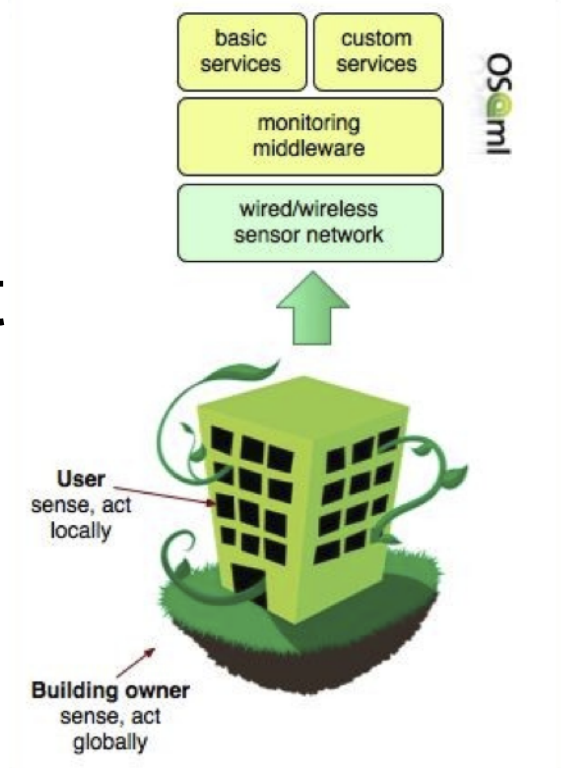


Approche orienté services pour la conception de noeuds de capteurs

François Revol
LCIS – équipe CTSYS
LIG – équipe ADELE
Francois.Revol@imag.fr

Contexte

- Projet européen OSAmI-Commons
 - Open Source Ambient Intelligence Commons
- OSAmI-FR :
Efficacité énergétique du bâtiment



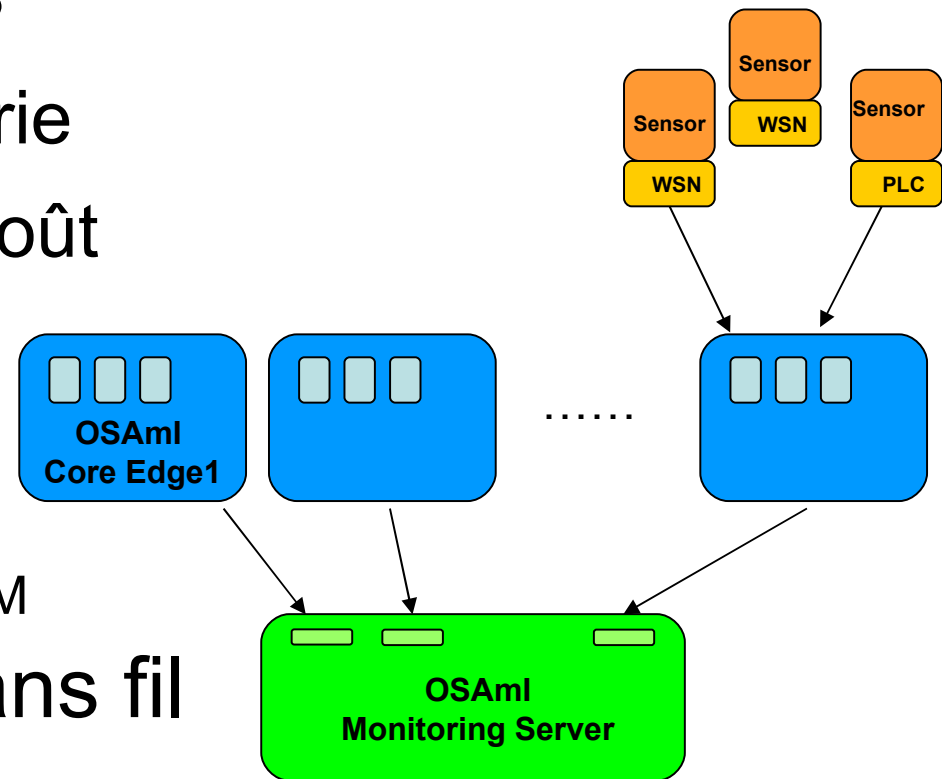
Architecture réseau/middleware

- Nœuds de capteurs
 - Alimentés par batterie
 - μ contrôleur faible coût

- Atmega128
 - 8bit, 1kB RAM
- MSP430
 - 16bit cpu, 10kB RAM

- Liens faible débit sans fil ou CPL

- Passerelles connectées à un back-office

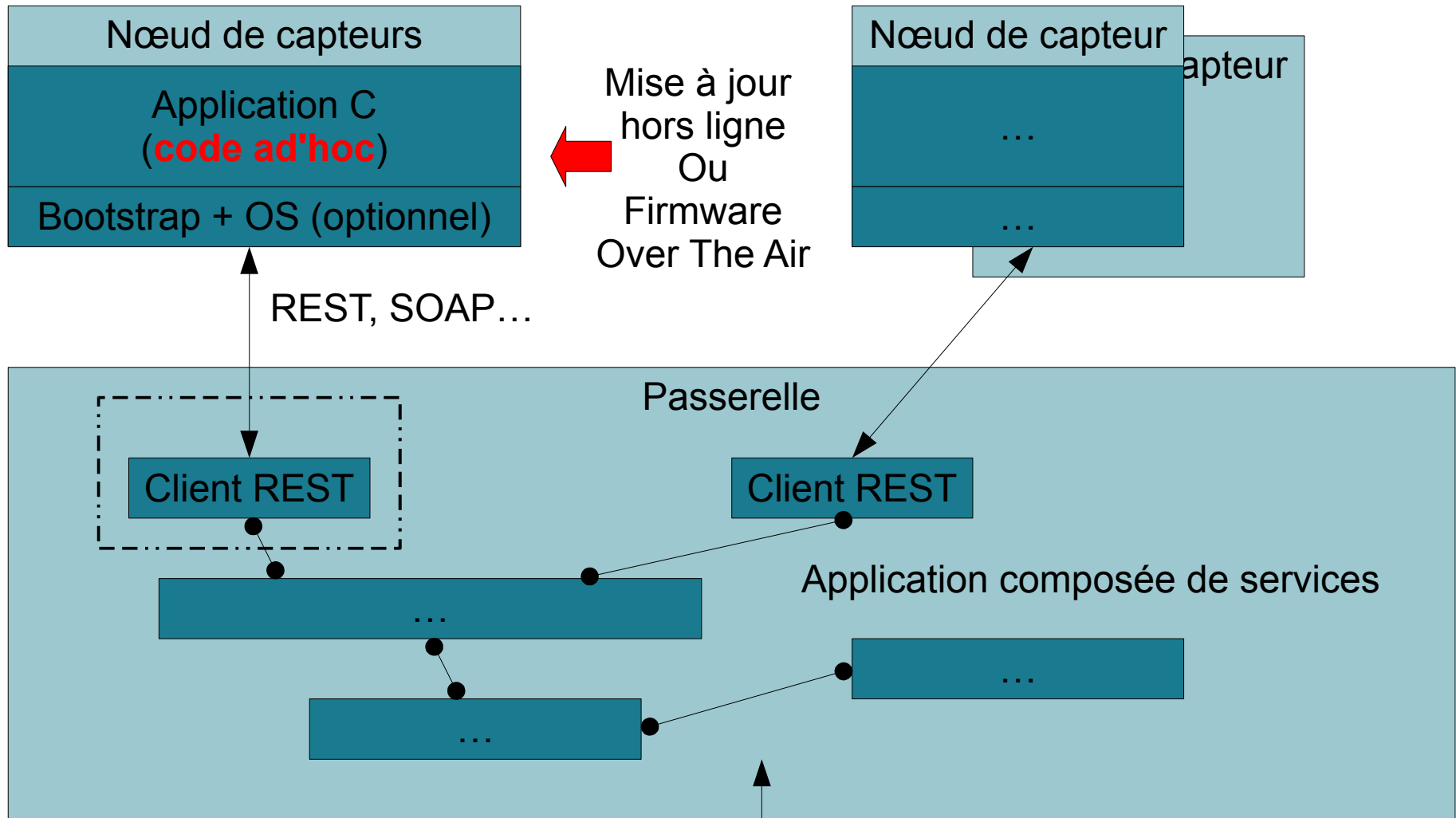


Plateforme pour nœuds de capteurs

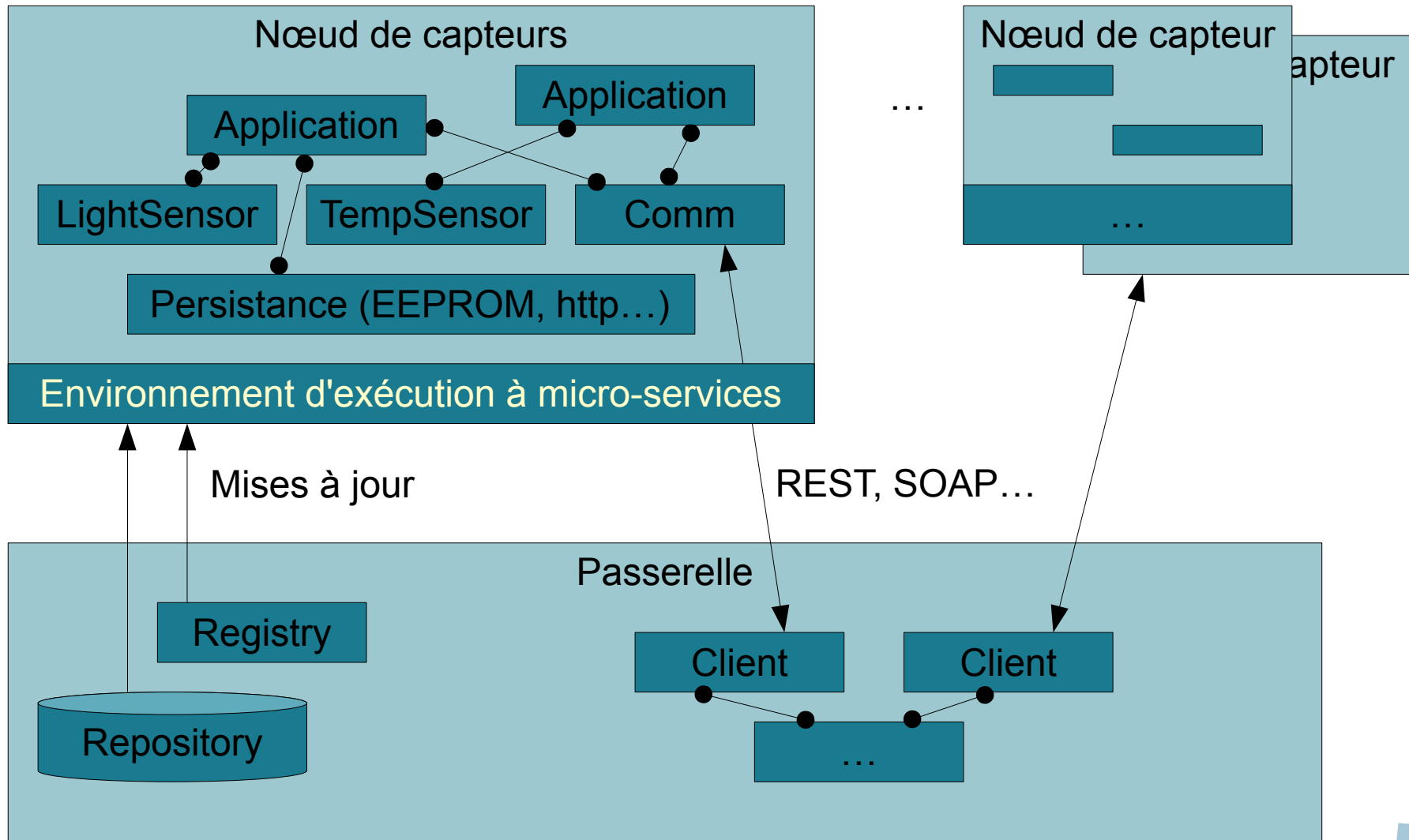
- Contraintes de coût
 - 8/16 bits (32 ?)
 - Taille de RAM déterminante (<16ko)
- Contraintes d'interopérabilité
 - 6LoWPAN (IPv6 sur IEEE 802.15.4)
 - Protocole applicatif orienté Web (Web of Things)
- Contraintes de maintenance
 - Faible consommation
 - Mise à jour à distance, portabilité

Architecture actuelle

Sensor as a service



Architecture visée : *Services on Sensor*



Bénéfices attendus

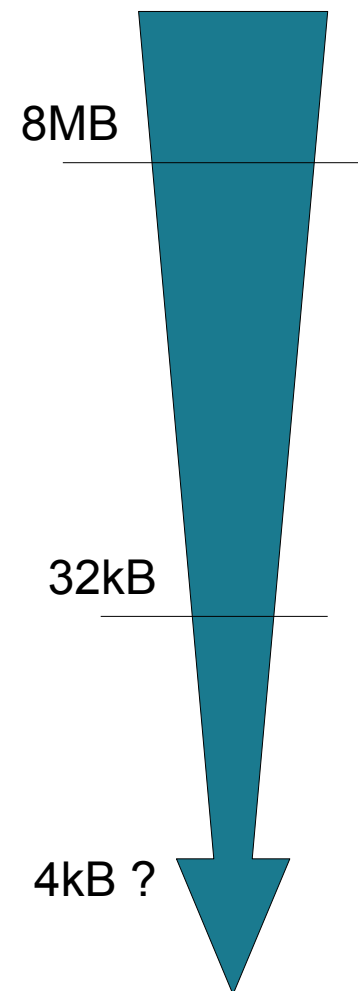
- Mise à jour partielle en ligne
 - Interruption de service minimale
- Segmentation du développement
- Niveau d'abstraction plus élevé
- Interopérabilité
- Réutilisation d'outils existants
- Facilitation du réemploi vers les cibles futures

OSGi™

- Framework Java pour la composition dynamiques de services
- Spécifie
 - Le déploiement
 - MAJ partielles
 - Le cycle de vie
 - La composition flexible des services
- Différentes implémentations
 - Apache Felix, Equinox (Eclipse, ...), Concierge (R3)

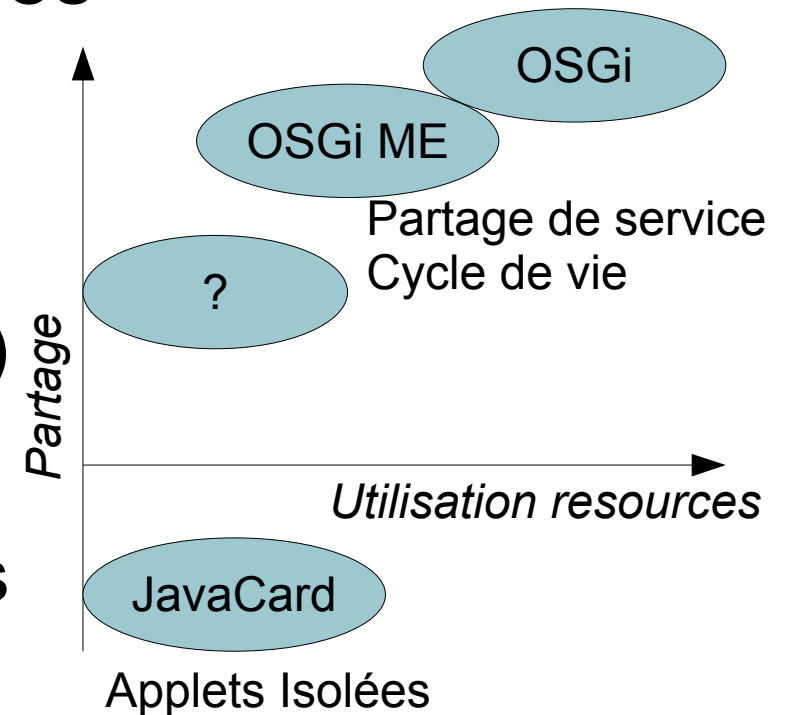
OSGi vers l'embarqué

- **Concierge** (R3), ProSyst mBS
 - Optimisé pour l'embarqué 32bit
 - Nécessite ClassLoader, HashMap, ...
- **Jadabs-CLDC** sur MIDlets (2004)
 - Incomplet
- **OSGi ME** (RFP 126, 2009) : CLDC
- Cible plus petite ?



Verrous technologiques

- Consommation des ressources
 - Mémoire, énergie
- Mise à jour
 - Compatibilité binaire (.class ?)
 - Chargement dynamique
 - Réseaux bas débit non fiables
- Isolation, Sécurité
 - JavaCard = isolation stricte
 - Certification des sources



OS pour le très embarqué

- Très nombreux
 - Généralistes
 - Spécialisés
- SOS, MANTIS OS...
- TinyOS
 - Architecture à composants (nesC)
- Smews [Duq09]
 - Spécifique (httpd)
- Contiki
 - Écrit en C
 - Protothreads
 - Chargement ELF dynamique [Dun06]
 - uIPv6 (Certifié Cisco)
 - Contiki Virtual Machine
 - Abandonnée

JVM pour le très embarqué

- **JavaCard 2.0**
 - Isolation forte car cible les cartes à puce
 - Chargement de CAPlets concurrents hors-ligne
- **JavaCard 3.0**
 - Réseau, web servlets, mais CLDC, 32 bits, **24ko**
- Nombreuses (J)VM issues de la recherche
 - **LeJOS (TinyVM), JITS, SimpleRTJ, TakaTuka, ...**
 - Chacune ayant une API spécifique d'accès aux capteurs
 - Tentatives de chargement dynamique de bytecode

Darjeeling

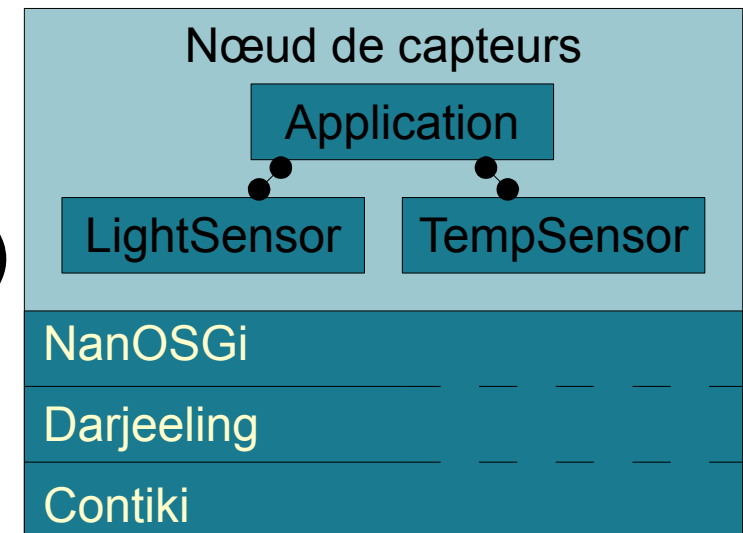
- Cibles : MSP430, Atmega128
- OS : Contiki, TinyOS, FOS
- Sous-ensemble de l'API Java
- Threads natives légères (pile = liste chaînée)
- Chargement dynamique (Infusions)
 - Édition de lien avant déploiement
 - Similaire au CAP JavaCard
 - Ordinaux (gain de place sur les noms : 50% à 80%)
 - Contraintes (ClassLoader, réflexion, introspection)

Contributions de Darjeeling [Bro09]

- Architecture 16 bits
 - Bytecodes optimisés pour les calculs 8 et 16 bits
- Garbage Collector fonctionnel
 - Absent de LeJOS, en option sur JavaCard 2.0
- Threads (Absent de JavaCard 2.0)
- java.lang.String
 - Utile pour REST/http, absent de JavaCard 2.0
- Testé avec 4ko de RAM
- Collection Tree Protocol (multithreadé)

Prototype actuel (en cours)

- Première approche sur
 - Contiki (ou natif pour le debug)
 - Darjeeling
 - Ajout du type Class (**contribué**)
- NanOSGi
 - Sous-ensemble d'OSGi compatible source
 - Implémenté comme une bibliothèque (infusion)
 - Adaptations de l'infuser de Darjeeling
 - La passerelle stocke les bundles pour les nœuds



Problèmes rencontrés

- L'API OSGi contraint l'implémentation
 - Réflexion : utilisation du type Class
 - `bc.registerService(FooServIntf.class.getName(), ...)`
 - Implique l'ajout du type Class dans Darjeeling
 - Impose ClassLoader comme mécanisme d'isolation
 - On choisit d'ignorer cette partie
- OSGi permettent des libertés d'importation
 - On impose des dépendances strictes
- Pas d'isolation par la VM
 - On se limite aux sources certifiées

Perspectives

- Intégration plus forte OS + VM + framework ?
- Pertinence du choix du modèle de composants
- Minimisation du code natif

Références

- Smews
 - [Duq09] S. Duquennoy, G. Grimaud & al. : Serving embedded content via Web applications: model, design and experimentation
 - Thèse de S. Duquennoy : Smews : un système d'exploitation dédié au support d'applications Web en environnement contraint

Références

- [Dun06] A. Dunkels & al. Run-Time Dynamic Linking for Reprogramming Wireless Sensor Networks (SenSys'06)
- [Bro09] N. Brouwers & al. Darjeeling, a feature-rich vm for the resource poor. (SenSys '09)
 - <http://darjeeling.sourceforge.net/>
- JavaCard 3.0 whitepaper
 - http://java.sun.com/javacard/3.0/javacard3_whitepaper.pdf

Références

- Jadabs-CLDC (IKS-ETHZ)
 - <http://jadabs.berlios.de/jadabs-cldc/>
- RFP 126 : OSGi ME: An OSGi Profile for Embedded Devices