



An Approach to Composition of EJB Components Using C2 style



*Dept. of Software· Contents Technology Research
Computer & Software Laboratory
Electronics and Telecommunications Research Institute*

You-Hee Choi (yhchoi@etri.re.kr)

Organization

- **Introduction**
- **The C2 architectural style**
- **Our approach**
- **Case Study**
- **Conclusion**

Introduction (1/2)

- ▣ **In CBSD,**
 - ◇ The notion of building a system by writing code
 - has been replaced with building a system by assembling existing software components into multiple applications

- ▣ **The component model, Enterprise JavaBeans(EJB) follows the "Write Once, Run Anywhere" philosophy of the Java programming language**
 - ◇ But it is difficult to compose components provided by third parties without recompilation or a source code modification in the absence of mediators.
 - Due to the need of a direct method invocation from the standpoint of a client for communicating with other components.

- ➡ **The component composition approach is needed**
 - ◇ To compose various EJB components provided by third-parties and reconfigure flexibly an architecture through the plug-and-play technique.

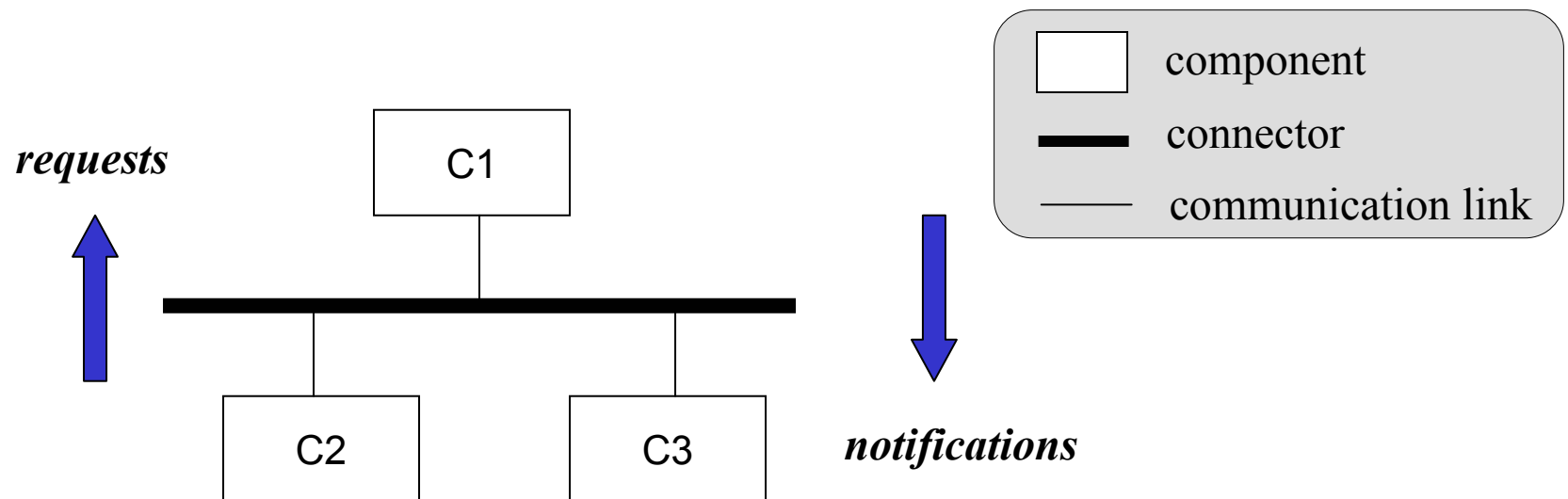
Introduction (2/2)

- ▣ **In order to compose EJB components through the plug-and-play technique,**
 - ◇ It is necessary to generate a mediator which handles mismatches of interfaces and interactions between components
 - Software architecture concept

- ▣ **For flexible reconfiguration when components are being composed,**
 - ◇ A communication method by an indirect message passing rather than a direct method invocation is required
 - The C2 architectural style

The C2 architectural style

▪ Message-based layered architectural style



- Two types of building blocks : components and connectors
- Connectors act as message routing devices.
- Components and connectors both have a defined top and bottom.

Our approach

□ Component composition approach for EJB

◇ Purpose

- To create a composite EJB for a C2 architecture which is composed of EJB components provided by third parties through the plug-and-play technique
- To compose EJB components without modification through the plug-and-play technique based on the C2 style,
 - Direct method invocations of EJB components should be replaced with indirect message passing
 - » Generate an EJB wrapper class by a wrapping mechanism, which has the logic of message passing instead of method invocations

EJB wrapper for EJB composition

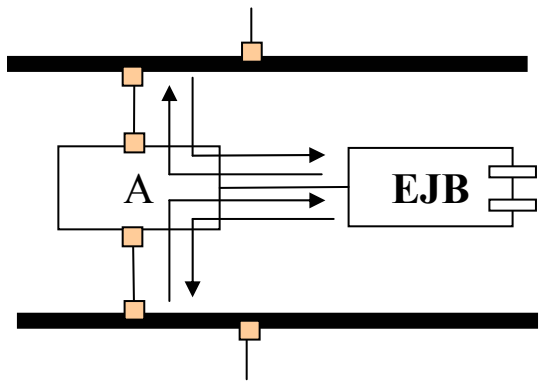
▣ The role of EJB wrappers

- ◆ To communicate with components
 - By the logic of message passing instead of the logic of method invocation
- ◆ To invoke methods of EJB components according to received messages

- ◆ The logic of message passing
 - Enables to send a message to the target component which is not explicitly specified, and to convert the message to the message the target component requires.

EJB wrapper for EJB composition

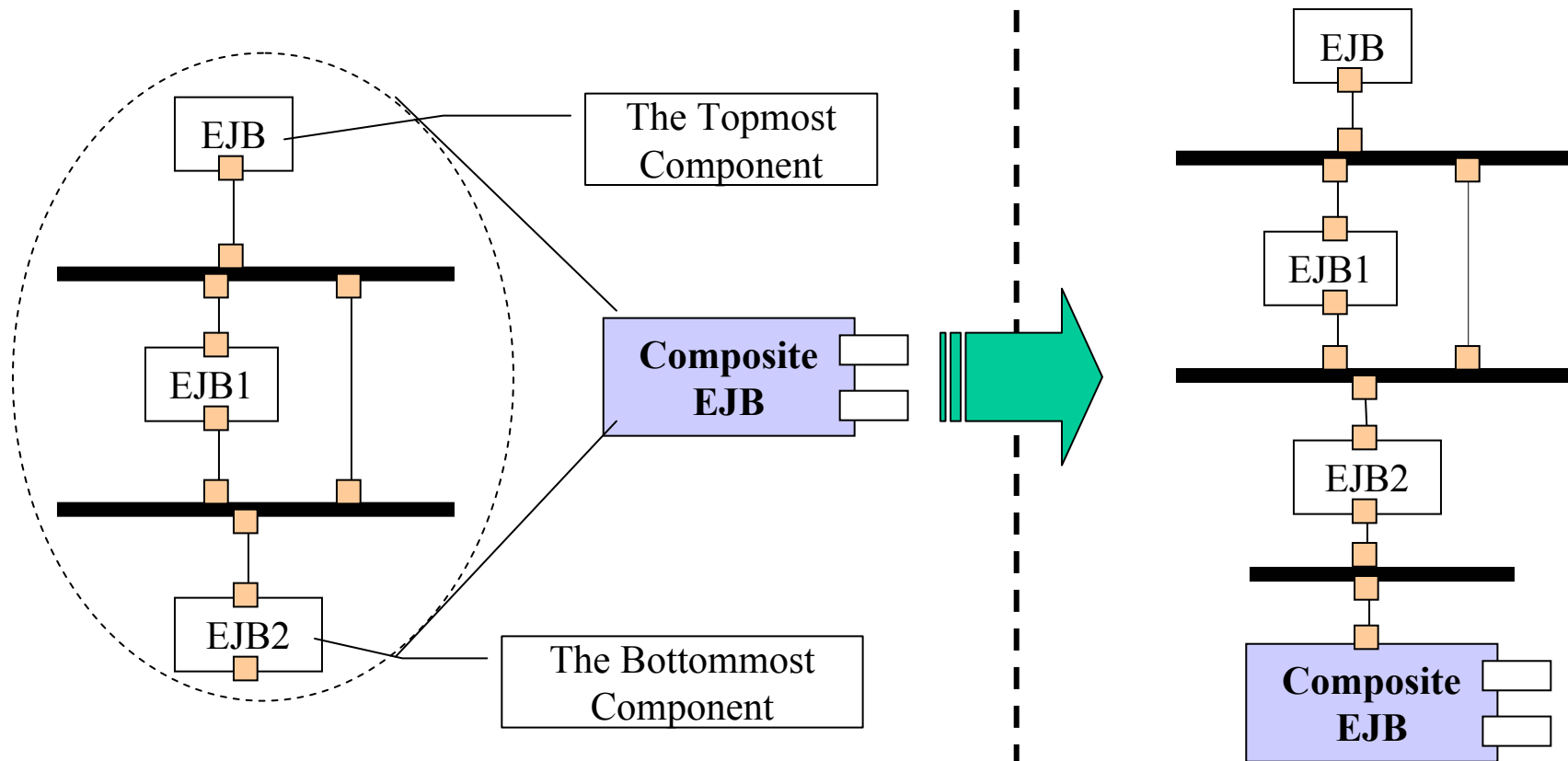
- The representative form of the message handling logic



```
if (message_received)
{
    //get parameters
    .....
    //invoke methods of EJB
    .....
    //create requests/notifications
    .....
    //send messages
    .....
}
```

Generating a composite EJB component

- The relationship of the C2 architecture which is composed of various EJB components and a composite EJB.



Generating a composite EJB component

- **A new C2 component which links the C2 architecture with the composite EJB is needed**
 - ◆ The role of initializing the C2 architecture to send messages, creating and sending requests, and receiving notifications

BeanGlueCode
private Object return_value
init(); remoteMethod1();
handle(Message n);

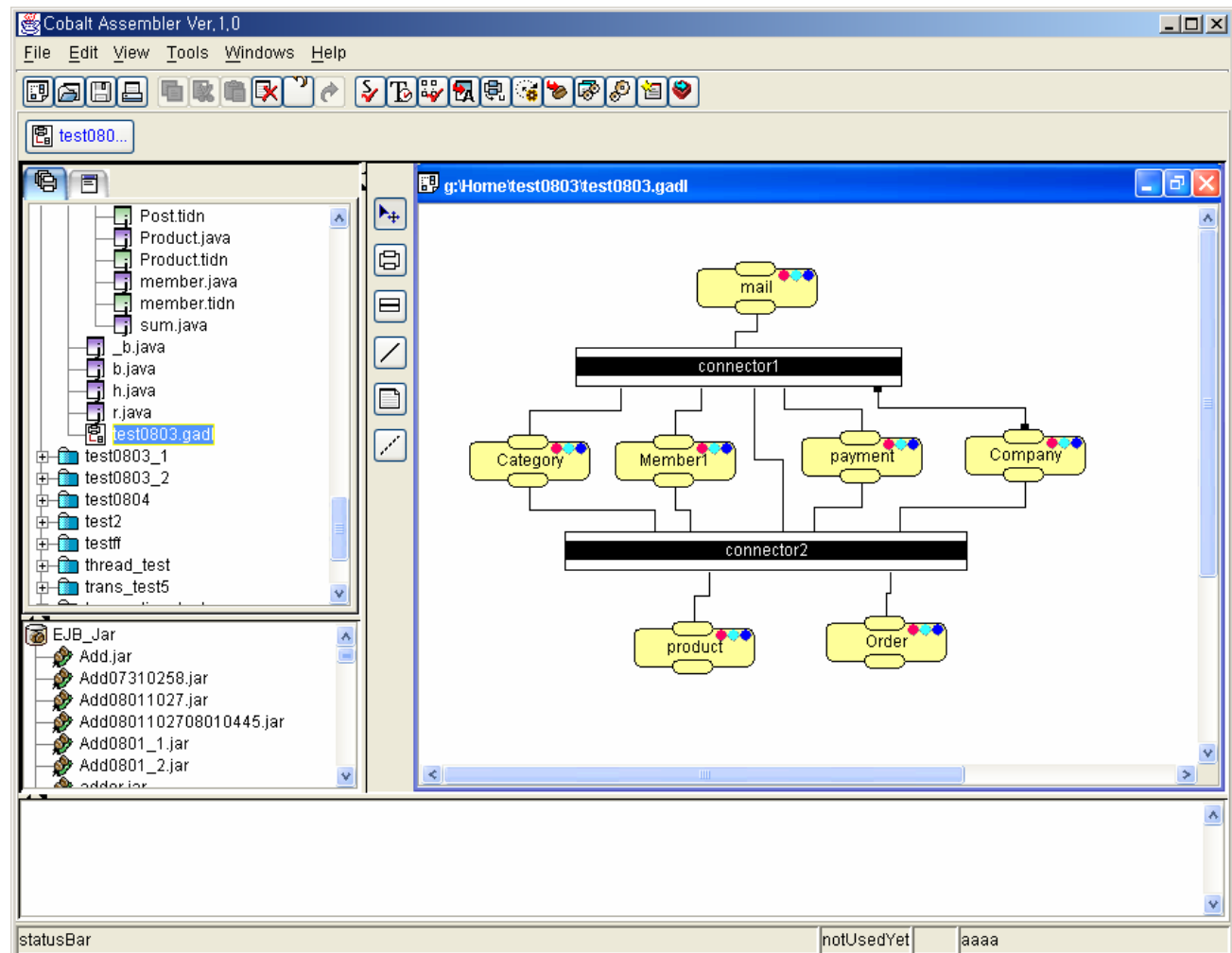
```
//remote method
protected Object remoteMethod1() {
    .....
    Request new_r = new Request ("request1");
    .....
    send (new_r);
    return return_value;
}
protected void handle(Message n) {
    // handle message
}
```

Case Study

- ▣ **A simple example using the component-assembly tool called COBALT (COmponent-Based Application deveLopment Tool) Assembler that has been developed for our approach**
 - ◇ The tool
 - Enables component reusers to assembly EJB components through the plug-and-play technique
 - Generates EJB wrappers and a composite EJB automatically based on component specifications described through a specification editor
 - ◇ The example system
 - Is a simple Internet shopping mall system

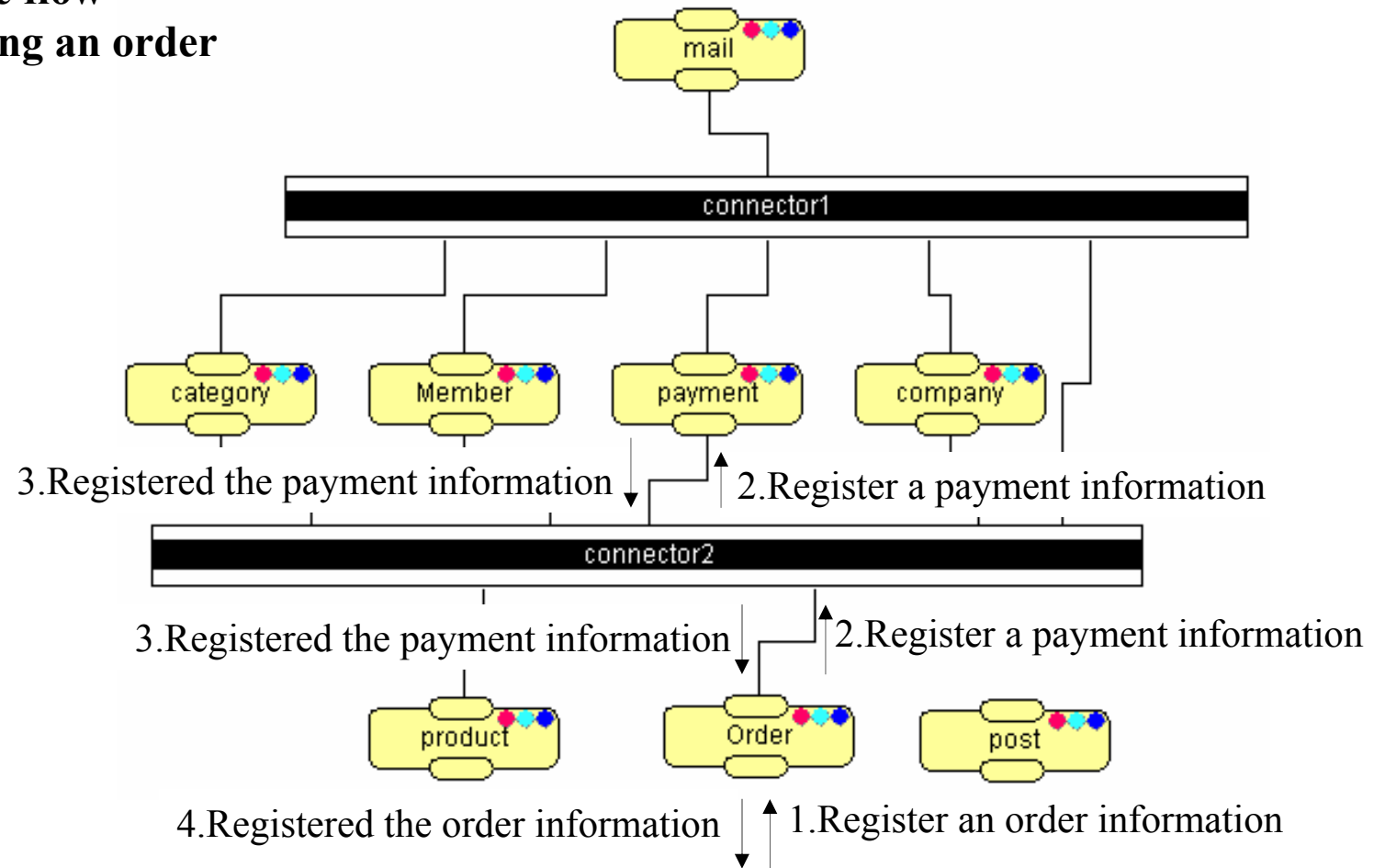
Case Study

- The architecture that is composed of 8 EJB components



Case Study

- The message flow for registering an order



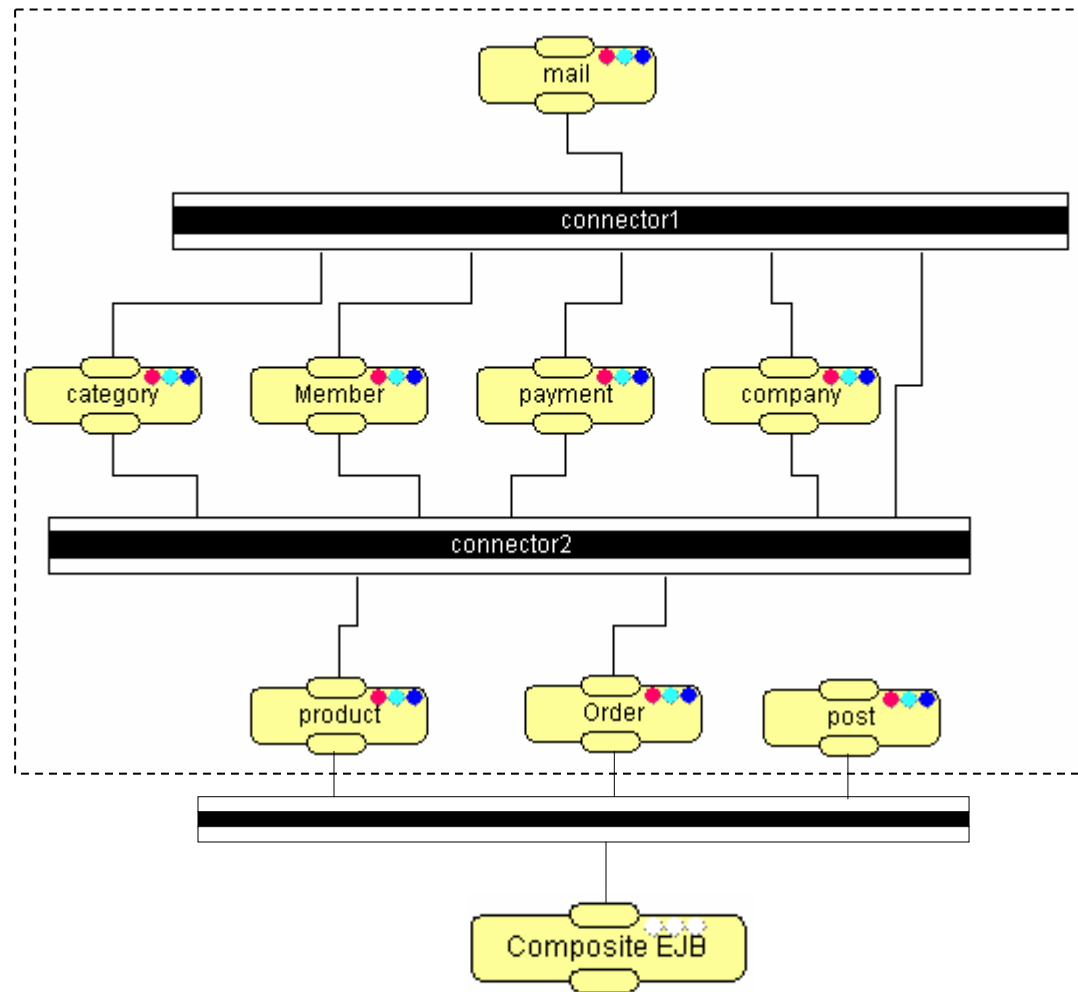
Case Study

```
g:\Home\app_test2\comp_spec\Order.java
38 message_name = r.getName();
39 if(message_name.equalsIgnoreCase ("registerBuyerOrder(etri.shoppingmall.
40 {
41     try
42     {
43         etri.shoppingmall.order.model.OrderInfo par_OrderInfo oInfo;
44         etri.shoppingmall.payment.model.PaymentInfo par_Pay
45         boolean par_boolean_registerOrderResult;
46
47         par_OrderInfo_oInfo =(etri.shoppingmall.order.mode
48         par_PaymentInfo_pInfo =(etri.shoppingmall.payment.
49         par_boolean_registerOrderResult = _OrderSession.re
50
51         if(par_boolean_registerOrderResult)
52         {
53             Message new_0 = new Message ("Request", "regist
54             new_0.addParameter ("pInfo", par_PaymentInfo_pI
55             send(new_0);
56         }
57     }
58     catch (Exception e)

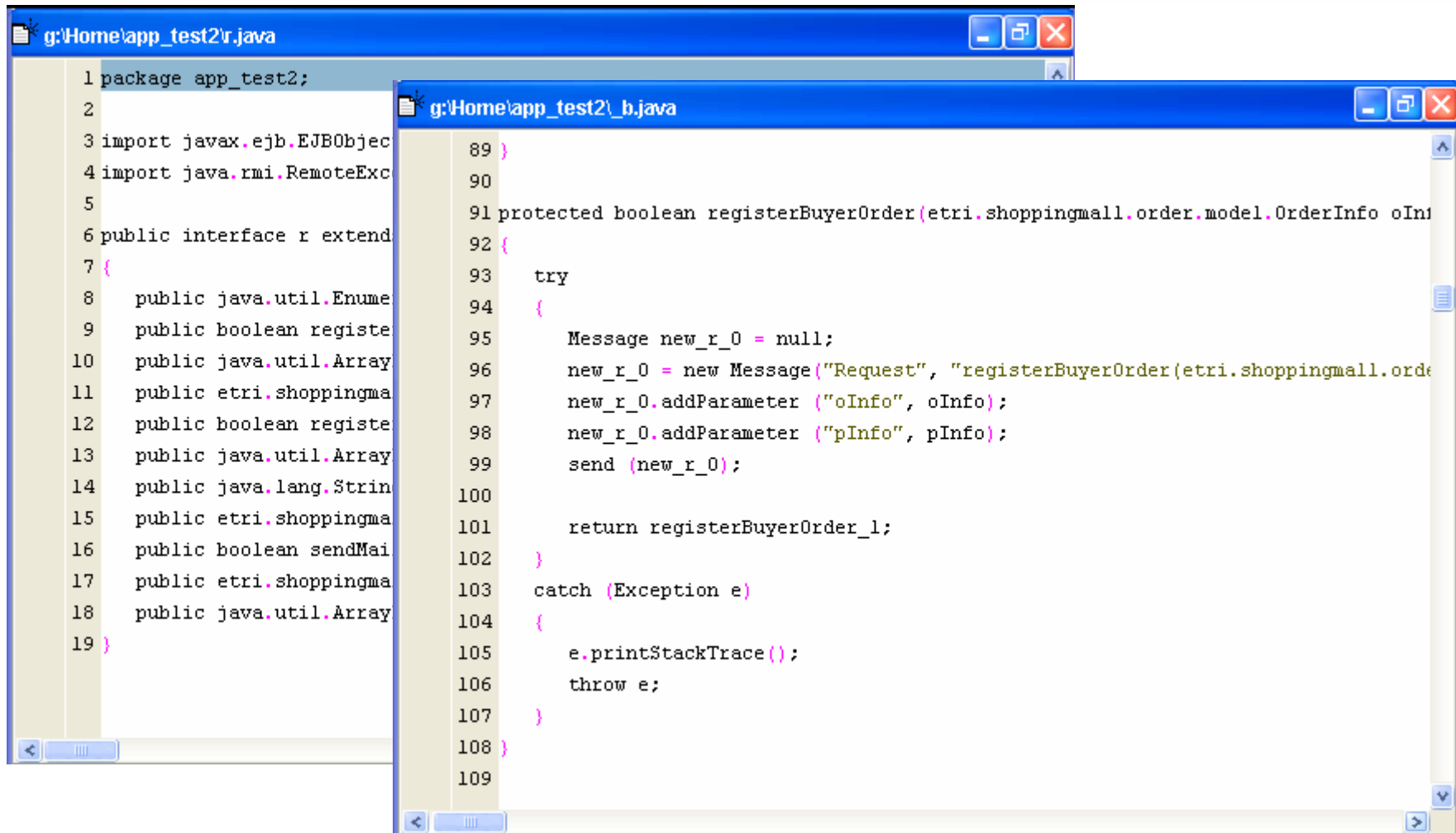
g:\Home\app_test2\comp_spec\Payment.java
30
31 String message_name;
32 message_name = r.getName();
33 if(message_name.equalsIgnoreCase ("registerPayment(etri.shoppingmall.pay
34 {
35     try
36     {
37         etri.shoppingmall.payment.model.PaymentInfo par_PaymentInfo_pInfo;
38         boolean par_boolean_registerPaymentResult;
39
40         par_PaymentInfo_pInfo =(etri.shoppingmall.payment.model.PaymentInf
41         par_boolean_registerPaymentResult = _PaymentSession.registerPaymen
42
43         Message new_0 = new Message ("Notification", "registeredPay(boolea
44         new_0.addParameter ("isRegPayVal", par_boolean_registerPaymentResu
45         send(new_0);
46     }
47     catch (Exception e)
48     {
49         e.printStackTrace();
50         throw e;
```

Case Study

- Generating a composite EJB



Case Study



```
g:\Home\app_test2\r.java
1 package app_test2;
2
3 import javax.ejb.EJBObject;
4 import java.rmi.RemoteException;
5
6 public interface r extends Remote {
7 {
8     public java.util.Enumeration<String> getAttributes();
9     public boolean registerBuyerOrder(etri.shoppingmall.order.model.OrderInfo oInfo, etri.shoppingmall.order.model.PaymentInfo pInfo);
10    public java.util.ArrayList<String> getAttributes();
11    public etri.shoppingmall.order.model.OrderInfo getoInfo();
12    public boolean registerBuyerOrder(etri.shoppingmall.order.model.OrderInfo oInfo, etri.shoppingmall.order.model.PaymentInfo pInfo);
13    public java.util.ArrayList<String> getAttributes();
14    public java.lang.String getAttributes();
15    public etri.shoppingmall.order.model.OrderInfo getoInfo();
16    public boolean sendMail(etri.shoppingmall.order.model.OrderInfo oInfo, etri.shoppingmall.order.model.PaymentInfo pInfo);
17    public etri.shoppingmall.order.model.OrderInfo getoInfo();
18    public java.util.ArrayList<String> getAttributes();
19 }
```

```
g:\Home\app_test2\b.java
89 }
90
91 protected boolean registerBuyerOrder(etri.shoppingmall.order.model.OrderInfo oInfo, etri.shoppingmall.order.model.PaymentInfo pInfo) {
92 {
93     try
94     {
95         Message new_r_0 = null;
96         new_r_0 = new Message("Request", "registerBuyerOrder(etri.shoppingmall.order.model.OrderInfo oInfo, etri.shoppingmall.order.model.PaymentInfo pInfo)");
97         new_r_0.addParameter("oInfo", oInfo);
98         new_r_0.addParameter("pInfo", pInfo);
99         send(new_r_0);
100
101         return registerBuyerOrder_1;
102     }
103     catch (Exception e)
104     {
105         e.printStackTrace();
106         throw e;
107     }
108 }
109 }
```

Conclusions

- ▣ **An approach to composition of EJB components based on the C2 style**
 - ◇ By generating a composite EJB
 - Allows component reusers to build a new function by composing prebuilt EJB components
 - ◇ By generating an EJB wrapper
 - Various EJB components can be composed without recompilation or source code modification
- ▣ **Our research work includes**
 - ◇ Redefinition of C2 ADL(Architecture Description Language) for EJB
 - ◇ The development of EJB component-assembly tool called COBALT (COmponent-Based Application deveLopment Tool) assembler that is the visual composition tool to support our approach.