

# **Environnement minimal pour la construction de systèmes adaptables et configurables**

## ***Projet ARCAD***

Equipe SARDES - 26 mai 2002

# Plan

- Environnement cible
  - Philosophie
  - Base logicielle
    - Think
    - YNVM
- Axes de développement
  - Outils élémentaires de sécurité
  - Modèle de composant
  - Mécanismes de reconfiguration

# Motivations

- Développement d'une infrastructure logicielle légère pour la construction d'applications spécialisées et réparties
  - Adaptée à des environnements diversifiés
    - Serveurs
    - Grappes de PC
    - Objets mobiles, communicants
  - Déclinable pour des domaines fortement spécialisés
    - Temps-réel
    - Multimédia
    - Mobilité

# Caractéristiques de l'environnement cible

- Minimaliste
  - Aussi proche que possible du matériel
  - Pas d'abstractions inutiles
- Extensible
  - Canevas élémentaires de liaison et de composition
    - Appliqués à bas niveau
    - Exportables à haut niveau

# Philosophie suivie

- Minimalisation des abstractions fournies par les systèmes d'exploitation
  - Extraire certains services du système
- Deux approches
  - Approche micro-noyau
    - Exporter les services dans des espaces d'adressage indépendants
  - Approche exo-noyau
    - Exporter les services dans l'espace d'adressage des applications
- Notre approche : non-noyau
  - Pousser la philosophie exo-noyau à l'extrême
  - N'abstraire que les interfaces matérielles

# Base logicielle

- Notre réponse : couplage Think/YNVM
  - TThink Is Not A Kernel
    - Architecture logicielle pour la construction de noyaux de systèmes d'exploitation
    - Système minimal et configurable
  - Ynvm is Not a Virtual Machine
    - Compilateur dynamique, introspectable et reconfigurable
- Résultat : un non-noyau minimal, extensible, introspectable et dynamiquement reconfigurable

# Think : rappels

- Développé par Jean-Philippe Fassino, FT R&D
- Propriétés
  - Modulaire
    - Gestion de la complexité par décomposition en sous-systèmes
  - Flexible
    - Capacités d'évolution et d'adaptation
  - Uniforme
    - Modèle de programmation utilisé à tous les niveaux
  - Administrable
    - Possibilités d'observation, de contrôle et de gestion des ressources logicielles et matérielles

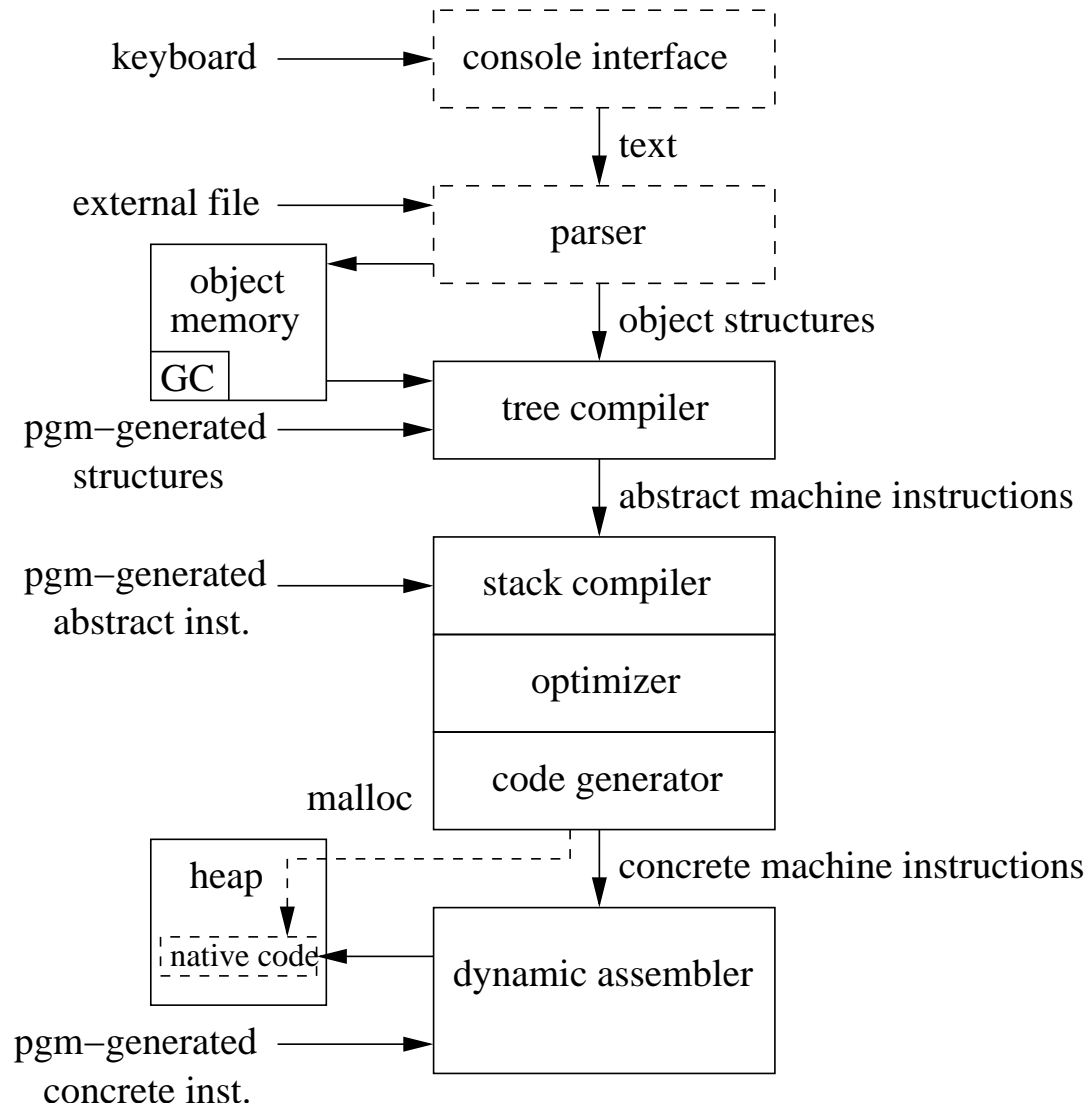
# Think : rappels

- Réification stricte des fonctionnalités du PowerPC
- Canevas logiciel de liaisons flexibles
- Bibliothèque de composants
  - Interruptions
  - Ordonnanceur
  - Mémoire
    - Modèles de mémoire plate, paginée
  - Réseau
    - Modèle de pile de protocoles
  - Pilotes de périphériques
    - Cartes réseau, pilotes IDE, etc.

# YNVM : présentation

- Développée par Ian Piumarta (LIP6/Projet SOR)
- Générateur dynamique de code natif
  - Compilation de programmes YNVM
    - Langage “à la Lisp”
  - Chargement de bibliothèques C
  - Introspection et adaptation
    - Code dynamiquement compilé
    - Machine YNVM
    - Environnement système
- Performances équivalentes à du code C optimisé, compilé statiquement

# YNVM : présentation



# Plan

- Environnement cible
  - Philosophie
  - Base logicielle
    - Think
    - YNVM
- Axes de développement
  - Outils élémentaires de sécurité
  - Modèle de composant
  - Mécanismes de reconfiguration

# Trois axes de développement

- Modèle de composant
  - Assemblage de composants
  - Structuration du système
- Mécanismes de reconfiguration
  - Évolution du système en fonction de son environnement
- Outils élémentaires de sécurité
  - Protection du système et des applications

# Outils de sécurité

- Point essentiel : découpler les politiques de sécurité des outils permettant de les mettre en oeuvre.
- Outils
  - Isolation mémoire
  - Ordonnanceur de disque équitable
  - Gestionnaire réseau sécurisé
- Canevas
  - Protection des liaisons logicielles : contrôles d'accès et d'intégrité
  - Utilisation des usines à liaisons : surcharge de la méthode de création de liaison pour ajouter des vérifications

# Modèle de composant

- Instanciation du modèle de cellules de JBS en gardant une approche minimaliste
- Définition des concepts primitifs tout en permettant des assemblages évolués
  - Fractal : trop complexe, pas applicable à bas niveau
  - Jonathan : manque d'introspection et de reconfiguration; pas de possibilité de partage de composants

# Canevas de liaisons

- Name
  - *String encode()*
  - *NamingContext getDefaultNC()*
- NamingContext
  - *Name decode(String name)*
  - *Name export(Top object, String hints)*
  - *Top resolve(Name name, String hints)*
- Binding Factory
  - *Top bind(Name name, String hints)*
  - *Top unbind(Name name, String hints)*

# Canevas de composition

- Component
  - *Name getInterface(String interface)*
  - *Name[] getInterfaces()*
- Configuration
  - *void addSubComponent(Component component)*
    - Vérifie les dépendances, alloue de la mémoire et démarre le composant
  - *void removeSubComponent(Component component)*
    - Vérifie les dépendances, arrête le composant et le supprime de la mémoire

# Canevas de composition

- Configuration (suite)
  - *void replaceSubComponent(Component component)*
    - Remplace un composant par un autre en préservant l'état
- Life cycle
  - *State getState()*
    - Capture l'état d'un composant
  - *void start()*
    - Démarre l'exécution d'un composant
  - *void stop()*
    - Arrête l'exécution d'un composant

# Mécanismes de reconfiguration

- Motivations
  - Réponse aux évolutions de l'environnement
  - Évolution d'un système sans interruption de services
- Objectif : permettre à l'exécution
  - le chargement / déchargement de composants
  - le remplacement de composants
  - la reconfiguration des liaisons
- En maintenant l'état global de l'application
- En minimisant les répercussions sur les composants non reconfigurés
- En limitant le surcoût aux opérations de reconfiguration

# Mécanismes de reconfiguration

- Mode opératoire
  - Charger un composant
    - Le copier en mémoire et le lier vers d'autres composants
  - Décharger un composant
    - Le supprimer de la mémoire
  - Remplacer un composant
    - Sauvegarder son état et le supprimer de la mémoire
    - Copier le nouveau composant en mémoire et restaurer l'état précédent

# Mécanismes de reconfiguration

- Problèmes soulevés
  - Références, liaisons et données internes
    - État de l'instance
  - Processus en cours d'exécution
    - État d'exécution du composant

=> Capture de l'état

# Capture de l'état

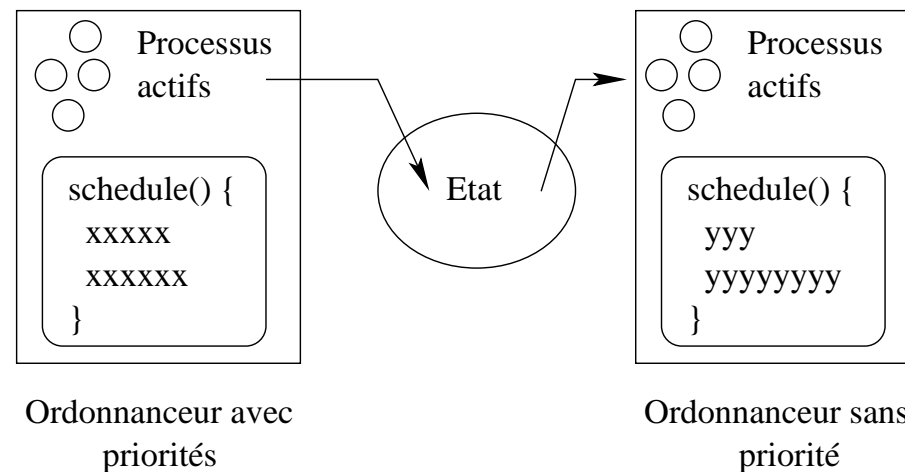
- État d'un composant = ensemble de ses connaissances sur l'environnement
  - Ensemble des variables d'instance
    - Variables globales
    - Variables locales aux méthodes
  - État de contrôle
    - Pile d'exécution
    - Registres processeur
    - Informations de gestion du processus
      - Priorité, status...

# Capture de l'état

- Ensemble des variables d'instance
  - Utiliser les mécanismes d'introspection de la YNVM
    - Accéder à la valeur des symboles d'un composant
- État de contrôle
  - Se fixer un modèle de processus
  - Définir un MOP d'accès à l'état d'un processus
  - Définir une abstraction de l'état d'un processus
  - Connaître les processus parcourant un composant à un instant donné

# Illustration

- Remplacement d'un ordonnanceur avec priorités par un ordonnanceur sans priorité
  - Sauvegarde de l'état de l'ancien composant (sauvegarde de la liste des processus actifs)
  - Chargement du nouveau composant et restauration de l'état sauvegardé



# État des lieux

- Bibliothèque de capture de l'état d'un processus donné
  - Capture, sauvegarde et restauration de processus
- Outils de capture de l'état d'instance d'un composant donné
- Bénéfices immédiats
  - Tolérance aux fautes
  - Service de persistance
  - Mécanisme de mobilité de composants

# Perspectives

- Hiérarchiser le modèle de reconfiguration
  - Caractériser l'état d'un composite
- Ajouter des contraintes d'intégrité dans le modèle de composant
- Développer un modèle de gestion de ressources reconfigurable