources utilisées par un composant* points to the "Components" tree view.

Perspectives

- Enrichir notre modèle de composant parallèle
- Définir et mettre en œuvre des mécanismes de notification sur changement d'état
 - Formulation des critères
 - Détection et gestion d'événements
- Gérer les ressources « globales »
 - Capture d'état global, cohérence...
- Trouver et mettre en œuvre quelques belles applications de démonstration (!!!)

Questions ?