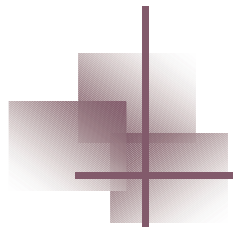


Secure Object Sharing Development Kit for Java Card

Daniel Perovich

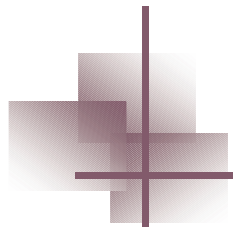
January 9, 2002





Agenda

- Shareable Interface Object mechanism
- Known drawbacks
- Challenge/Response mechanism
- Approaches
 - Delegate Object approach
 - Methodological approach
- Secure Object Sharing Development Kit
 - SOS Framework
 - SOS Toolkit
- Conclusion and Future Work



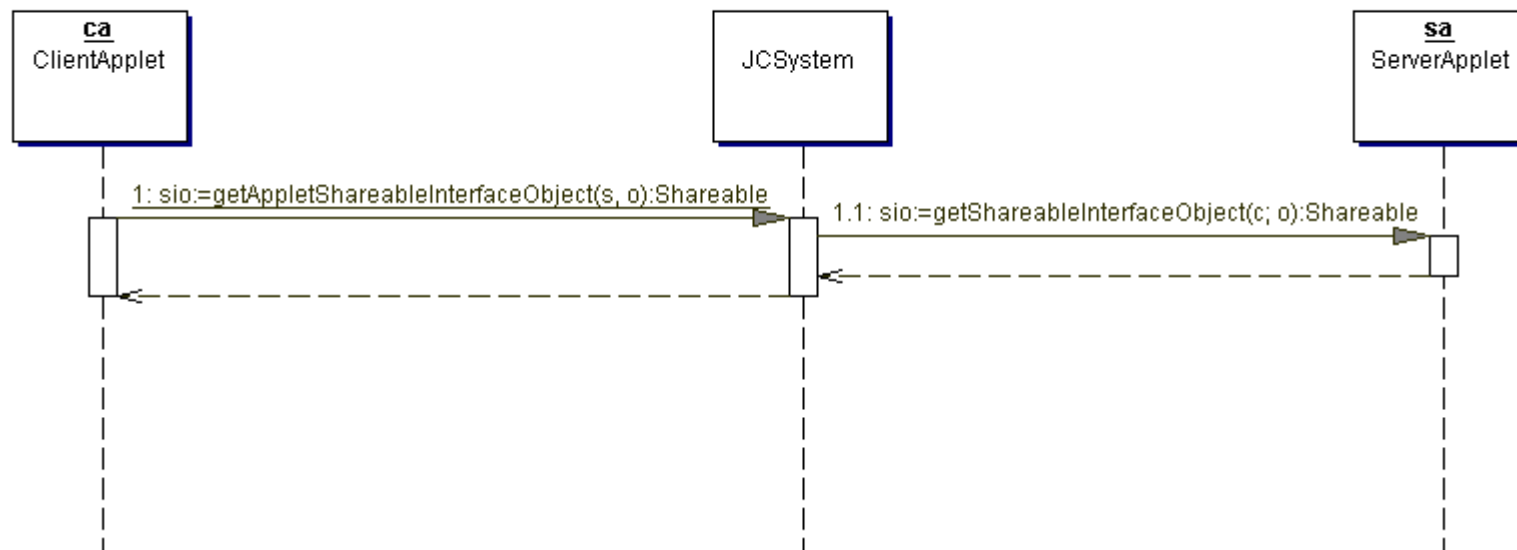
Java Card basics

- Multi-application smart cards
- Applets from different vendors can coexist in a single card
- Post-issuance of applets
- Applet uniquely identified by an AID
- Firewall protection prevents illicit access to other applet's data
- Inter-applet collaboration provided by the Shareable Interface Object mechanism



Shareable Interface Object mechanism

- Extend and implement the `Shareable` interface to by-pass firewall restrictions
- Applet AID is used for service requests





Known Drawbacks

- (1) Applet impersonation
 - Client authorization to obtain a SIO is based on the AID
 - A malicious applet could be installed with the same AID of a valid client
 - In this compromised card, the malicious client can get access to restricted services and data

- (2) Limited clients
 - With this selection criterion the server must know the AID of every possible client
 - It is impossible to allow access to new valid clients once the server applet has been deployed and widely distributed

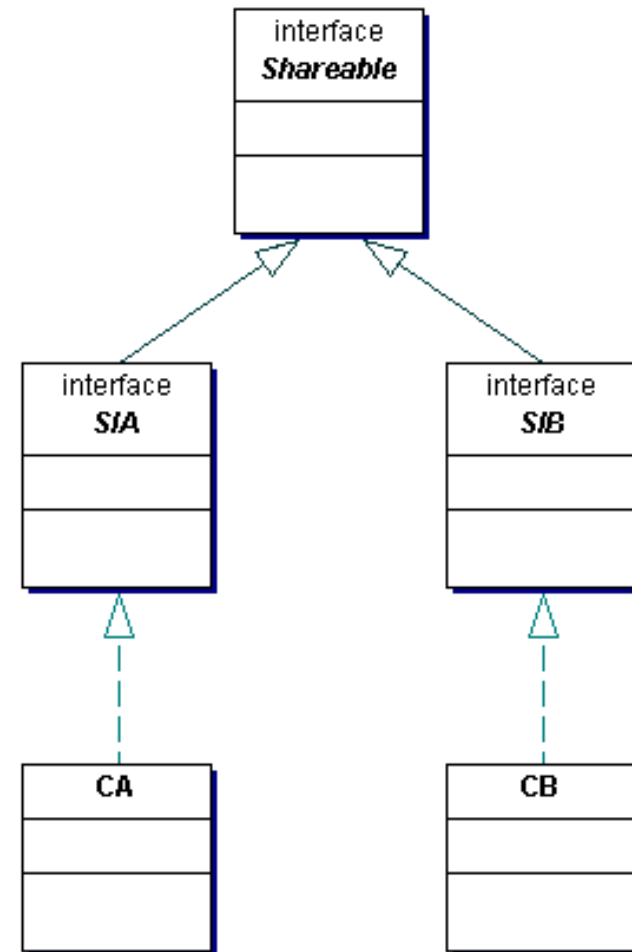


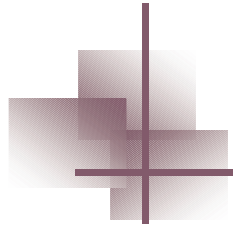
Known Drawbacks (2)

- (3) Inappropriate casting
 - A client can get access to a SIO for which it is not authorized
 - This may happen only if a class implements more than one `Shareable Interface`
 - The client can cast the reference to the interface it has not been authorized from the one it has been authorized
- (4) Impossibility of passing objects as parameters
 - If the operations in the SI receive or return references, they cannot be used due to the firewall protection

Inappropriate Casting

- Implements each Shareable Interface in a separate class
- Any two classes implementing different Shareable Interfaces must not be in the subclassing relation





Challenge/Response mechanism

- Proposed approaches are based on it for solving Applet Impersonation and Limited Clients
- It relies on a shared knowledge
- Not adequate for untrusted clients
- Solution:
 - The provider develops a Certificate class and it is sold to the client developer
 - This class authenticates to the server
 - It is intended for a specific client
 - At most one instance of the Certificate class, and can be use at most by one client



Delegate Object Approach

- A Delegate Object is registered with the applet
- It provides an interface for all applet services
- Those services can be both fields and operations
- The Delegate Object can be used by any applet
- Client authentication takes place every method invocation



Delegate Object Approach (2)

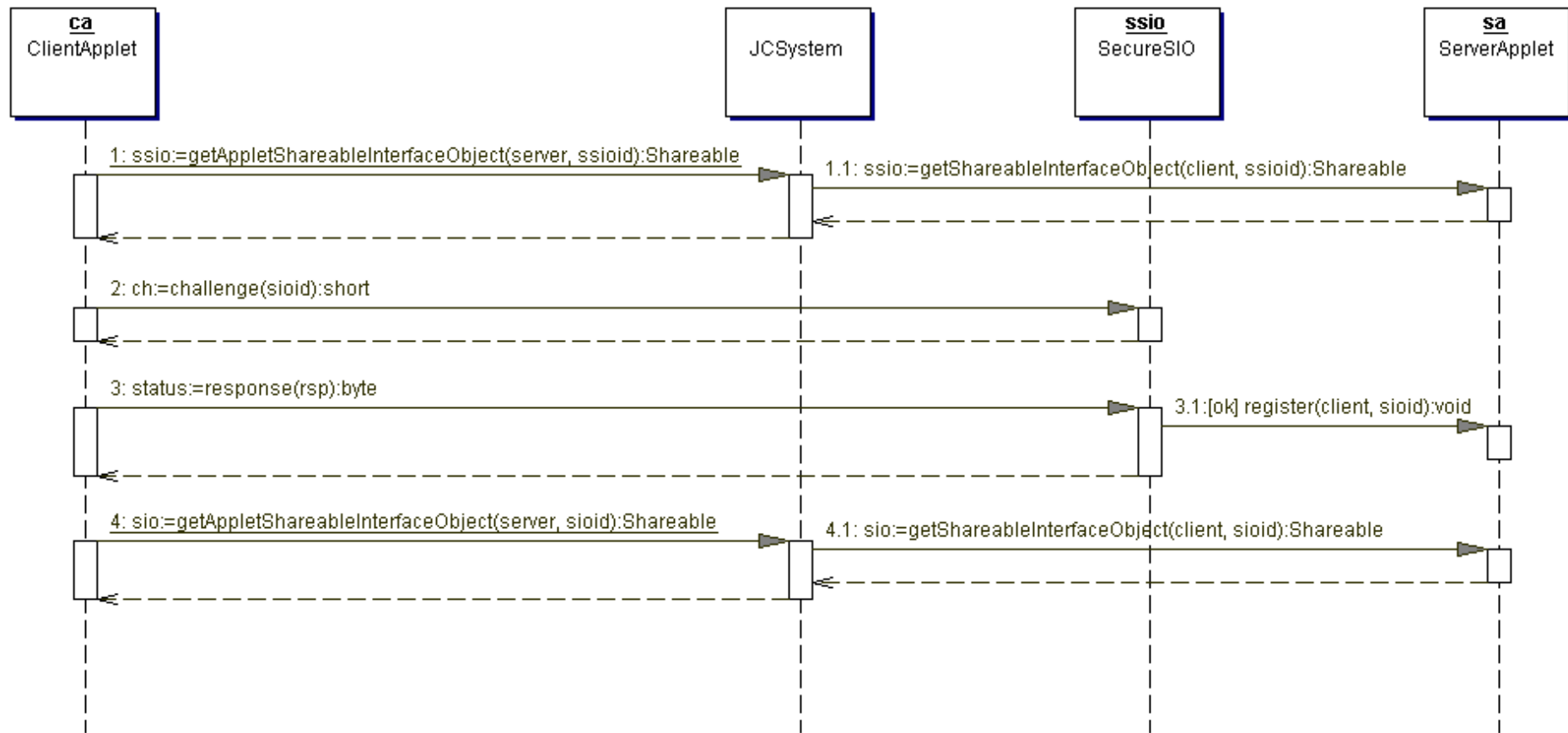
- Drawbacks:
 - It eliminates the current object sharing mechanism as legitimate access is provided by the delegate object
 - All exported services must be defined in one class
 - It requires changes to the JCRE
 - This introduces incompatibility issues with existing hardware and software



Methodological Approach

- Based on similar concepts as the Delegate Object approach
 - Challenge/Response authentication
 - Fine-grained control over access to applet services
- Key concepts
 - A Shareable Interface SecureSI is defined which provides the authentication mechanism
 - A class SecureSIO implements the SecureSI interface
 - An Authorization Manager maintains the set of registered clients

Methodological Approach (2)



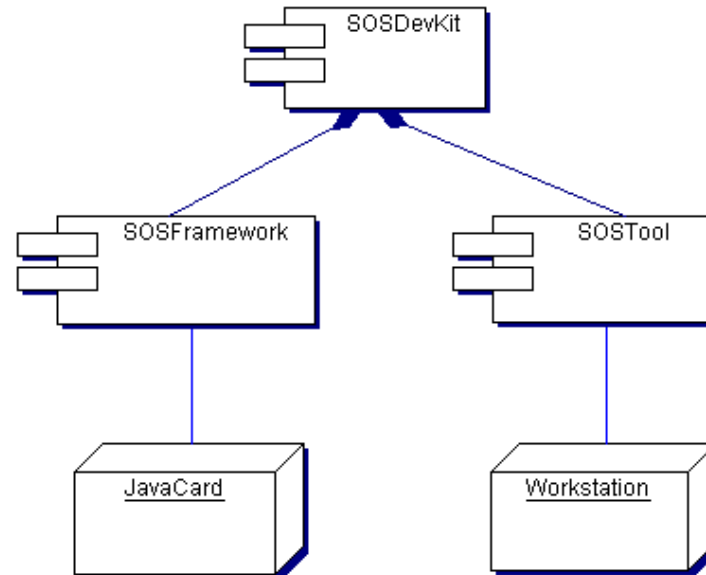


Methodological Approach (3)

- Advantages:
 - It does not require changes to the JCRE
 - The current object sharing mechanism is preserved
- Disadvantages:
 - The mechanism must be implemented for each server applet
 - The Authorization Manager is not used for checking valid access to services, it is just used for checking valid requests of SIOs

Secure Object Sharing Development Kit

- The main goal is to lighten the developer's work by providing an Object Sharing mechanism in which the problems are already solved

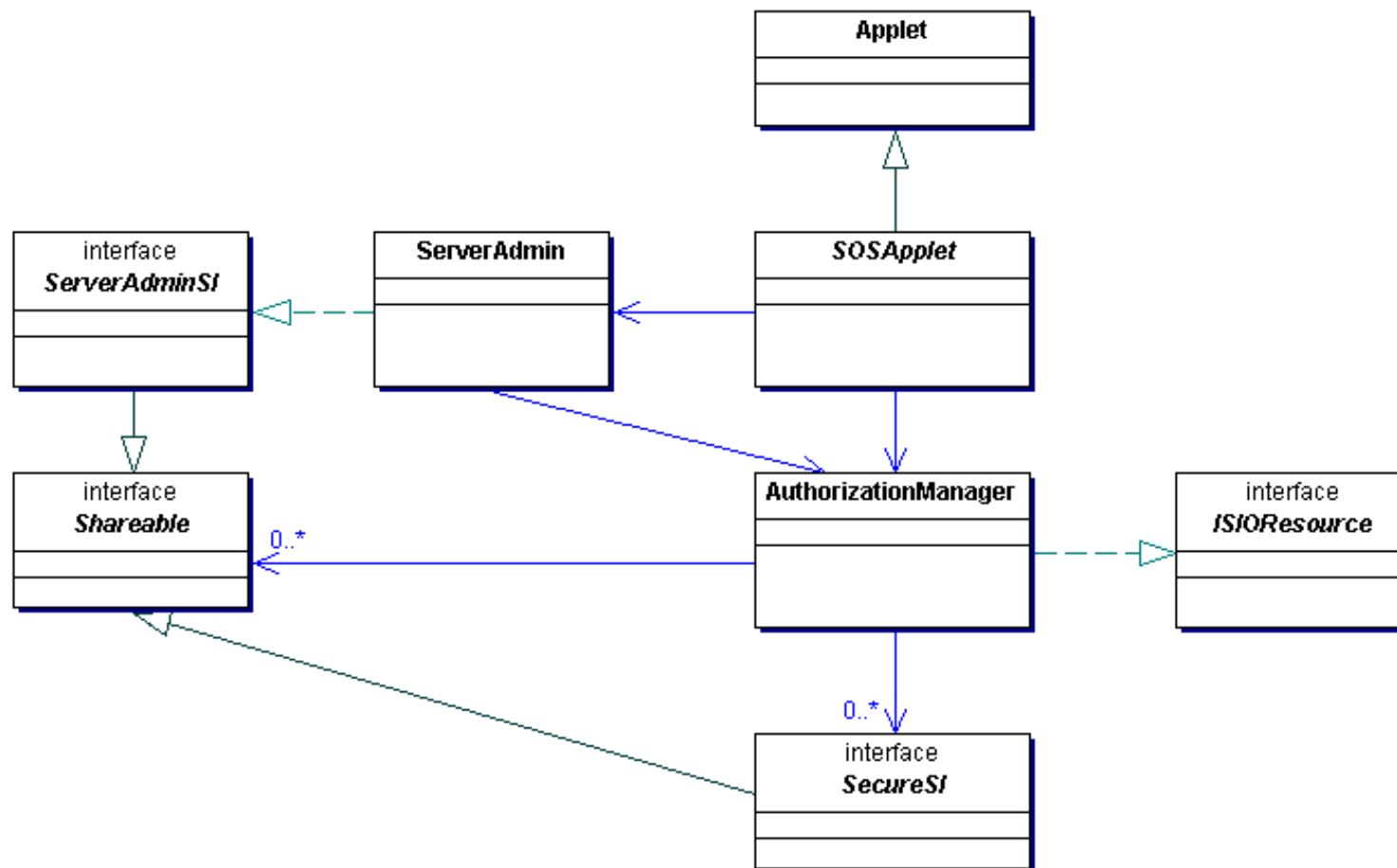




Secure Object Sharing Framework

- Based on the Methodological approach
- Improves it by providing extra services
 - Allows server applet to block clients
 - Allows to change service availability
 - Automatically blocks clients which do an incomplete authentication procedure
 - Authentication mechanism can be extended by the server applet developer
- It must be loaded just once in the Java Card

Secure Object Sharing Framework (2)



Secure Object Sharing Framework (3)

- SOSApplet object sharing mechanism:

```
public final Shareable getShareableInterfaceObject(AID clientAID, byte parameter) {
    if (parameter == 0) {
        return lvServerAdminm;
    }
    else {
        if (lvAM.canBeProvided(parameter)) {
            if (!lvAM.isBlocked(clientAID)) {
                if (lvAM.isAvailable(parameter)) {
                    if (lvAM.isRegistered(parameter, clientAID)) {
                        lvServerAdmin.setRejectionReason(ServerAdmin.NOT_REJECTED);
                        return lvAM.getSIO(parameter);
                    }
                    else {
                        lvServerAdmin.setRejectionReason(ServerAdmin.CLIENT_NOT_REG_TO_SIO);
                        return lvAM.getSecureSI(parameter, clientAID);
                    }
                }
            }
        }
    }
}

// ...
```

Secure Object Sharing Framework (4)

- Services provided to sub-applets:

```
public interface ISIOResource {
    public void provideSIO(byte pSIOID, Shareable pSIO, SecureSI pSecureMechanism);

    public void makeAvailable(byte pSIOID);
    public void makeUnavailable(byte pSIOID);
    public boolean isAvailable(byte pSIOID);

    public boolean isAuthorized(byte pSIOID);
    public boolean isAuthorized(byte pSIOID, AID pClient);

    public void block();
    public void block(AID pAID);
    public boolean isBlocked(AID pAID);

    public void register();
    public void register(byte pSIOID, AID pClient);
    public boolean isRegistered(byte pSIOID, AID pClient);
}
```

Secure Object Sharing Framework (5)

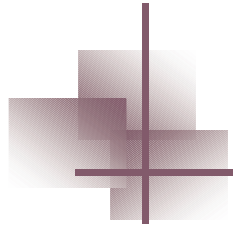
- Example of service implementation:

```
public class DebitSIO implements DebitSI {  
  
    private PApplet lvApplet;  
  
    public DebitSIO(PApplet pApplet) {  
        lvApplet = pApplet;  
    }  
  
    public void debit(short pAmount, byte pCode) {  
  
        // Checks if the client is authorized to use the Debit operation  
        if (lvApplet.getSIOResource().isAuthorized(ePurseInfo.PDEBIT_SIOID)) {  
            lvApplet.debit(pAmount, pCode);  
        }  
    }  
}
```



Secure Object Sharing Toolkit

- Inappropriate Casting:
 - Checks that every class implements at most one Shareable interface
- Dynamic Checks:
 - The framework relies on dynamic checks when a client request a service from a SIO
 - The framework cannot prevent illicit service request directly as the request is placed to the SIO itself, not to a class on the framework
 - This tool checks that each SIO do the necessary dynamic check before providing each service



Conclusion

- The SOSDevKit provides framework and tools for overcoming the Shareable Interface Object mechanism drawbacks
- Solves problems found in previous approaches
- Dynamic checks does not imply to go through the authentication procedure
- `cap` file size of all framework classes is 4 KB, so it fits well in current Java Cards
- An alternative to the Challenge/Response mechanism was proposed



Future Work

- The development of the two tools proposed by the SOSDevKit
- Finish the implementation of the Electronic Purse Case Study using the framework
 - ePurse + AirFranceLoyalty + RentACarLoyalty can be solved using the framework
- Reformulate the Certificate approach where only interface `class` files are provided to the client

Secure Object Sharing Development Kit for Java Card

Thank you

Daniel Perovich

January 9, 2002

