Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 4
Remote Procedure Calls and Distributed Transactions

Outline

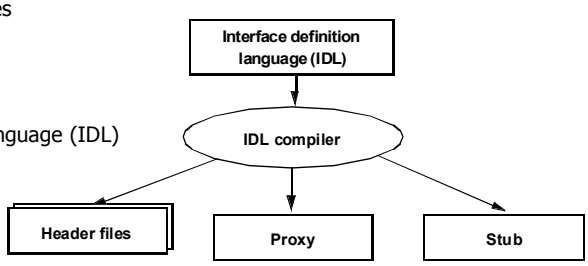
- Remote Procedure Call
  - concepts
    - IDL, principles, binding
  - variations
    - remote method invocation
      - example: Java RMI
    - stored procedures
- Distributed Transaction Processing
  - distributed TAs, 2PC
  - transactional RPC
  - X/Open DTP
- Summary

Communication and Distributed Processing

- Distributed (Information) System
  - consists of (possibly autonomous) subsystems
  - jointly working in a coordinated manner
- How do subsystems communicate?
  - **Remote Procedure Calls (RPC)**
    - transparently invoke procedures located on other machines
  - Peer-To-Peer-Messaging
  - Message Queuing
- Transactional Support (ACID properties) for distributed processing
  - Server/system components are Resource Managers
  - (Transactional) Remote Procedure Calls (TRPC)
  - Distributed Transaction Processing

Remote Procedure Call (RPC)

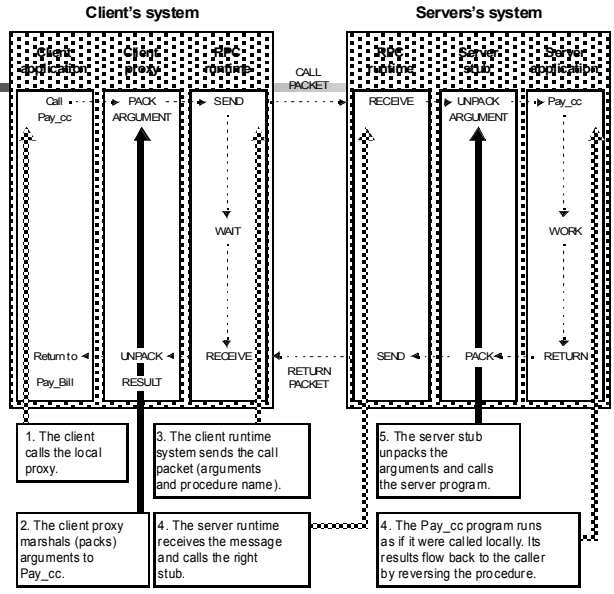
- Goal: Simple programming model for distributed applications
  - based on procedure as an invocation mechanism for distributed components
- Core mechanism in almost every form of middleware
- Distributed programs can interact (transparently) in heterogeneous environments
  - network protocols
  - programming languages
  - operating systems
  - hardware platforms
- Important concepts
  - Interface Definition Language (IDL)
  - Proxy (Client Stub)
  - Stub (Server Stub)



How RPC Works

- Define an interface for the remote procedure using an IDL
  - abstract representation of procedure
    - input and output parameters
  - can be independent of programming languages
- Compile the interface using IDL-compiler, resulting in
  - client stub (proxy)
  - server stub
  - auxiliary files (header files, …)
- Client stub (proxy)
  - compiled and linked with client program
  - client program invokes remote procedure by invoking the (local) client stub
  - implements everything to interact with the server remotely
- Server stub
  - implements the server portion of the invocation
  - compiled and linked with server code
  - calls the actual procedure implemented at the server

RPC



Binding in RPC

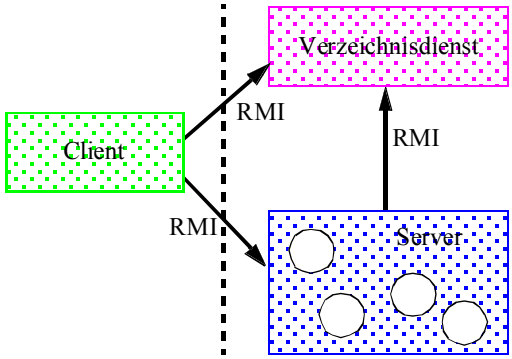
- Before performing RPC, the client must first locate and *bind* to the server
  - create/obtain an (environment-specific) *handle* to the server
  - encapsulates information such as IP address, port number, Ethernet address, …
- Static binding
  - handle is "hard-coded" into the client stub at compile-time
  - advantages: simple and efficient
  - disadvantages: client and server are tightly coupled
    - server location change requires recompilation
    - dynamic load balancing across multiple (redundant) servers is not possible
- Dynamic binding
  - utilizes a name and directory server
    - based on logical names, signatures of procedures
    - server registers available procedure with the N&D server
    - client asks for server handle, uses it to perform RPC
    - requires lookup protocol/API
  - maybe performed inside the client stub (automatic binding) or outside
  - opportunities for load balancing, more sophisticated selection (traders)

Variation 1: Distributed Objects

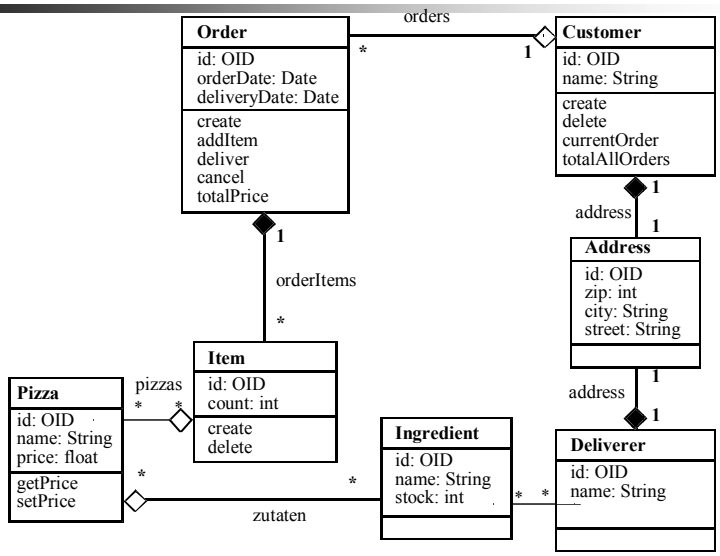
- Basic Idea: Evolve RPC concept for objects
  - application consists of distributed object components
  - object services are invoked using Remote Method Invocation (RMI)
- Utilizes/matches advantages of object-oriented computing
  - object identity
  - encapsulation: object manipulated only through methods
  - inheritance, polymorphism
  - interface vs. implementation
  - reusability

Distributed Objects with Java RMI

- Mechanism for communication
  - between Java programs
  - between Java programs and applets
- Capabilities
  - finding remote objects
  - transparent communication with remote objects
  - loading byte code for remote objects

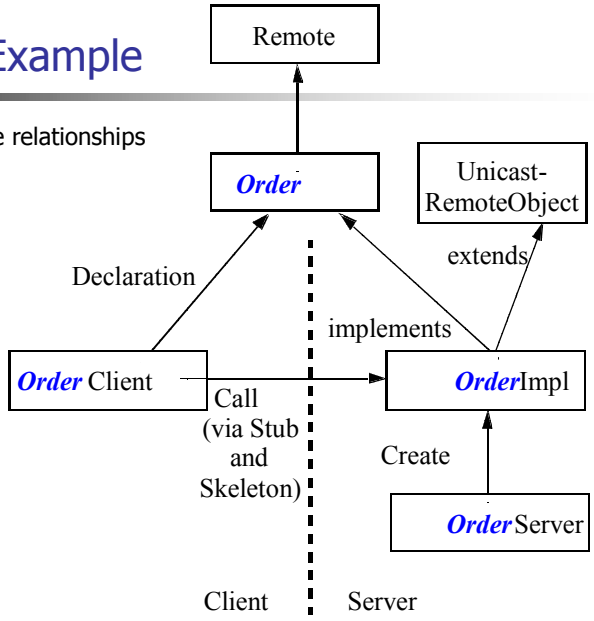


Example Scenario: Pizza-Service



Java RMI - Example

- Class and interface relationships



Java RMI – Example (continued)

```
import java.rmi.*;
import java.util.Date;
public interface Order extends Remote {
        public void addItem(int pizzaId, int number)
                                        throws RemoteException;
        public Date getDeliveryDate() throws
                        RemoteException;
        public Date setDeliveryDate (Date newDate) throws
        RemoteException;
        }


...
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;
```

Java RMI - Example (continued)

```
...
public class OrderImpl
        extends UnicastRemoteObject
        implements Order {
                private Vector fItems;
                private Date fDeliveryDate;
                public OrderImpl(String name) throws RemoteException {
                  super();
                  try {
                     Naming.rebind(name, this); // Register with Nameserver
                     fItems = new Vector();
                     fDeliveryDate = null;
                      }
                  catch (Exception e) {
                     System.err.println("Output: " + e.getMessage());
                     e.printStackTrace();
                                }
                        }
...
```


Java RMI - Example (continued)

```
...
public void addItem(int pizzaId, int number )
                throws RemoteException {
                // assuming class Item is known
                Item item = new  Item(pizzaId, number);
                fItems.addElement(item);
                        }
                ... // Impl. of other methods          }
```

Java RMI - Example (continued)

```
...
import java.rmi.*;
public class OrderClient {
        public static void Main(String args[]) {
        try {
                Order order = (Order)
                    Naming.lookup("rmi://berlin:9000/my_order");
                int pizzaId = Integer.parseInt(args[0]);
                int number = Integer.parseInt(args[1]);
                order.addItem(pizzaId, number);
                        }
        catch (Exception e) {
                System.err.println("system error: " + e);
                        }
                }
        }
```


Java RMI - Example (continued)

```
...
import java.rmi.*;
import java.server.*;
public class OrderServer {
        public static void main(String args[]) {
          try {
             OrderImpl order = new OrderImpl("my_order");
             System.out.println("Order server is running");
                }
          catch (Exception e) {
              System.err.println("Exception: " + e.getMessage());
              e.printStackTrace();
                        }
                }
        }
```

Java RMI - Example (continued)

- Compile:
  javac Bestellung.java BestellungImpl.java BestellungClient. java BestellungServer.java
- Generate stub and skeleton code:
  rmic BestellungImpl
- administrative steps:
  - Start directory server: ***rmiregistry***
  - Start RMI-Servers: ***java BestellungServer***
  - Run clients: ***java BestellungClient***

Variation 2: Stored Procedures

- Named persistent code to be invoked in SQL, executed by the DBMS
  - SQL *CALL* statement
    - RPC is not transparent!
- Created directly in a schema or in a SQL-server module
- Have a header and a body
  - Header consists of a name and a (possibly empty) list of parameters.
    - may specify parameter mode: IN, OUT, INOUT
- SQL routines
  - Both header and body specified in SQL
- External routines
  - Header specified in SQL
  - Bodies written in a host programming language
    - May contain SQL by embedding SQL statements in host language programs or using CLI

RPCs and Transactions

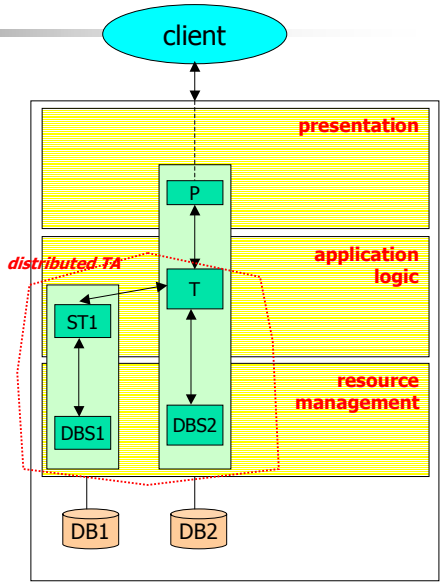
- Example scenario for T: debit/credit
  - T invokes debit procedure (ST1), modifying DB1
  - T performs credit operation on DBS2, modifying DB2
- Need transactional guarantees for T
- Program structure of T

  BOT

  CALL debit(…)

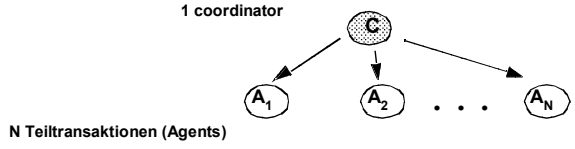
  CONNECT (DB2)
  UPDATE ACCOUNTS SET …
  DISCONNECT

  EOT
- Requires coordination of distributed transaction



Distributed Transactions

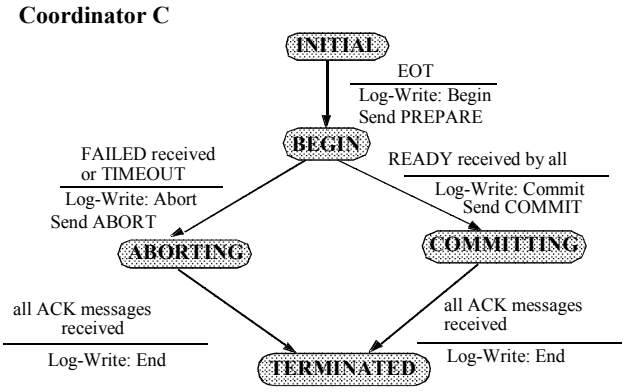
- Require global (multi-phase) commit protocol
  - guaranteed atomicity of global TA
  - requirements for commit protocol
    - minimal effort (#messages, #log entries)
    - minimal response delay (parallelism)
    - robustness against failure
  - expected failure
    - partial failure (connection loss, …)
    - transaction failure
    - system failure (crash)
    - hardware failure
  - failure detection (e.g., using time-out)



Two-phase Commit

- Prepare-Phase, Commit/Abort-Phase
- Requires sequence of state transitions, to be safely stored in the transaction log

Coordinator View

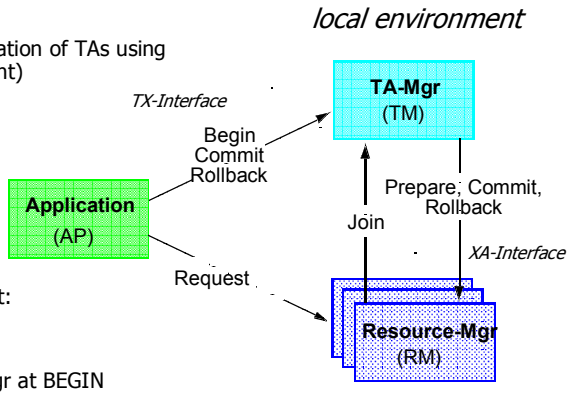


Transactional RPC (TRPC)

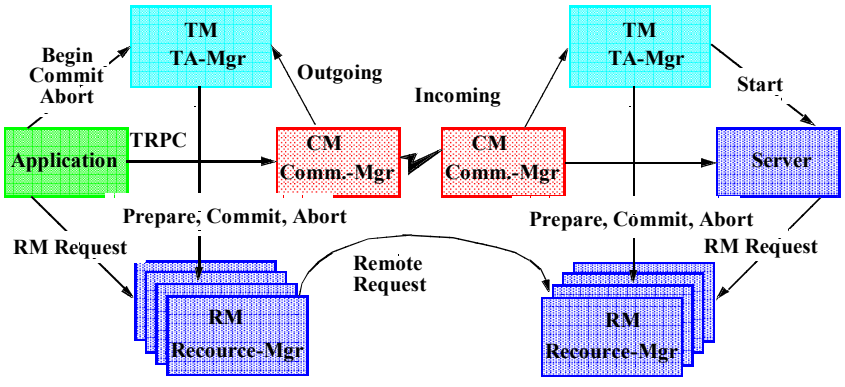
- Servers are resource managers
- RPCs are issued in the context of a transaction
  - demarcation (BOT, EOT) usually happens on the client
- TRPC-Stub
  - like RPC-Stub
  - additional responsibilities for TA-oriented communication
- TRPC requires the following additional steps
  - binding of RPC to transactions using TRID
  - notifying TA-Mgr about RM-Calls if performed through RPC (register participant of TA)
  - binding processes to transactions: failures (crashes) resulting in process termination should be communicated to the TA-Mgr

X/OPEN – Standard for Distributed TA Processing

- Resource Manager
  - recoverable
  - supports external coordination of TAs using 2PC protocol (XA-compliant)
- TA-Mgr
  - coordinates, controls RMs
- Application Program
  - demarcates TA (TA-brackets)
  - invokes RM services
    - e.g., SQL-statements
  - in distributed environment: preforms (T)RPCs
- Transactional Context
  - TRID generated by TA-Mgr at BEGIN
  - established at the client
  - passed along (transitively) with RM-requests, RPCs



X/OPEN DTP – Distributed Environment



Summary

- Remote Procedure Call
  - importance core concept for distributed IS
  - RPC model is based on
    - Interface definitions using IDL
    - Client stub (proxy), Server Stub for transparent invocation of remote procedure
    - Binding mechanism
- RPC Variations
  - Remote Method Invocation
    - supported in object-based middleware (e.g., CORBA, Enterprise Java)
  - Stored Procedures
- Transaction support for RPCs
  - distributed transaction processing guarantees atomicity of global TA
  - 2PC
  - transaction RPC
  - X/Open DTP as foundation for standardized DTP
    - variations/enhancements appear in object-based middleware (CORBA OTS, Java JTA/JTS)