OpenCard Framework (OCF)

Didier DONSEZ
Université Joseph Fourier – Grenoble 1
Polytech'Grenoble – LIG/ADELE
Didier.Donsez@imag.fr

Motivations

- OCF : Open Card Framework
  - Consortium initié par IBM
    - Bull, Dallas Semiconductors, First Access, Gemplus, International Business Machines Corp., Network Computer Inc., Schlumberger, SCM Microsystems, Sun Microsystems, UbiQ et Visa International
  - Définition d'un framework standard d'accès à des cartes et des lecteurs depuis un environnement Java
- 2 notions
  - CardTerminal
    - Drivers Terminal fourni par le fabricant du lecteur
  - CardService
    - Interface (de haut niveau) pour dialoguer avec une carte d'un type donnée
      - Fournit par l'émetteur de la carte ou le développeur de l'applet
      - Masque l'encodage et le décodage des APDU

Différents métiers de l'industrie de la carte

Le développeur d'applications ou d'applets

- Construit une application ou une applet dont le comportement est piloté par l'insertion ou le retrait de cartes des lecteurs du terminal de l'usager final. Utilise les CardServices pour s'abstraire de l'encodage/décodage des échanges avec la carte

L'émetteur de la carte

- Fournit le CardService qui permet de gérer le cycle de vie d'une carte multi-applicative (comme la Javacard) permettant l'ajout ou le retrait de services de prestataires.

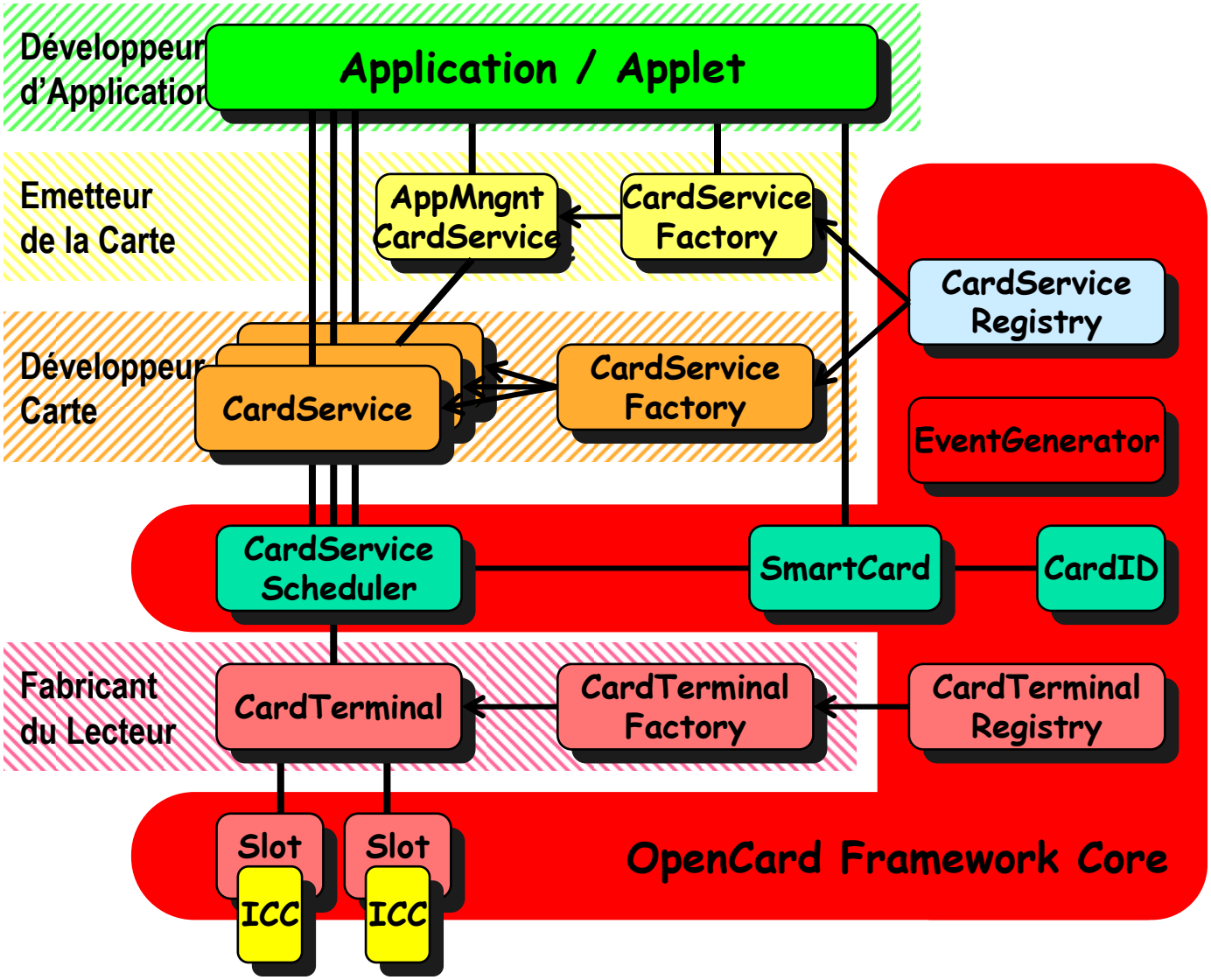
Les prestataires des services installés dans la carte

- Fournissent chacun un CardService offrant une interface de programmation de haut niveau avec le service installé. Le CardService est chargé de l'encodage et du décodage des échanges APDU avec la carte via le pilote du lecteur dans lequel elle est insérée.

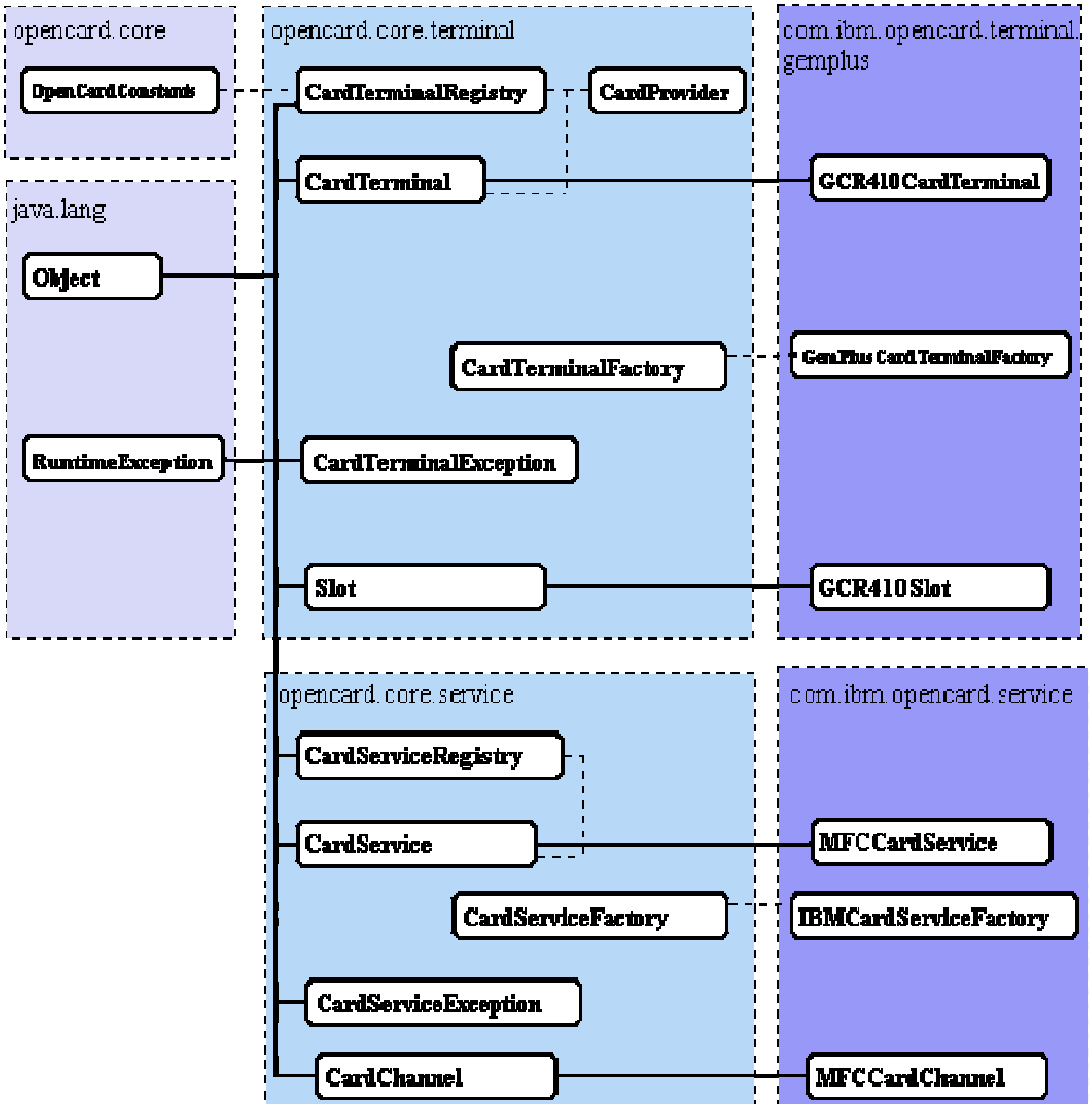
Les fabricants des lecteurs de carte

- Fournissent les CardTerminal qui sont les pilotes Java des lecteurs de carte installés sur le terminal. Le lecteur peut comporter plusieurs fentes (slot), un périphérique de saisie (mot de passe, empreinte digitale, …) et un écran. Les pilotes peuvent utilisés les API natives, l'API JavaComm d'accès au port série ou les pilotes PC/SC et doivent de préférence supporter une variété assez large de systèmes d'exploitation.

Architecture OCF



Hiérarchie de classes



CardTerminal

- CardTerminal
  - Drivers Terminal fourni par le fabricant du lecteur
    - Lecteurs cartes contact ou sans contact ou custom (eg iButton, …)
- Plusieurs possibilités d'implémentation
  - Driver natif accessible depuis JNI (.so, .dll)
  - Driver Java utilisant l'API javax.comm/gnu.rxtx
    - Port série et parallèle
  - Passerelle vers les Drivers PC/SC
    - (ou MUSCLE pour Linux/MacOSX)
- CardTerminalFactory
  - Fabrique qui enregistre 1 ou plusieurs CardService auprès du framework
    - CardTerminalRegistry.add(CartTerminalFactory factory)

CardService

- Interface (de haut niveau) pour dialoguer avec une carte d'un type donnée
  - Fournit par l'émetteur de la carte ou le développeur de l'applet carte
  - Masque l'encodage et le décodage des APDU
    - Les erreurs (SW!=0x9000) peuvent être convertis en exception
- CardServiceFactory
  - Fabrique qui enregistre 1 ou plusieurs CardService auprès du framework
    - CardServiceRegistry.add(CardServiceFactory factory)

Exemple de CardService (i)

```
public class PurseCardService extends CardService {
    /**  APDU to select the root of the card file system */
    private final byte selectRoot[] = {
      (byte) 0xA0, (byte) 0xA4, (byte) 0x00, (byte) 0x00, (byte) 0x02, (byte) 0x3F, (byte) 0x00 };
    /** APDU to select the Purse applet's AID */
    private final byte selectApp[] = {
                    (byte) 0xA0, (byte) 0xA4, (byte) 0x04, (byte) 0x00,
                    (byte) 0x07, (byte) 0x00, (byte) 0x00, (byte) 0x00,
                    (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x07
                    };
    boolean appletSelected;
    public PurseCardService() { super(); }
    protected void initialize(CardServiceScheduler scheduler, SmartCard card,
                                  boolean blocking) throws CardServiceException {
      super.initialize(scheduler, card, blocking);
      appletSelected = false;
    }
```

Exemple de CardService (ii)

```
public short debit(short montant)
    throws CardServiceException, CardTerminalException, PMEException {
 if (!appletSelected) { selectApplet();}
 if(montant<0) // vérifie les paramêtres
     throws new PMEException(response.sw())
  try {
    allocateCardChannel(); // alloue un CardChannel
    // encode la commande
    byte[] buf= new byte[]{(byte)0x80, (byte)0x02, (byte)(montant / 0x100), (byte)(montant && 0xFF)
    CommandAPDU  command  = new CommandAPDU(buf);
    // envoie la commande à la carte et reçoit la réponse
    ResponseAPDU response = getCardChannel().sendCommandAPDU(command);
  } finally { releaseCardChannel(); // free the card channel
  }
  // decode la réponse
  if(!(response.sw() & 0xFFFF)==0x9000))
     throws new PMEException(response.sw())
 byte[] data=response.data();
 return data[0]*0x100+data[1];
}
```

CardServices disponibles

- OpenCard et IBM
  - classe MFCFileAccess
    - pour la manipulation des fichiers ISO7816-4
  - classe BasicDatabase
    - pour la manipulation de cartes CQL ISO7816-7,
  - classe MFCSignatureService
    - pour les cartes cryptographiques,
  - classe BasicEMVAppletAccess
    - pour lister les applets d'une carte multi-application au standard EMV et selectionner une applet particuliere.
  - interface CHVCardService
    - qui définit un contrat à implementer pour les cartes requiérant une identification du porteur par mot de passé (Card Holder Verification ou CHV),
  - classe PassThruCardService
    - Envoie et réception d'APDU
- Visa
  - Classe VOPCardService

L'attente avec waitForCard()

```
SmartCard.start();
CardRequest cr = new CardRequest(
        CardRequest.ANYCARD,
        null,
        epurse.opencard.EpurseCardService.class
);
System.out.println("Insert a purse card ...");
SmartCard sc = SmartCard.waitForCard(cr); // wait for card supporting EpurseCardService
EpurseCardService ssp = (EpurseCardService) // instantiate card service
  sc.getCardService(EpurseCardService.class, true);
int balance=ssp.getBalance();
System.out.println("Balance:"+balance);
System.out.println("Verify PIN 1234");
byte pin[] = new byte[] {(byte)'1',(byte)'2',(byte)'3',(byte)'4'};
boolean pinValided = ssp.verifyPIN(pin);
balance = ssp.debit(100);
sc.close();
SmartCard.shutdown();
```

Notification d'événements (i)

```
public class TestCTListener implements CTListener {
  static Object monitor=new Object();
  public static void main (String [] args) {
    int cpt=Integer.parseInt(args[0]);
    try {
      SmartCard.start ();
      EventGenerator.getGenerator().addCTListener (new TestCTListener() );
      while(cpt-->0){
        synchronized (monitor) {
          System.out.println ("Veuillez insérer ou retirer une carte !\n");
          monitor.wait();
        }
      }
      SmartCard.shutdown ();
    }
    catch (Exception e) { System.out.println (e.getMessage () ); }
  }
```

Notification d'événements (i)

```
public void cardInserted (CardTerminalEvent ctEvent) {
   CardTerminal  ct=ctEvent.getCardTerminal();
   int          slotid=ctEvent.getSlotID();
   String       ctname=ct.getName();
   try {
     CardID       cardid=ct.getCardID(slotid);
     byte[]       atr=cardid.getATR();
     String       atrhex=HexString.hexify(atr);
     System.out.println("La carte "+ atrhex +" est insérée "
                    +"dans le slot "+slotid+" du lecteur "+ctname);
   } catch(CardTerminalException e) {
     System.out.println("Une carte a été insérée "
                    +"dans le slot "+slotid+" du lecteur "+ctname);
   }
   synchronized (monitor) {
     monitor.notifyAll();
   }
 }
```

Notification d'événements (iii)

```
public void cardRemoved (CardTerminalEvent ctEvent) {

   CardTerminal  ct=ctEvent.getCardTerminal();
   int          slotid=ctEvent.getSlotID();
   String       ctname=ct.getName();

   System.out.println("Une carte a été retiré "
                    +"du slot "+slotid+" du lecteur "+ctname);
   synchronized (monitor) {
     monitor.notifyAll();
   }
  }
}
```

Enregistrement des CardTerminal et CardService

- Les fabriques (factory) doivent enregistrer 1 ou plusieurs services auprès des registres du framework

- A l'initialisation du framwork
  - SmartCard.start()
  - Fichiers de propriétés opencard.properties
- À l'exécution
  - CardServiceRegistry
  - CardTerminalRegistry

A l'Installation

- Fichiers de propriétés opencard.properties
  - Par défaut, chargé au démarrage par opencard.opt.util.OpenCardPropertyFileLoader
    - java.home/jre/lib, répertoire user.home
  - Peut être remplacé par une classe custom
    - Implémentant l'interface opencard.core.util.OpenCardConfigurationProvider
    - Positionné par la propriété OpenCard.loaderClassName
- Propriétés
  - OpenCard.terminals liste des CardTerminalFactory
  - OpenCard.services liste des CardServiceFactory
- Attention
  - si plusieurs versions de JDK ou du JRE seul (1.1, 1.2, 1.3, 1.4, pJava, …) installées

personnalisée des propriétés OpenCard

```
package ocf.config;
// A utiliser avec
//java -DOpenCard.loaderClassName=ocf.config.PropertiesConfigurer MyOCFAppli
public class PropertiesConfigurer
   implements opencard.core.util.OpenCardConfigurationProvider
{
  private static final String SEP = " ";
  private static String[][] props={
   { "OpenCard.services", "epurse.opencard.EpurseCardServiceFactory"
     + SEP + "etravel.opencard.EtravelCardServiceFactory" },
   { "OpenCard.terminals",
      "com.gemplus.opencard.terminal.GemplusCardTerminalFactory"
     + "|MonLecteurDeCarte|GCR410|COM1" /* paramètres pour la fabrique */ },
   { "OpenCard.trace", "opencard:5" }
  };
  public void loadProperties() { // seule méthode à implémenter
   for(int i=0;i<props.length;i+=1){
     System.setProperty(props[i][0],props[i][1]);
   }}}
```

Enregistrement d'un CardService en cours d'exécution

```
SmartCard.start(); // le framework est démarré
                   // et des CardServicesFactory ont été enregistrés
try {
  CardServiceRegistry serviceRegistry= CardServiceRegistry.getRegistry();
  Class factoryClass = Class.forName("epurse.opencard.EpurseCardServiceFactory");
  serviceRegistry.add((CardServiceFactory)factoryClass.newInstance());
} catch (ClassNotFoundException cnfe) {
  System.out.println("<configureServiceRegistry>"+cnfe.getMessage());
} catch (InstantiationException ie) {
  System.out.println("<configureServiceRegistry>"+ie.getMessage());
} catch (IllegalAccessException iae) {
  System.out.println("<configureServiceRegistry>"+iae.getMessage());
}
…
EpurseCardService ecs =
     (EpurseCardService)sc.getCardService(EpurseCardService.class,true);
```

OCF et Applet

- Rappel
  - Le modèle de sécurité de Java empêche une applet non signée d'accéder aux ressources de la machine tel un fichier (opencard.properties) ou un lecteur de carte.
- Applet signée
  - Signer l'archive Java (.jar) de l'applet
    - avec un certificat auto-signé ou contresigné par une autorité de certification., javakey (JDK1.1), keytool (JDK1.2 et +), jarsigner
  - Nouvelles permissions dans java.policy du navigateur

eOCF : Embedded OCF

- version allégée de l'OpenCard Framework
  - Cible : l'informatique nomade et embarqué (J2ME)
    - contraintes mémoire et CPU des terminaux nomades ou légers tels que les terminaux de paiement, les décodeurs de télévision interactive, téléphones portables …
  - reste néanmoins au maximum compatible avec les développements déjà effectués avec OCF (CardTerminal et CardService).
- Différences
  - nombre de classes, interfaces et exceptions réduit à 34
  - eOCF : 25 Ko sans CardTerminal concret.
  - un seul lecteur sur le terminal avec éventuellement plusieurs Slots
  - Pas de CardTerminalRegistry : un singleton CardTerminal
  - Enregistrement des CardServices avec CardServiceRegistry
  - Pas de SmartCard.waitForCard() : utilisation des CTListener

JC-RMI (JavaCard 2.2)

- Fournit un proxy dynamique (OCFCardAccessor) vers une applet JavaCard 2.2
  - CardService/CardServiceFactory
  - Applet implémentant une interface étendant java.rmi.Remote
  - jcrmi : Générateur de Souche-Talon masquant le encodage/decodage en APDU des invocations de méthodes de l'interface

OCF : Conclusion

- Objectifs
  - Tenter de définir un consensus pour une API d'accès aux lecteurs et aux cartes régi par un consortium autour d'IBM
  - Approche orientée objet : extensibilité, réutilisabilité, *etc.*
- Commentaires
  - Standard ouvert qui commence à susciter de l'intérêt (*vs* PC/SC)
  - Complémentaire à Java Card pour le développement des applications clientes
  - Problème d'adaptation aux petits environnements (*e.g.*, terminaux de paiement)

Bibliographie

- Uwe Hansmann, Martin S. Nicklous, Thomas Schäck, Frank Seliger, Smart Card Application Development Using Java, Ed Springer, 2000, ISBN: 3-540-65829-7,
  - http://www.opencard.org/SCJavaBook
  - Orienté OCF et livré avec une carte pour les tests
- Open Card Framework (OCF)
  - http://www.opencard.org/
  - http://www.gemplus.fr/developers/technologies/opencard/index.htm
- Désormais maintenu par http://www.openscdp.org/ocf

Extra

- APDUTool
  - permet de tester et de comprendre les principales fonctions d'OCF.
    - http://membres-liglab.imag.fr/donsez/dev/apdutool
- BazTracker
- JCOP Tools
  - http://www.zurich.ibm.com/csc/infosec/jcop_tools/samples.html
- «SmartCard simulation with Virtual Card Terminal(VCAD) project»
  - http://tochna.technion.ac.il/project/vcad/html/vcadbook.html

Exercices

- Fusionnez les 2 EpurseCardService pour BasicCard et pour JavaCard en un seul en fonction de l'ATR de la carte insérée.
- Complétez les 2 EpurseCardService pour récupérer l'historique des dernières opérations.
- Ecrivez une applet qui affiche le solde d'une carte PME, valide le code NIP, débite le solde entré par l'usager. Testez cette applet dans un document HTML ouvert dans Netscape Communicator et dans MicroSoft Internet explorer.
- Ecrivez un BCCryptoCardService utilisant une BasicCard pour chiffrer, déchiffrer, signer des Streams (il devra implémenter l'interface opencard.opt.signature.SignatureCardService).
- Ecrivez un JCCryptoCardService utilisant une JavaCard pour chiffrer, déchiffrer, signer des Streams (il devra implémenter l'interface opencard.opt.signature.SignatureCardService).
- Ecrivez un CardCryptoProvider pour JCE utilisant le fonctions de chiffrage/déchiffrage du BCCryptoCardService et JCCryptoCardService de deux exercices précédents.
- Ecrivez un BCFileCardService permettant de manipuler les fichiers d'une BasicCard.
- Ecrivez une servlet qui sert les fichiers d'une BasicCard insérée dans le lecteur du poste sur lequel le serveur HTTP tourne. Vous utilisez le BCFileCardService et Jakarta/Tomcat comme serveur HTTP. Transformez votre carte en portail personnel avec votre liste de signets. Amusez vous à ajouter au fichier host de votre machine la ligne suivante pour rentrez l'adresse http://card:8080/index.htm dans votre navigateur.
  - card 127.0.0.1