



**JavaOne**<sup>SM</sup>

Sun's 2000 Worldwide Java Developer Conference<sup>SM</sup>

# Tools for Integrating the Java Card<sup>TM</sup> API into Jini<sup>TM</sup> Connection Technology

**Eric Vétillard**

Java Card Architect  
Gemplus Software

# Prologue

- Vision
  - Java Card™ technology provides small, personal and secure programmable servers hosting services
  - Card applets are part of a global distributed information system
- Goals
  - To provide development tools and deployment facilities that sustain the vision
  - Based on RMI and Jini™ connection technology



# Summary

- The Java Card™ API and Distributed Applications
  - *At the development level*
  - At the deployment level
- Towards RMI on the Java Card™ platform
- Towards the Java Card™ API with Jini™ connection technology

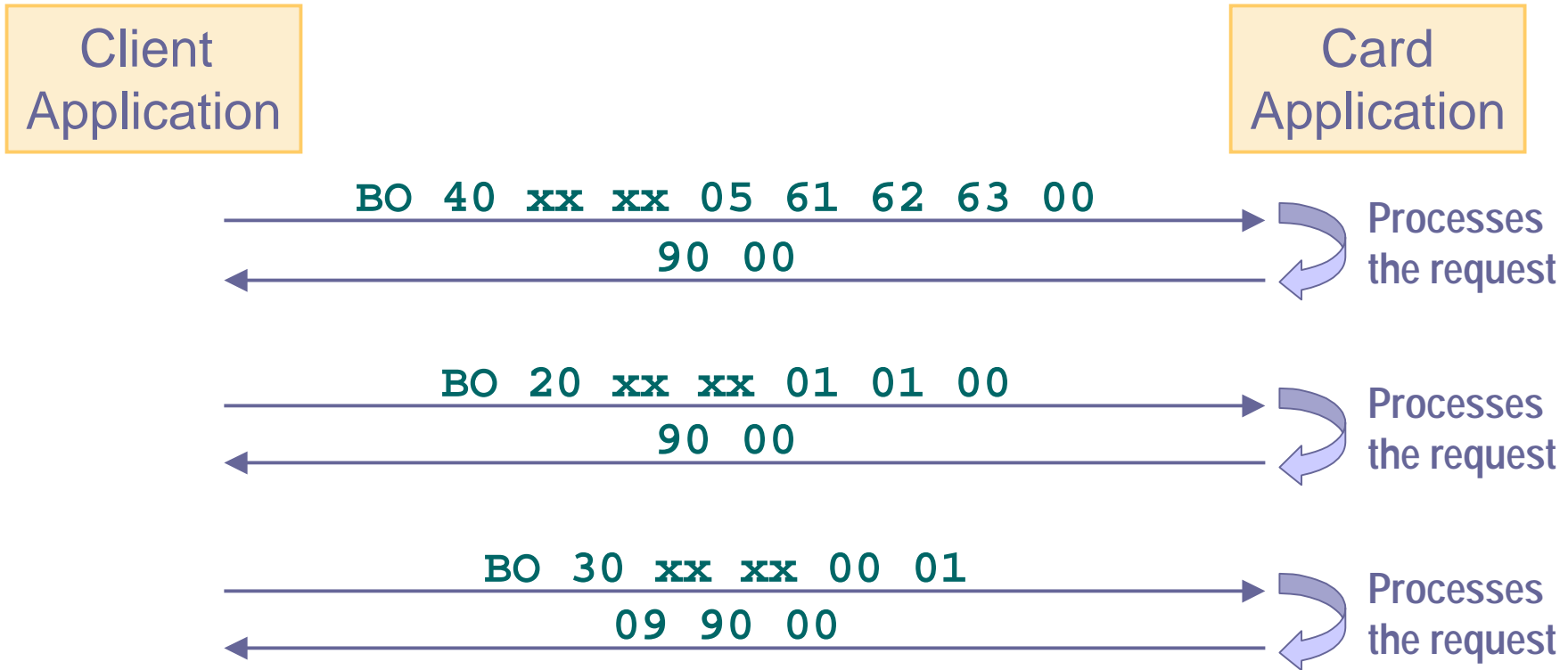


# The Smallest Java™ Platform

- Smart cards are limited devices
  - 128K ROM, 64K EEPROM, 4K RAM
  - 1 serial port for communication
  - Single-threaded, no internal clock
- Java Card specification also is limited
  - Very small footprint, very small API
  - Low-level communication
  - All services are provided by applications



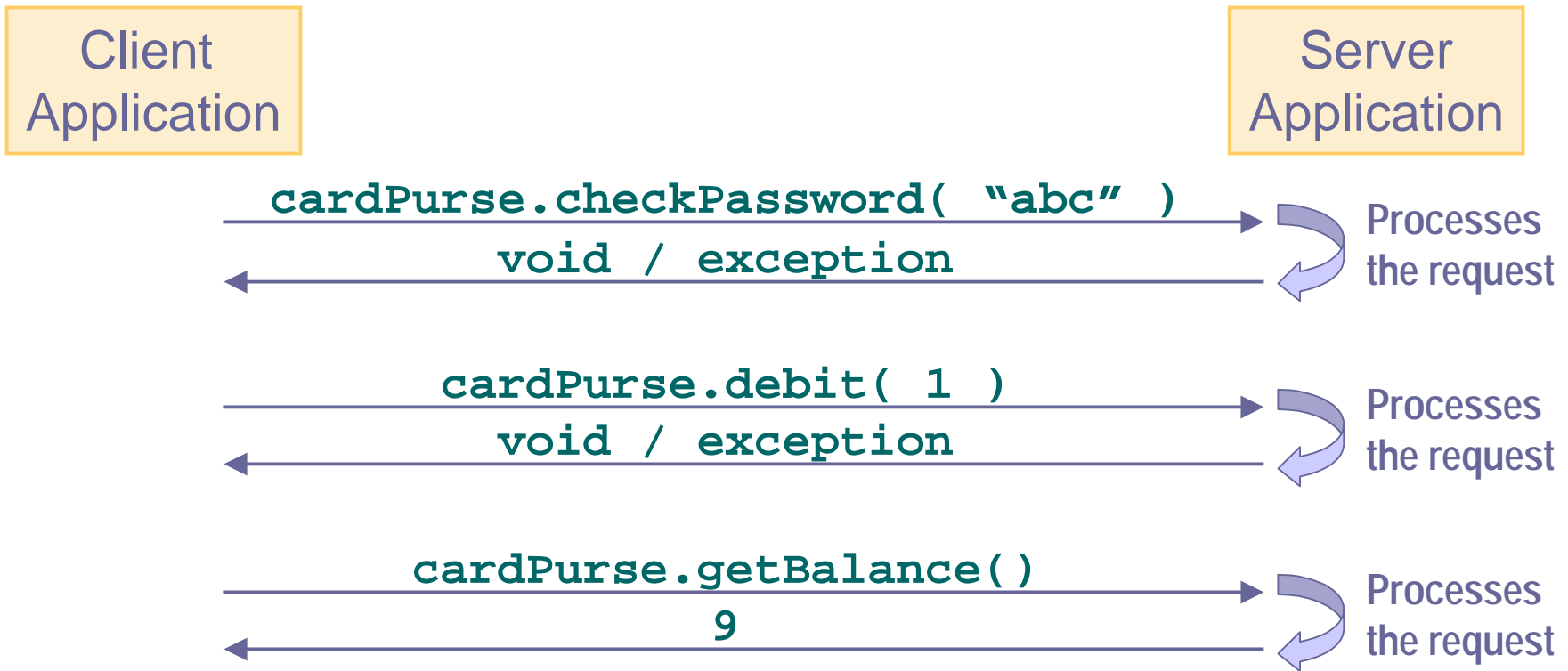
# Java Card API- Based Application



Exchanges are done through APDU messages supported by the smart card protocol



# Client/Server Application



Exchanges are done through method invocations supported by the object abstraction



# Development Issues

- Two kinds of applications
  - Traditional smart card applications
  - “New” applications, without history
- Java Card technology is a core platform
  - Satisfactory for traditional applications
  - Not optimal for new application design
    - Smart card knowledge is required
    - Design needs to be very detailed



# Development Vision

- Provide a high-level application model
  - To provide a rich semantics of messages
  - To reduce the communication errors
  - To avoid dealing with low-level protocols
- Be compatible with Java Card 2.1 API
- Integrate Java Card platform in mainstream Java platform



# Summary

- The Java Card™ API and Distributed Applications
  - At the development level
  - *At the deployment level*
- Towards RMI on the Java Card™ platform
- Towards the Java Card™ API with Jini™ connection technology



# Deploying a Card Application

- For a traditional application
  - Build specific terminals (POS, phones, ...)
  - Develop an ad-hoc software layer
- For a new application
  - Add a card reader/writer to an existing device
  - Develop ad hoc software
  - Low-level protocol remains exposed



# Deployment Vision

- See smart cards as servers
  - An applet is a service
  - Describe applets by Java platform interfaces
- Use an open card terminal
  - Ready to accept any application
  - See card applets as remote references



# Following...

- Technologies for developing and deploying Java Card API-based applications like distributed applications
  - a RMI framework dedicated to develop card services
  - a deployment strategy using Jini technology



# Summary

- The Java Card™ API and Distributed Applications
- Towards RMI on the Java Card™ platform
  - *Architecture*
  - Example
- Towards the Java Card™ API with Jini™ connection technology



# RMI in the J2SE™ Platform

- Client-server application protocol
  - Defines a format for method invocation
- Uses Java platform interface as contracts
  - Low-level code is automatically generated
- Supports mobile code



# RMI Advantages

- Communicates through invocation
- No message encoding and decoding
  - Independent of low-level protocol
- Reduces communication errors
  - Reports problems with RemoteException



# A Java Card Platform as RMI Server?

- Many features are missing
  - No introspection, no serialization
  - No IP support
- Limited requirements
  - Interfaces as contracts
  - Communication through invocation
  - Hidden low-level protocol
  - No code mobility



# Toward a Specification

- RMI-like protocol through APDU's
- Compatibility with Java platform RMI
  - For a subset of the RMI functionality
- Using RMI's 5-step methodology
- Compatible with Java Card 2.1 API



# RMI Subset

- Remote object features
  - Designed with Remote and RemoteException
  - Generated skeleton is a Java Card API-based applet instance
- Remote method features
  - Parameters and return types can be primitive types (boolean, byte, short) and single-dimensional arrays.
  - Java Card API exceptions are reported like in RMI



# Five-Step Development

## Java Card Application

Interface  
Definition

- 1 Define the remote interface to your card service**



# Five-Step Development

## Java Card Application

Interface  
Definition

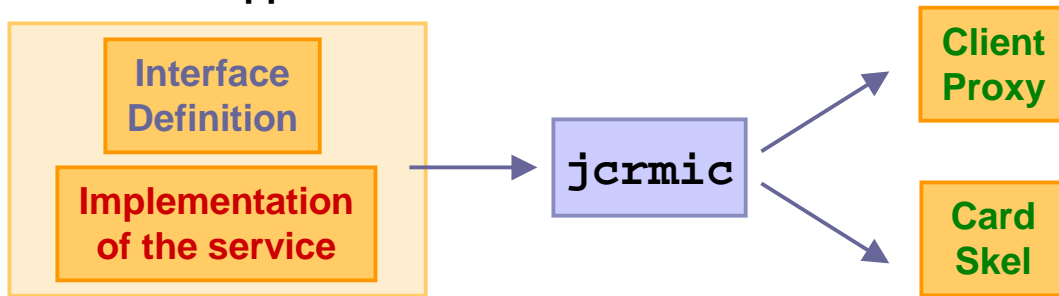
Implementation  
of the service

## ② Implement your card service



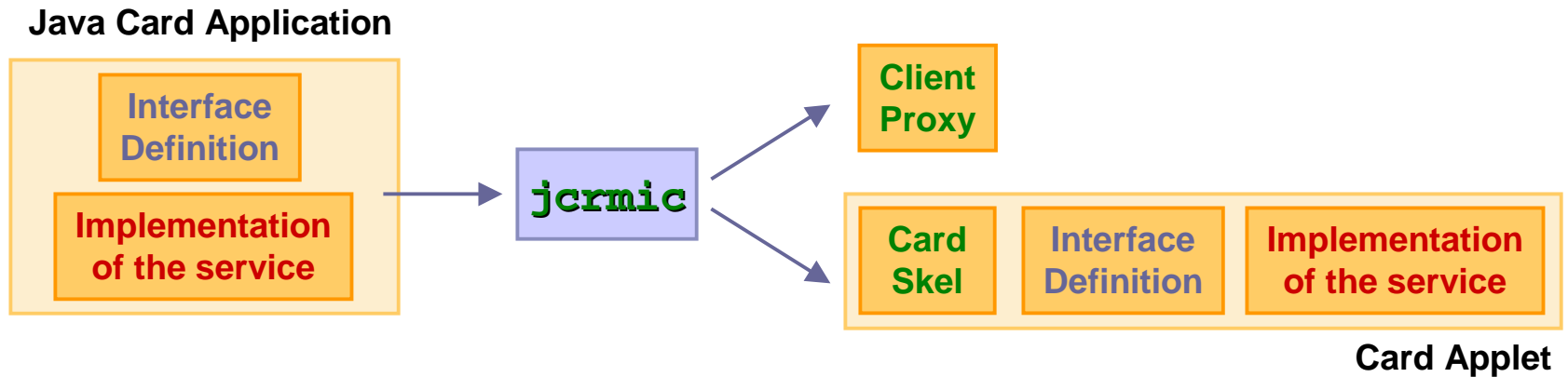
# Five-Step Development

## Java Card Application



③ Run “**jcrmic**” on the implementation classes

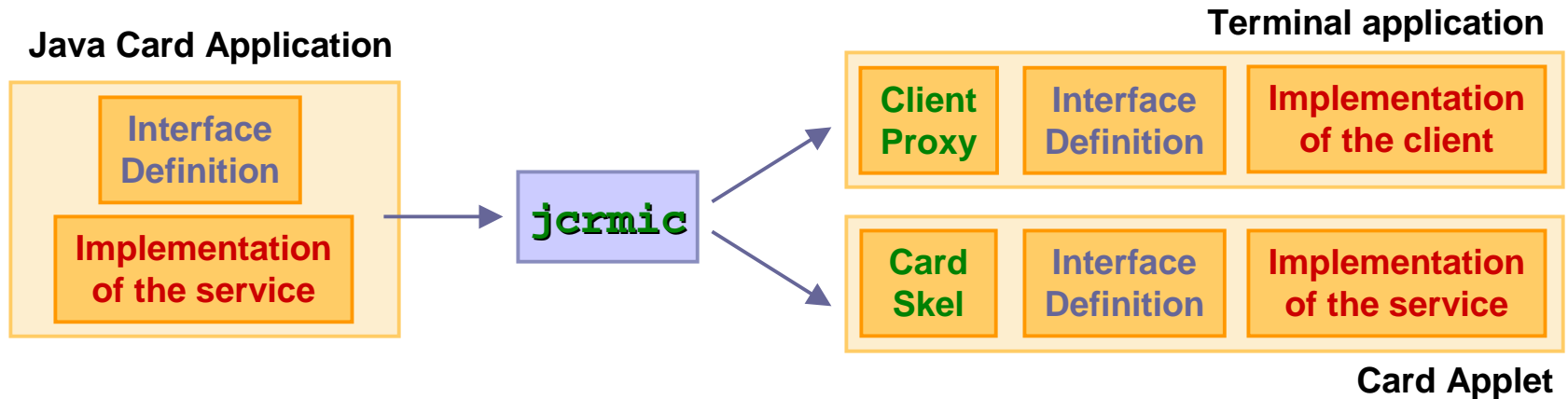
# Five-Step Development



## ④ Install your service with the **Card Skel** in the card



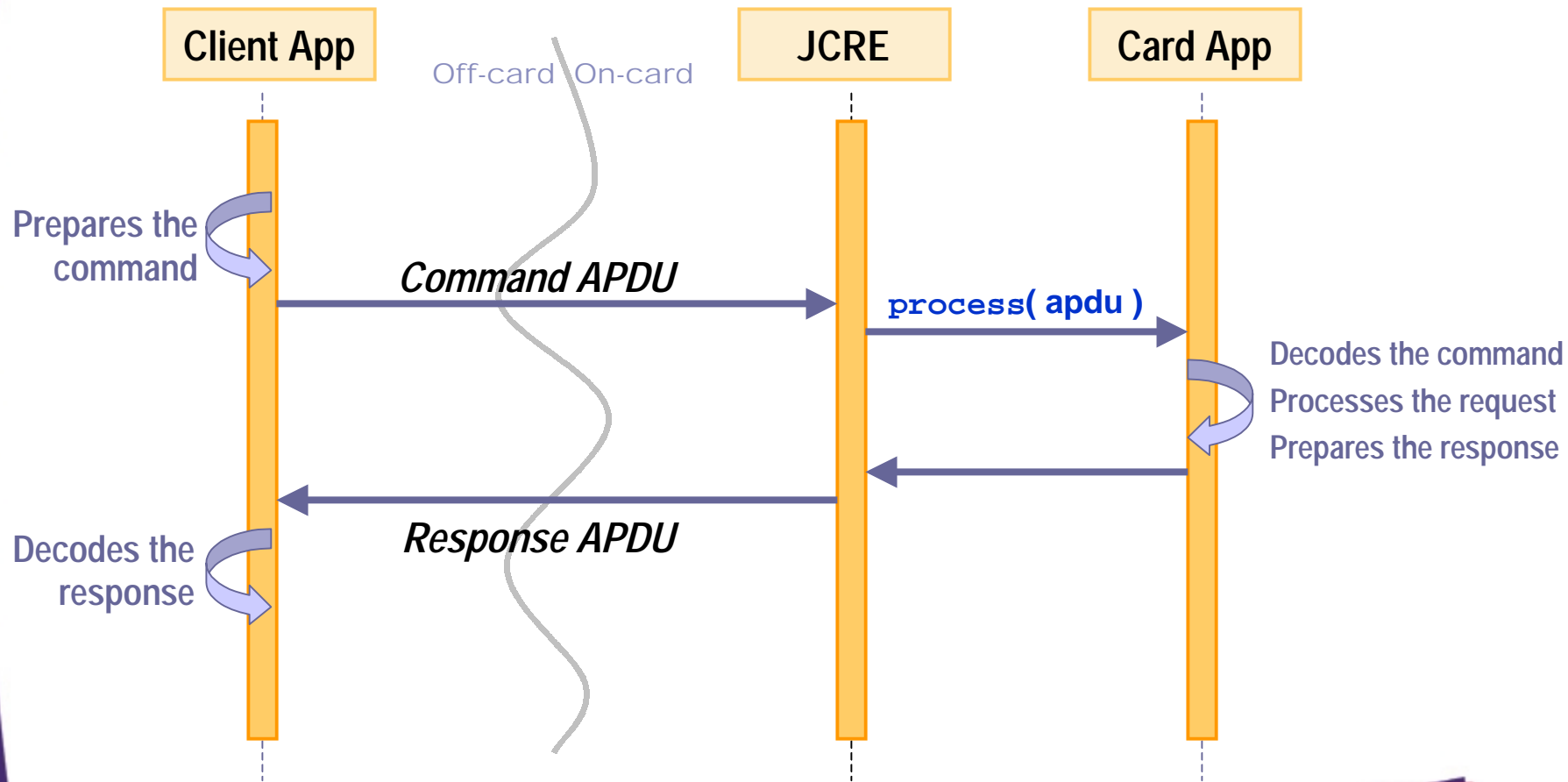
# Five-Step Development



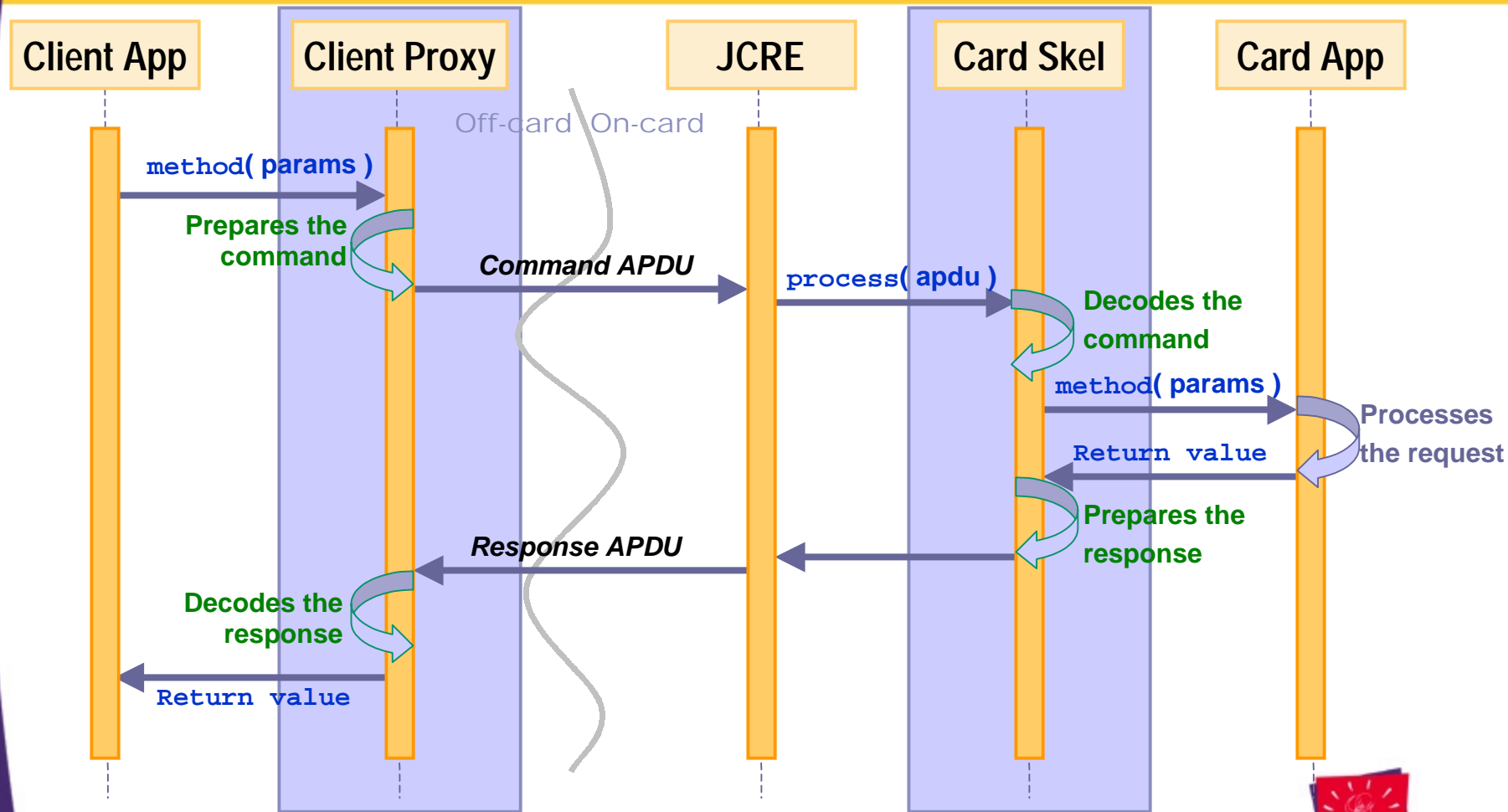
- 5 Use the **Client Proxy** to invoke card service's methods from your client application



# Java Card 2.1 Platform: Sequence Diagram



# Java Card 2.1 RMI: Sequence Diagram



# Java Card

## 2.1 RMI:Status

- Specifications
  - In course of discussion by the Java Card Forum
  - Will be proposed to Sun for next releases
- Implementation
  - On-card and off-card libraries, code generators, and IDE components are ready



# Java Card

## 2.1 RMI: Added-Value

- Simplify Java Card API-based applet development
  - Hide low-level protocol details
  - Handle both card and terminal sides
- Use a distributed computing model
- Push Java Card technology closer to Java technology
  - Java Card technology can be part of a distributed app



# Summary

- The Java Card™ API and Distributed Applications
- Towards RMI on the Java Card™ platform
  - Architecture
  - *Example*
- Towards the Java Card™ API with Jini™ connection technology



# A Basic Card Service

- The card applet is a simple «Counter»
  - A positive integer value is stored by the applet
- Operations applicable to the counter
  - Read: returns the counter's value
  - Increment/decrement: adds/subtracts an amount to/from the counter's value, and returns the new counter's value



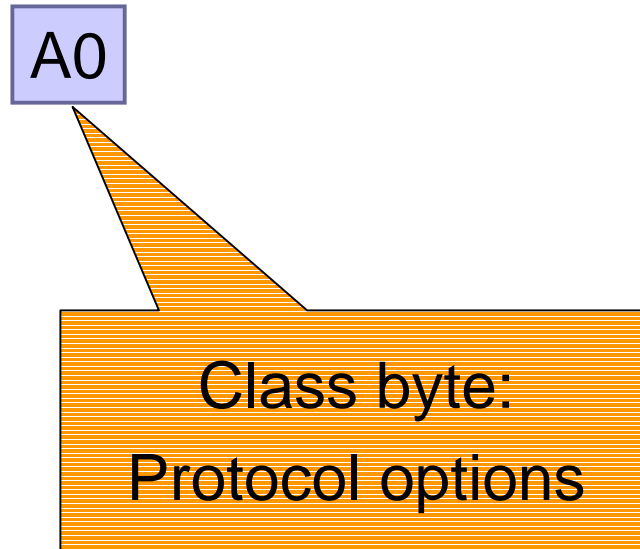
# Traditional Design

- Define application protocol
  - Command messages
    - Choose an instruction code
    - Define command options
    - Define format of command parameters
  - Response messages
    - Define format of response parameters
    - Define possible error codes
- Many constraints to consider!



# Design for Counter

- Read Command



# Design for Counter

- Read Command

A0 01

Instruction byte:  
Command Id

# Design for Counter

- Read Command

A0	01	00	00
----	----	----	----

Parameter bytes:  
Command Options

# Design for Counter

- Read Command

A0	01	00	00	00
----	----	----	----	----

Lc byte:  
Input data length

# Design for Counter

- Read Command

A0	01	00	00	00	02
----	----	----	----	----	----

Le byte:  
Expected data length

# Design for Counter

- Read Command

A0	01	00	00	00	02
----	----	----	----	----	----

Rh	RI
----	----

Output data:  
Two bytes

# Design for Counter

- Read Command

A0	01	00	00	00	02
----	----	----	----	----	----

Rh	RI	90	00
----	----	----	----

Status word:  
Two bytes (standard)

# Design for Counter

- **Read Command**

A0	01	00	00	00	02
----	----	----	----	----	----

Rh	RI	90	00
----	----	----	----

Input data:  
Two bytes

- **Increment Command**

A0	02	00	00	02	Vh	VI	02
----	----	----	----	----	----	----	----

Rh	RI	90	00
----	----	----	----



# The Applet Class

```
import javacard.framework.*;

public class Counter extends Applet {

    private short value;

    public Counter() { value = 0; }

    public static void install(
        byte[] array, short offset, byte len ) {
        Counter myCounter = new Counter();
        myCounter.register();
    }

    public void process( APDU apdu )
    {
        ...
    }
}
```



# The Applet Class

```
import javacard.framework.*;

public class Counter extends Applet {

    private short value;

    public Counter() { value = 0; }

    public static void install(
        byte[] array, short offset, byte len ) {
        Counter myCounter = new Counter();
        myCounter.register();
    }

    public void process( APDU apdu )
    {
        ...
    }
}
```



# The Applet Class

```
import javacard.framework.*;

public class Counter extends Applet {

    private short value;

    public Counter() { value = 0; }

    public static void install(
        byte[] array, short offset, byte len ) {
        Counter myCounter = new Counter();
        myCounter.register();
    }

    public void process( APDU apdu )
    {
        ...
    }
}
```



# The Applet Class

```
import javacard.framework.*;

public class Counter extends Applet {

    private short value;

    public Counter() { value = 0; }

    public static void install(
        byte[] array, short offset, byte len ) {
        Counter myCounter = new Counter();
        myCounter.register();
    }

    public void process( APDU apdu )
    {
        ...
    }
}
```



# The Applet Class

```
import javacard.framework.*;

public class Counter extends Applet {

    private short value;

    public Counter() { value = 0; }

    public static void install(
        byte[] array, short offset, byte len ) {
        Counter myCounter = new Counter();
        myCounter.register();
    }

    public void process( APDU apdu )
    {
        ...
    }
}
```



# The Process Method

```
public void process( APDU apdu ) {
    byte[] buffer = apdu.getBuffer();

    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);

    switch ( buffer[ISO.OFFSET_INS] ) {
        case 0x01: ... // The «read» operation
        case 0x02: ... // The «increment» operation
        case 0x03: ... // The «decrement» operation
        default:
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```



# The Process Method

```
public void process( APDU apdu ) {  
    byte[] buffer = apdu.getBuffer();  
  
    if ( buffer[ISO.OFFSET_CLA] != 0x00 )  
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);  
  
    switch ( buffer[ISO.OFFSET_INS] ) {  
        case 0x01: ... // The «read» operation  
        case 0x02: ... // The «increment» operation  
        case 0x03: ... // The «decrement» operation  
        default:  
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);  
    }  
}
```



# The Process Method

```
public void process( APDU apdu ) {
    byte[] buffer = apdu.getBuffer();

    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);

    switch ( buffer[ISO.OFFSET_INS] ) {
        case 0x01: ... // The «read» operation
        case 0x02: ... // The «increment» operation
        case 0x03: ... // The «decrement» operation
        default:
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```



# The Process Method

```
public void process( APDU apdu ) {
    byte[] buffer = apdu.getBuffer();

    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);

    switch ( buffer[ISO.OFFSET_INS] ) {
        case 0x01: ... // The «read» operation
        case 0x02: ... // The «increment» operation
        case 0x03: ... // The «decrement» operation
        default:
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```



# The Process Method

```
public void process( APDU apdu ) {
    byte[] buffer = apdu.getBuffer();

    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);

    switch ( buffer[ISO.OFFSET_INS] ) {
        case 0x01: ... // The «read» operation
        case 0x02: ... // The «increment» operation
        case 0x03: ... // The «decrement» operation
        default:
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# The Decrement Operation

```
case 0x03: { // The «decrement» operation
    // Decodes the data from the APDU I/O buffer
    byte byteRead = apdu.setIncomingAndReceive();
    if ( byteRead != 2 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    short amount =
        Util.getShort(buffer, ISO7816.OFFSET_CDATA) ;

    // Processes the command with the received data
    if (amount < 0 || value - amount < 0 )
        ISOException.throwIt((short)0x6910);
    value = value - amount;

    // Returns the response
    Util.setShort(buffer, (short)0, value);
    apdu.setOutgoingAndSend((short)0, (short)2);
    return;
}
```



# Development With Java Card™ 2.1 API RMI

- Design the Counter interface
  - Design the method signatures
  - Define required constants
  - Add checked exceptions
- Then, implement a class ...



# Interface ICounter

```
import java.rmi.Remote ;
import java.rmi.RemoteException ;
import javacard.framework.UserException ;

public interface ICounter extends Remote
{
    public static final short NEGATIVE_VALUE = 1;

    public short read()
        throws RemoteException;

    public short increment( short amount )
        throws RemoteException, UserException;

    public short decrement( short amount )
        throws RemoteException, UserException;
}
```



# Interface ICounter

```
import java.rmi.Remote ;
import java.rmi.RemoteException ;
import javacard.framework.UserException ;

public interface ICounter extends Remote
{
    public static final short NEGATIVE_VALUE = 1;

    public short read()
        throws RemoteException;

    public short increment( short amount )
        throws RemoteException, UserException;

    public short decrement( short amount )
        throws RemoteException, UserException;
}
```



# Interface ICounter

```
import java.rmi.Remote ;
import java.rmi.RemoteException ;
import javacard.framework.UserException ;

public interface ICounter extends Remote
{
    public static final short NEGATIVE_VALUE = 1;

    public short read()
        throws RemoteException;

    public short increment( short amount )
        throws RemoteException, UserException;

    public short decrement( short amount )
        throws RemoteException, UserException;
}
```



# Implementation Class Counter

```
public class Counter implements Icounter
{
    private short value;

    public Counter() { value = 0; }

    public short read() throws RemoteException {
        return value;
    }

    public short decrement(short amount)
        throws RemoteException, UserException {
        if (amount<0 || value-amount<0)
            UserException.throwIt(NEGATIVE_VALUE);
        value -= amount;
        return value;
    }
}
```



# Implementation Class Counter

```
public class Counter implements Icounter
{
    private short value;

    public Counter() { value = 0; }

    public short read() throws RemoteException {
        return value;
    }

    public short decrement(short amount)
        throws RemoteException, UserException {
        if (amount<0 || value-amount<0)
            UserException.throwIt(NEGATIVE_VALUE);
        value -= amount;
        return value;
    }
}
```



# Implementation Class Counter

```
public class Counter implements Icounter
{
    private short value;

    public Counter() { value = 0; }

    public short read() throws RemoteException {
        return value;
    }

    public short decrement(short amount)
        throws RemoteException, UserException {
        if (amount<0 || value-amount<0)
            UserException.throwIt(NEGATIVE_VALUE);
        value -= amount;
        return value;
    }
}
```



# Implementation Class Counter

```
public class Counter implements Icounter
{
    private short value;

    public Counter() { value = 0; }

    public short read() throws RemoteException {
        return value;
    }

    public short decrement(short amount)
        throws RemoteException, UserException {
        if (amount<0 || value-amount<0)
            UserException.throwIt(NEGATIVE_VALUE);
        value -= amount;
        return value;
    }
}
```



# Development With Java Card 2.1 API RMI

- Prepare the Card application
  - Generate the skeleton from the interface
  - Convert and download the applet
- Prepare the client application
  - Generate the client proxy from the interface
  - Write a client application using the proxy



# Summary

- The Java Card™ API and Distributed Applications
- Towards RMI on the Java Card™ platform
- Towards the Java Card™ API with Jini™ connection technology
  - *Architecture*
  - On-Going and Future Work



# Java Card API and Jini Technology?

- Smart cards can benefit from Jini technology
  - Card services can be used in Jini technology
  - A smart card can represent a user
  - Smart card use is dynamic by essence
- The Java Card API is a natural link to Jini technology
  - Object-oriented applications
  - Using the Java™ programming language
  - Using the RMI foundations



# Deployment Issues

- Java Card API-based devices are not Jini technology-ready
  - They need to be represented in the federation
  - The card terminal can be used for that purpose
- Java Card API-based applets are not Jini technology services
  - They need to be represented in the federation
  - Card service deployment information can be used for that purpose



# Using a Surrogate Host

- Bridging the Java Card API with Jini technology
  - Provides a communication link to the card
  - Provides a link to the Jini technology-based network
- Representing the Java Card API
  - Retrieves the Jini technology descriptions of the services
  - Instantiates the surrogate objects



# Using a Surrogate

- Representing a Java Card technology-based service
  - Encapsulates the proxy objects
  - Encapsulates the Jini description of the card service
- A member of the Jini technology federation
  - Registers the card service
  - Forwards incoming requests to the card applet



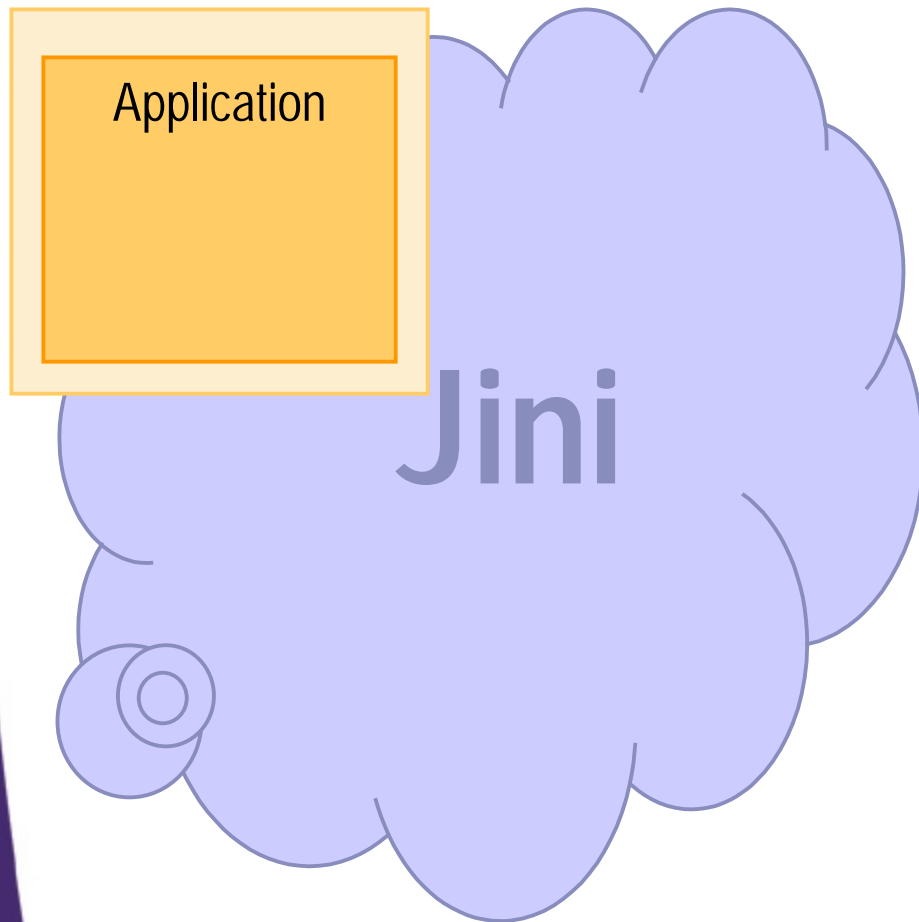
# Using a Service Provisioning Server

- The card maintains “card” information
  - AID’s of applets on the card
  - Service Id
  - Identification of SPS (URL, ...)
- The SPS maintains “Jini” information
  - class/provider attributes
  - Service Object
  - Service UI

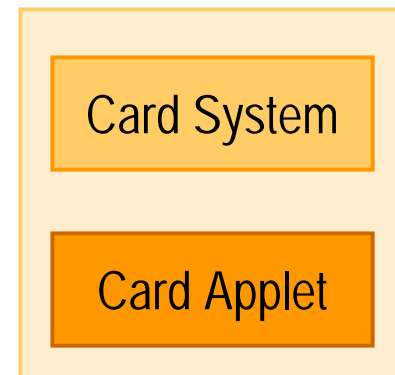


# Deployment Scenario With Jini Technology

## Client Machine

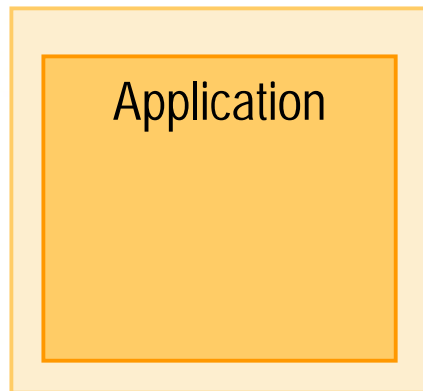


## Java Card

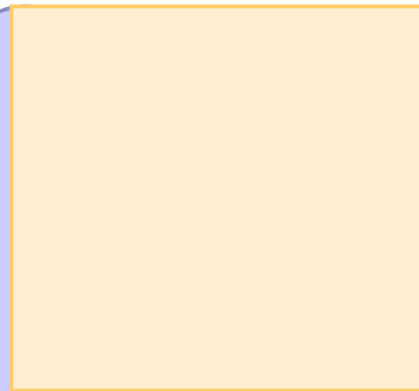


# Deployment Scenario With Jini Technology

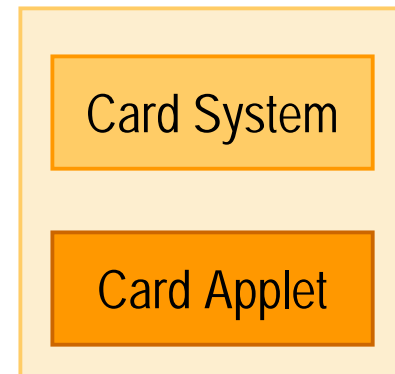
## Client Machine



## Surrogate Host



## Java Card



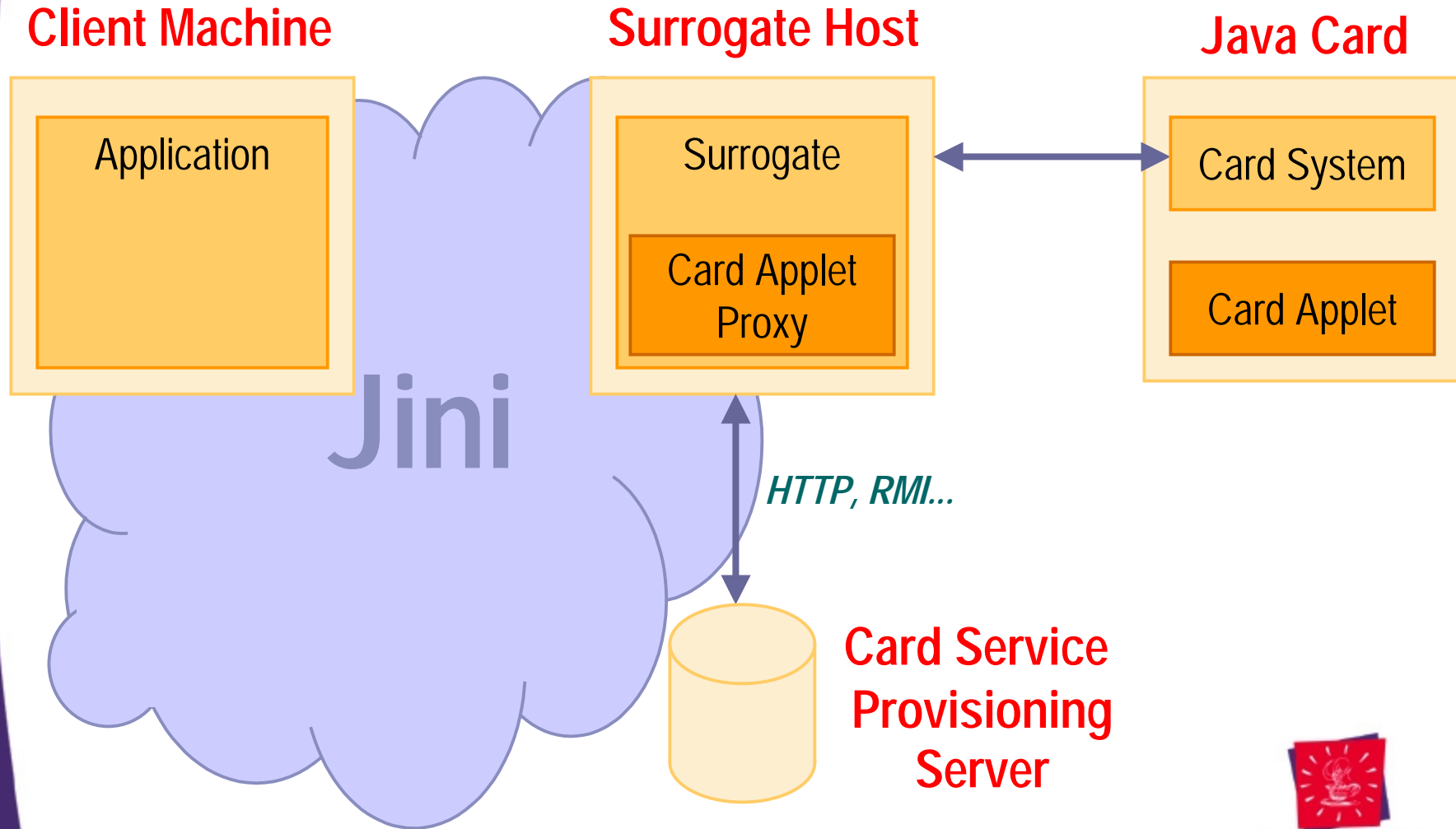
Jini



A large, light blue cloud-like shape with the word "Jini" written in a dark blue font across its center. The cloud has several smaller circles of varying sizes attached to its left side, resembling a stylized elephant's trunk or a decorative element.

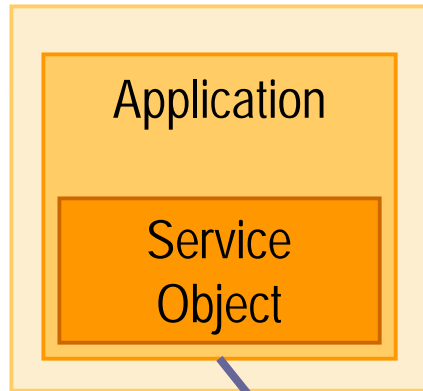


# Deployment Scenario With Jini Technology

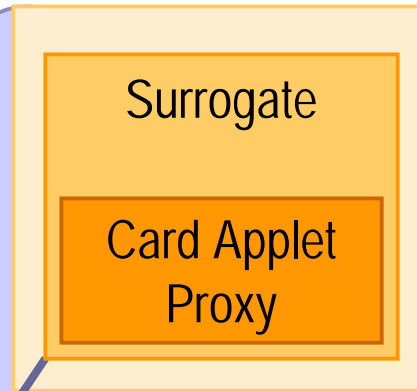


# Deployment Scenario With Jini Technology

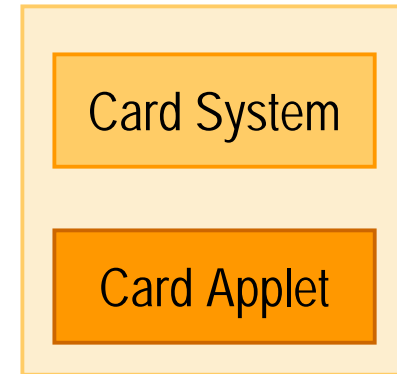
## Client Machine



## Surrogate Host



## Java Card

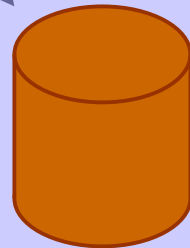


Jini

Lookup

Join

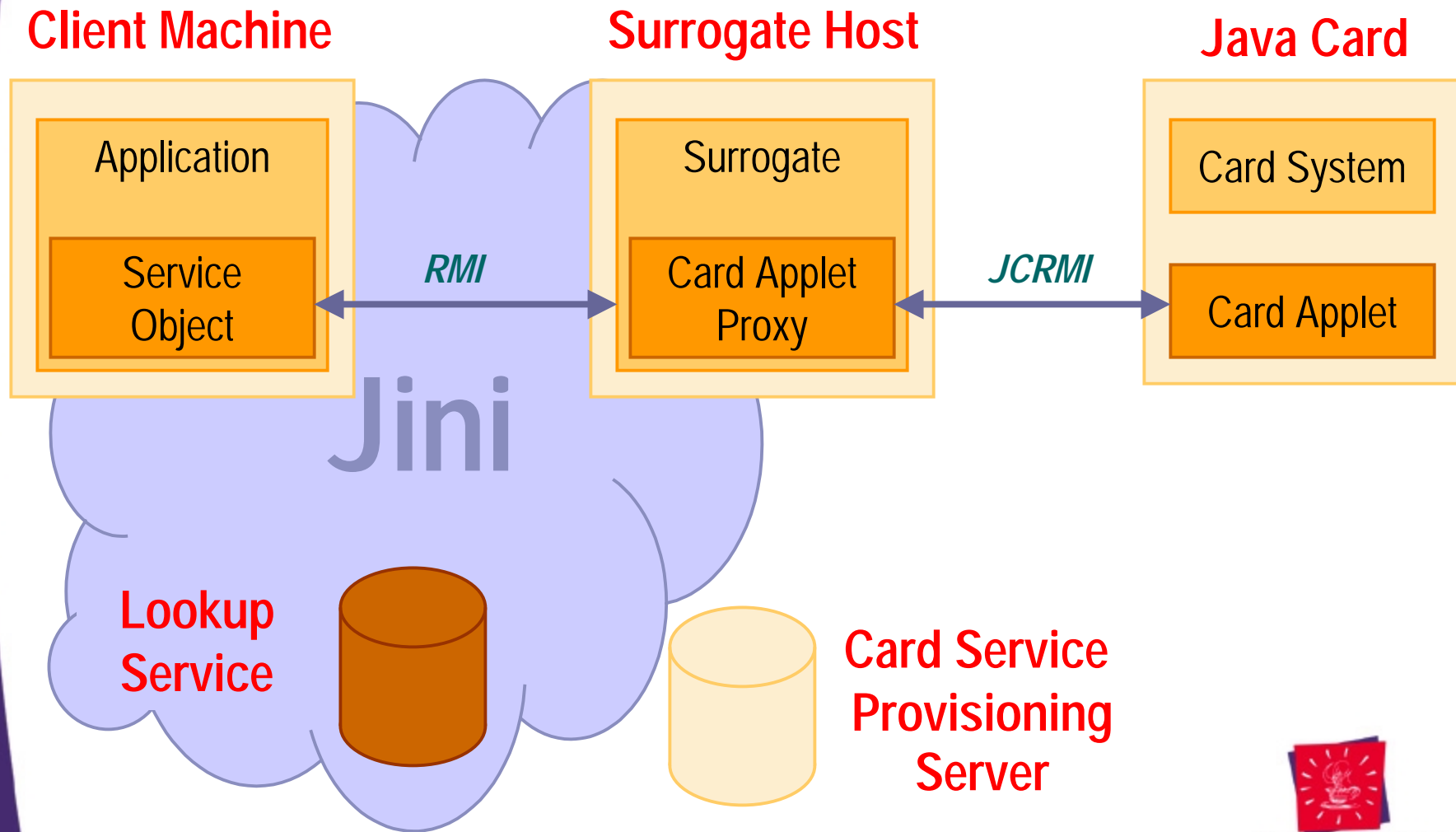
Lookup Service



Card Service Provisioning Server



# Deployment Scenario With Jini Technology



# Surrogate Architecture

- In discussion by the Jini Community
  - For all kinds of devices
  - *Cf.* TS 984
- Provides the device with many features
  - A full-fledged environment for the Java™ platform
  - Access to the surrogate host's resources
  - A connection to a Jini technology federation



# Java Card API-Specific Features

- The surrogate host bridges Java Card platform and Jini technology
  - A smart card contains several services
    - Several surrogates on the host
- The surrogate bridges two kinds of RMI
  - Very simple task (with introspection)
- A Service Provisioning Server is used
  - The surrogate host is not tailored for an app



# Service Provider Benefits

- Card services are easily deployed
  - Using a “standard” deployment infrastructure
    - Jini technology, card-enabled surrogate hosts and SPS
  - Simple interfaces and attributes definitions
    - Possibly integrated as business objects
- Simple and clear deployment task
  - Define information for storage in the SPS
    - Attributes, proxies, surrogates, GUI...
  - Ensure SPS is accessible



# Application Developer Benefits

- For the card developer
  - Combined with RMI, little extra work for developing proxies and surrogates
- For the client developer
  - Leverages the Jini technology application model
    - Card services are retrieved by their interfaces and attributes
    - Card services are used through their interfaces
  - Card services are accessed seamlessly



# Example

```
// Counter interface
Class interfaces[] = new Class[] { ICounter.class };
// One simple attribute
Entry[] entries = new Entry[]{new Name("J1 Counter")};
// Builds the template
ServiceTemplate template =
    new ServiceTemplate(null, interfaces, entries);

// Looks up
ServiceItem item = aLookupService.lookup(template);

if (item != null) {
    // Service retrieval and use
    ICounter counter = (ICounter)item.service;
    System.out.println("counter value: " + counter.read());
}
```



# Example

```
// Counter interface
Class interfaces[] = new Class[] { ICounter.class };
// One simple attribute
Entry[] entries = new Entry[]{new Name("J1 Counter")};
// Builds the template
ServiceTemplate template =
    new ServiceTemplate(null, interfaces, entries);

// Looks up
ServiceItem item = aLookupService.lookup(template);

if (item != null) {
    // Service retrieval and use
    ICounter counter = (ICounter)item.service;
    System.out.println("counter value: " + counter.read());
}
```



# Example

```
// Counter interface
Class interfaces[] = new Class[] { ICounter.class };
// One simple attribute
Entry[] entries = new Entry[]{new Name("J1 Counter")};
// Builds the template
ServiceTemplate template =
    new ServiceTemplate(null, interfaces, entries);

// Looks up
ServiceItem item = aLookupService.lookup(template);

if (item != null) {
    // Service retrieval and use
    ICounter counter = (ICounter)item.service;
    System.out.println("counter value: " + counter.read());
}
```



# Example

```
// Counter interface
Class interfaces[] = new Class[] { ICounter.class };
// One simple attribute
Entry[] entries = new Entry[]{new Name("J1 Counter")};
// Builds the template
ServiceTemplate template =
    new ServiceTemplate(null, interfaces, entries);

// Looks up
ServiceItem item = aLookupService.lookup(template);

if (item != null) {
    // Service retrieval and use
    ICounter counter = (ICounter)item.service;
    System.out.println("counter value: " + counter.read());
}
```



# Summary

- The Java Card™ API and Distributed Applications
- Towards RMI on the Java Card™ platform
- Towards the Java Card™ API with Jini™ connection technology
  - Architecture
  - *On-Going and Future Work*



# The Card Is the User

- A card typically represents a user
  - It contains personal information
  - It is usually kept close to the user
- Card services are very personal
  - Authentication or use rights
  - Payment (debit/credit or e-purse)
  - Personal information (agenda, ...)



# Linking the User, the Card and Jini Technology

- The surrogate host is the link
  - It usually is the machine on which the user works and plugs in the card
  - It can interact with the user
  - It bridges the card with the network
- The card acts on behalf of the user
  - Its services represent the user in the network
  - The user's personal data and secrets never leave the card



# Networking Personal Services

- **Keeps the user as the central interest of applications**
- **Enables fine-grained user's representation on the network**
- **Ensures security of the user's transactions on the network**



# Java Card 2.1 API and Jini Technology Status

- Experimental Gemplus project
  - Architecture in course of definition
    - Be compatible with Jini Technology architecture
    - Integrate smart card specificity
  - Prototype implementations based on Java Card 2.1 API RMI
- Needs consolidation from real-life applications



# Java Card 2.1 API and Jini Technology Added Value

- Demonstrate the potential of the Jini technology to integrate limited devices
- Enable generic terminals to support the deployment of any smart card service
- Demonstrate that Java Card platforms are mature participants in networked infrastructure



# Potential Enhancements

- Surrogates can include more services
  - Abstracting the card application
  - Providing a user interface
- Cards can be linked to Jini technology security
  - Cards can provide security-related services
  - Card services may require secure RMI
  - *Cf. TS 573*



# Extension of the work

- Other J2ME™ platform devices are similar to cards
  - They are limited devices
  - They need a specific connection interface
  - They are personal objects (PDAs, phones)
  - They can hold several applications
- The work can be extended
  - Defining RMI frameworks for other devices
  - Share the surrogate and SPS concepts





# JavaOne<sup>SM</sup>

Sun's 2000 Worldwide Java Developer Conference\*