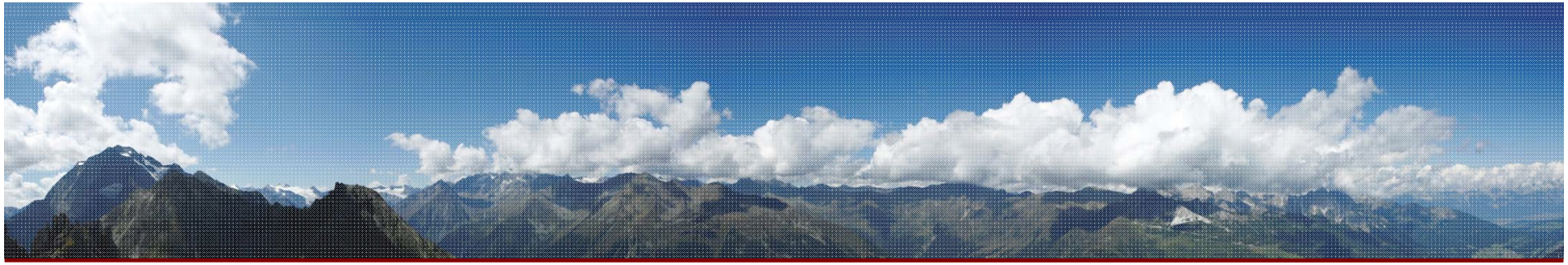
Web Services

Middleware



Where are we?

STI · INNSBRUCK

#	Title
1	Distributed Information Systems
2	**Middleware**
3	Web Technologies
4	Web Services
5	Basic Web Service Technologies
6	Web 2.0 Services
7	Web Service Security

Outline

STI · INNSBRUCK

- Motivation
- Technical solution
  - Understanding Middleware
  - Remote Procedure Call (RPC) and Related Middleware
  - Transaction Processing (TP) Monitors
  - Object Brokers
  - Message-Oriented Middleware
- Illustration by a larger example
- Summary
- Resources

Motivation

Motivation

STI · INNSBRUCK

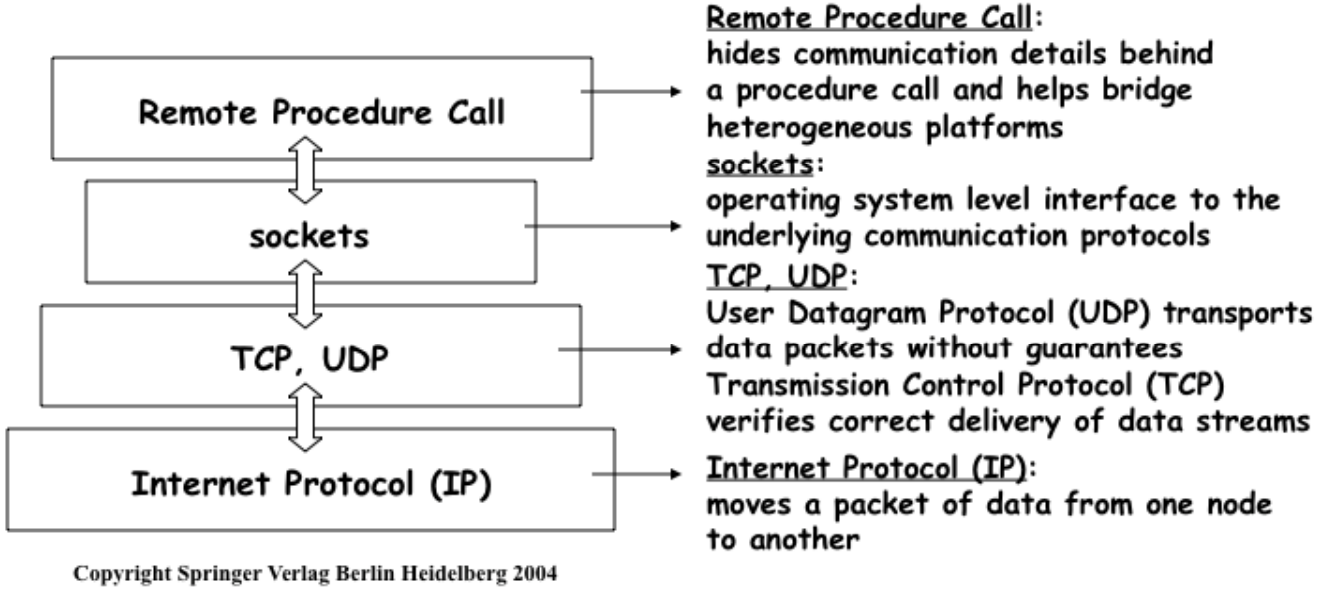
- Efficient solutions are needed to enable seamless integration of servers and applications (2-tier, 3-tier, and N-tier architectures).
- The problems encountered in the context are reoccurring
  - Synchronous and asynchronous communication between the distributed nodes.
  - Transactional guarantees over the distributed resources.
- Solutions to the problems can be useful to many different users.
- Conventional ***middleware*** platforms are providing well developed, standardized, robust and tested solutions to the reoccurring problems.
  - Used in restricted settings like LANs or over a collection of systems which are physically close.

Technical Solution

Understanding Middleware

STI · INNSBRUCK

- Middleware hides some of the complexities of building a distributed application
  - Programmer can focus on business-related functionality.
  - The functionality of middleware should otherwise be developed from scratch.

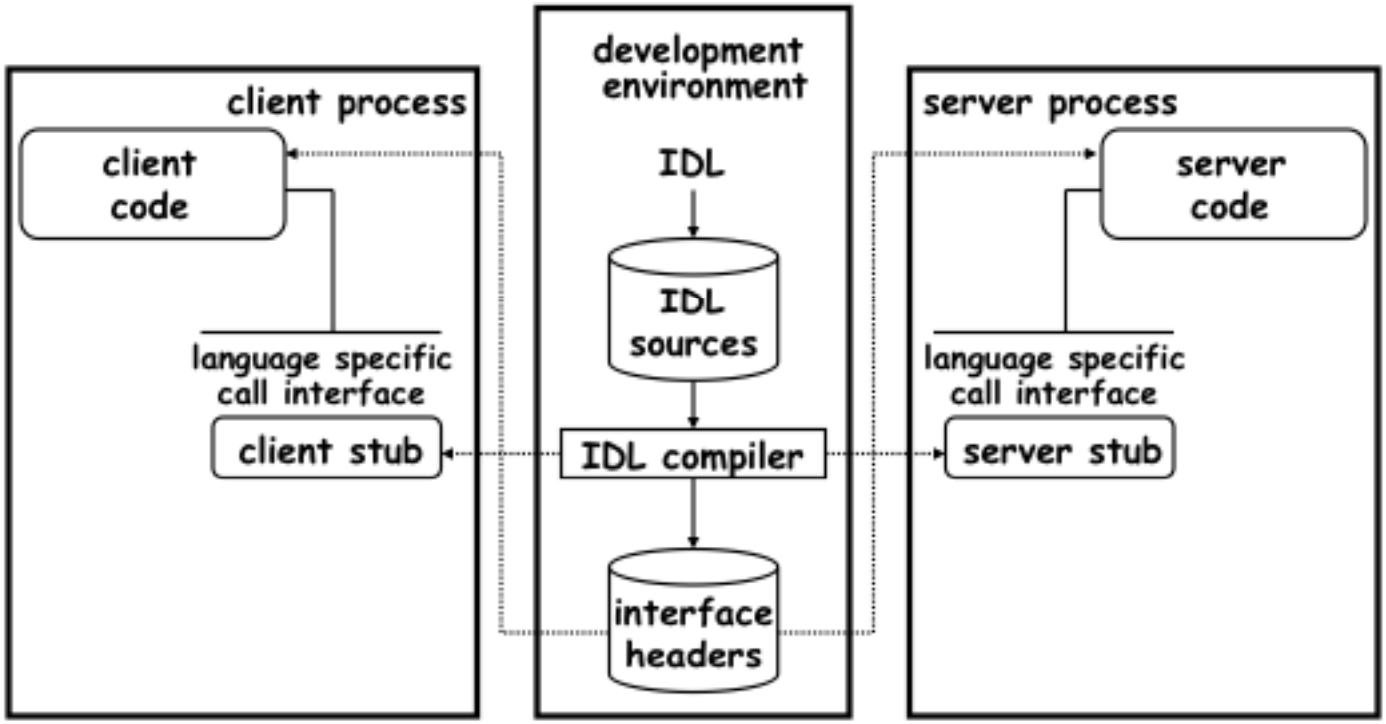


- Infrastructure is needed to implement programming abstractions.
  - It usually has a large footprint – complex software systems.
- Simplest form of RPC requires IDL, compiler and supporting software libraries.
  - Development vs. run-time parts of infrastructure.
- Extensions and enhancements make programming abstractions more expressive, flexible and easier to use
  - But also make systems more complex and the learning curve steeper.
  - E.g., authentication, dynamic binding support, etc.
- Components may get more complex interfaces, allowing direct access to low-level details.

Technical Solution

Remote Procedure Call (RPC) and Related Middleware

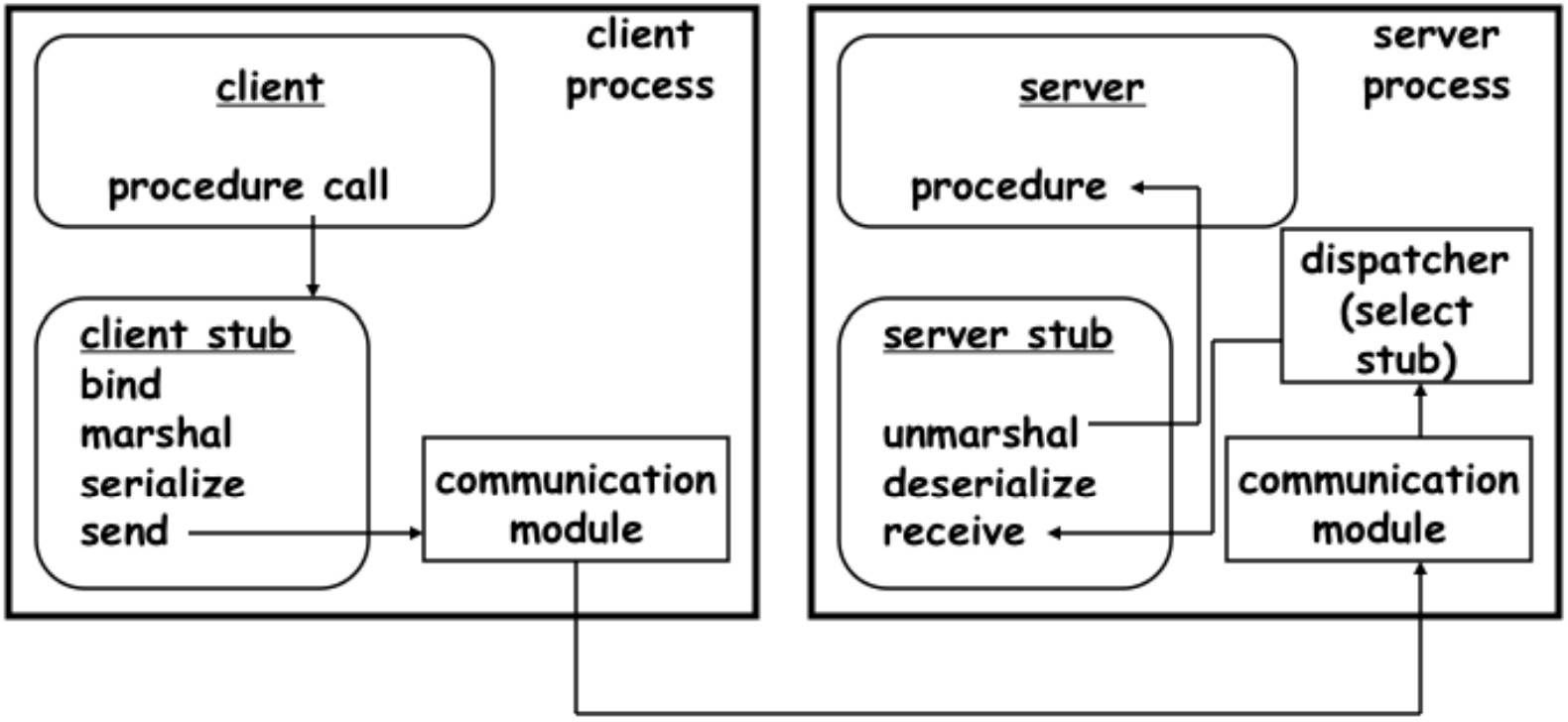
- Introduced at the beginning of 1980s by Birell and Nelson.
- RPC was introduced as a way to transparently call procedures located on other machines.
- It became the basis for building 2-tier systems and it established various notions:
  - Client/Server computing,
  - Interface Definition Language (IDL),
  - Name and Directory Services,
  - Dynamic Binding, etc.
- RPC is a clean way to deal with the distribution
  - Relies on the concept of procedure
  - No need to change the programming language or paradigm to create distributed applications.
- Whether RPC should be transparent or not?
  - Simplicity vs. changing of program nature (functional and non-functional concerns)



RPC has a clearly defined methodology:

1. Defining the interface for the remote procedure
   - Relies on Interface Definition Language (IDL) – abstract representation of the procedure
   - Specification of services provided by the server.
2. Compiling IDL description by using interface compiler to produce
   - Client stubs
     - Piece of code to be compiled and linked with the client.
     - Client makes actually local call to the stub.
     - Stub locates server, formats the data (marshaling and serialization), communicates to server and provides the response as the procedure response.
   - Server stubs
     - Similar in nature to client stubs.
     - Implementing the server side invocation.
   - Code templates and references
     - Auxiliary files – headers, code templates, etc.

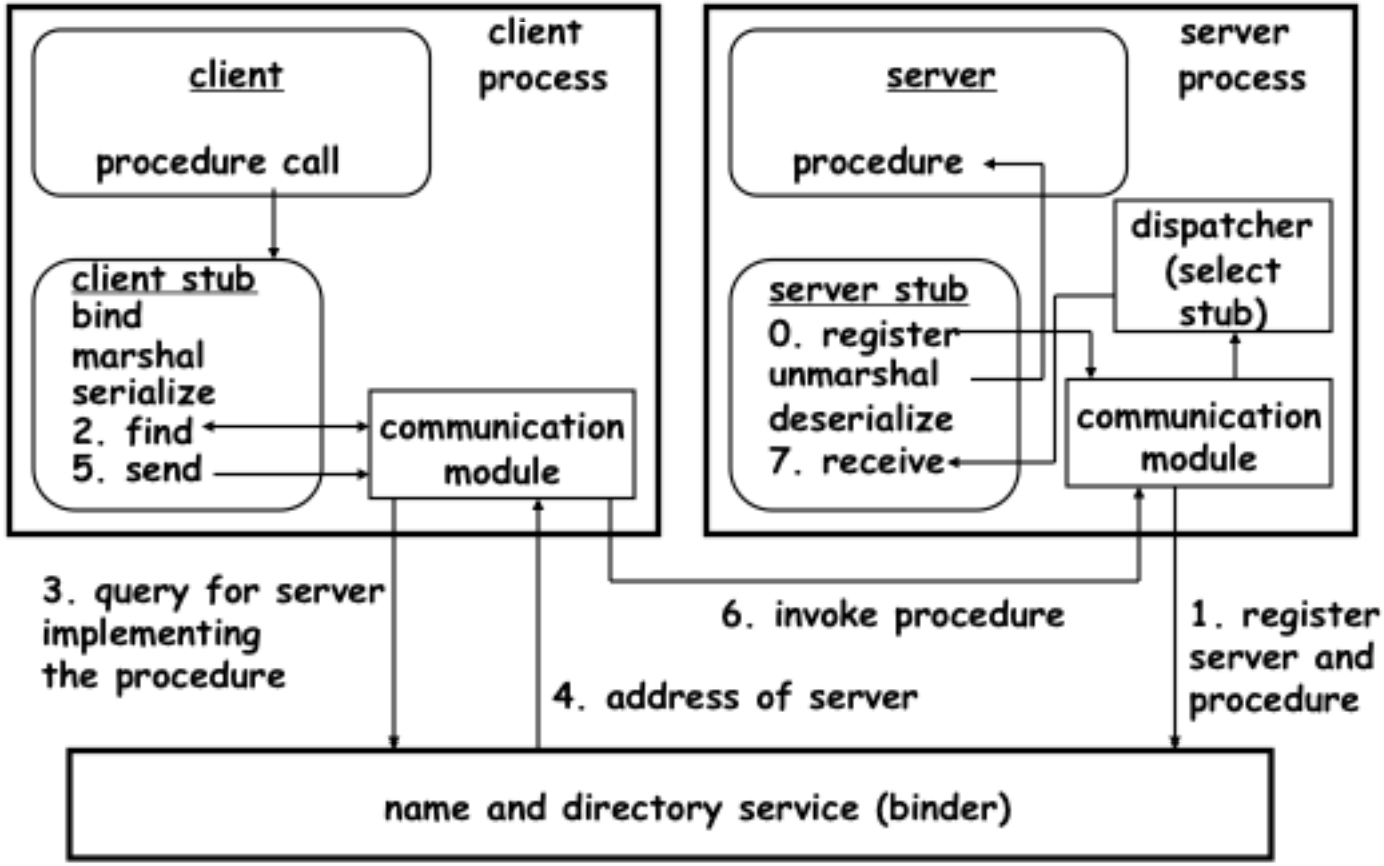
- Stubs are handling all of the network programming details
  - Different interfaces can give different level of detail exposure.



STI · INNSBRUCK

- Binding is the process during which a client creates a local association for a given server in order to invoke a remote procedure.

- The process can be
  - Static
    - Hardcoded reference to the server.
    - Simple and effective, no additional infrastructure is required.
    - Client and Server are becoming tightly coupled (e.g., failures, relocations and load balancing are problematic).
  - Dynamic
    - Relies on specialized services to locate servers (additional layer of indirection).
    - Usually called Name and Directory Service or Binder Service.
    - Client stub communicates to binder to discover suitable service
      - May implement load balancing, relocation of services
    - Cost is in additional infrastructure, protocol and registration primitives.
      - The complexities may be reduced by stubs.
    - Binding can be based on method signature but also based on non-functional properties (traders).



- Push towards 2-tier systems made systems heterogeneous.
- RPC is a mechanism to bridge heterogeneous systems
  - Client and server platforms may be different.
- Naïve approach is to make all possible combinations of C/S stubs
- More efficient approach is to use some intermediate representation
  - Client and server stubs need to know how to translate between their native and the intermediate representation
- IDL can be used to define such a mapping from concrete to intermediate representation
  - We can ignore differences in terms of machine architecture, programming language and used type system.
  - Used to define the exchange data representation, parameter values, etc.

- RPC is inherently a call to a local function in a program
  - synchronous, sequential, blocking for a client.
  - threading model employed at the server side.

- Asynchronous RPC
  - Basis for Message-Oriented Middleware and Message Brokers
  - Non-blocking at the client side
  - Client stub provides two entry points – to invoke a procedure and to retrieve the result
    - Result retrieval can generate error (result not ready!!!) or failure code (server side problem).
  - Stubs basically execute synchronously while client has an illusion of async behavior.
  - Much more sophisticated infrastructure is needed than stubs
    - Recovery from failures is problematic, it requires some queuing which eventually lead to TP monitors and Message Brokers.

Technical Solution

Transaction Processing (TP) Monitor

STI · INNSBRUCK

- Transaction Processing (TP) Monitor
  - Oldest, well understood, well studied, tested, most efficient form of middleware.
- IBM Customer Information and Control System (CICS) 1960s and still in use today
  - Efficient multiplexing of resources among concurrent users of mainframes.
  - Relying on heavy usage of multithreading and data consistency which eventually lead to transactions.
- Developed to bypass the limitations of early operating systems
  - CICS can create and dispatch a thread faster than operating systems.
- TP-Lite vs. TP-Heavy
  - Lite has only a core functionality of TP Monitor as an additional layer over database management systems.
- Cornerstone of many N-tier systems nowadays
  - IBM CICS, Microsoft MTS, BEA Tuxedo, etc.

STI · INNSBRUCK

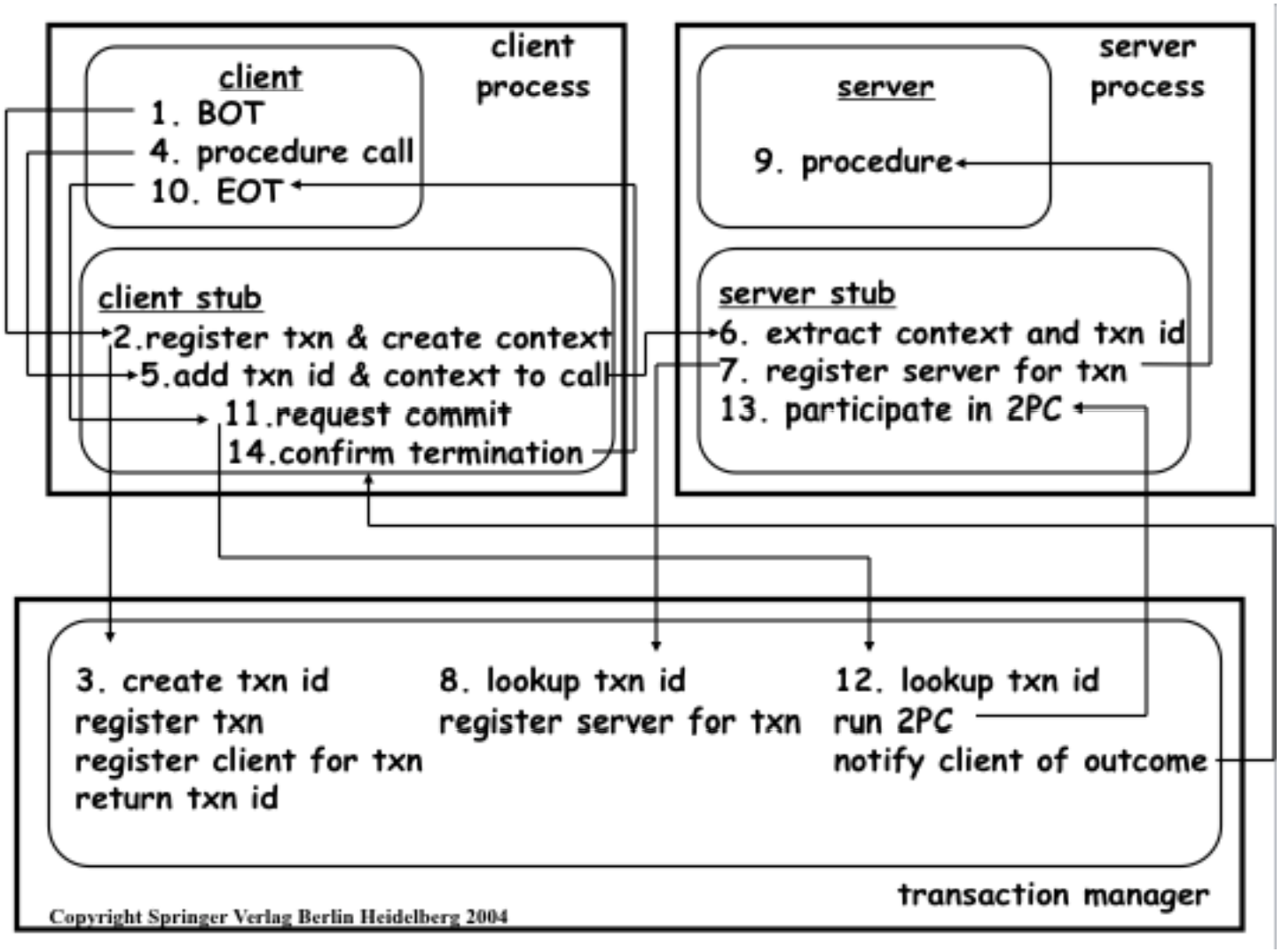
- TP Monitor supports the execution of distributed transactions.
- It is implemented as an abstraction layer over RPC called Transactional RPC (TRPC).
- RPC assumes independency between multiple calls even though they may be interrelated.
  - In case of partial system failures these interdependencies may be problematic.
  - Without any middleware support mechanism must be implemented by client itself.
- The solution is to extend RPC protocol to wrap a series of calls into a transaction.

STI · INNSBRUCK

- Concept of transaction is developed in the context of Database Management Systems (DBMS).
- Transaction represents a set of operations characterized by the so called ACID properties.
- TRPC provides a possibility to enforce ACID properties when dealing with data distributed across multiple (and heterogeneous) systems.
- Procedure calls enclosed within the transactional brackets are an atomic unit of work.
  - Beginning of Transaction (BOT) and End of Transaction (EOT) – RPC call.
  - Infrastructure guarantees their atomicity.
- Transaction Management module is responsible to coordinate interactions between clients and servers.

TP Monitor

Encapsulating an RPC call in transactional brackets



- Distributed transactions are done in a context of 2PC protocol
  - Commercially first used within CICS,
  - Standardized within X/Open specification.

- Commit is executed in context of two phases
  - Phase 1
    - Contacting each server involved in transaction by sending a **prepare to commit** message
    - If server successfully executed the TRPC procedure it answers with **ready to commit**.
    - In opposite server replies with **abort**.
  - Phase 2
    - Transaction manager examines all answers and decides what to do…
    - In case of all **ready to commit** answers it instructs each and every server to commit the local changes.
    - If at least one server responded with **abort** all servers are requested to roll back.

- Fault tolerance in 2PC is achieved through logging
  - Situation before the failure occurred can be reconstructed to recover the system.
  - It can have performance impacts.

- TP Monitor provides functionality necessary to develop, run, manage, and maintain transactional distributed systems which includes:
  - Basic RPC functionality
  - Programming abstractions to deal with TRPC (BOT, EOT, callback, commit, etc.)
  - Transactional Manager which includes logging, recovery, locking, etc.
  - Monitor systems for scheduling, assigning priorities, load balancing.
  - Run-time environment as a computational background for all the applications suing TPM
  - Specialized components such as protocol managers for interacting with different systems, and
  - Tools for installing, managing and monitoring the performance of all these components.

Technical Solution

Object Brokers

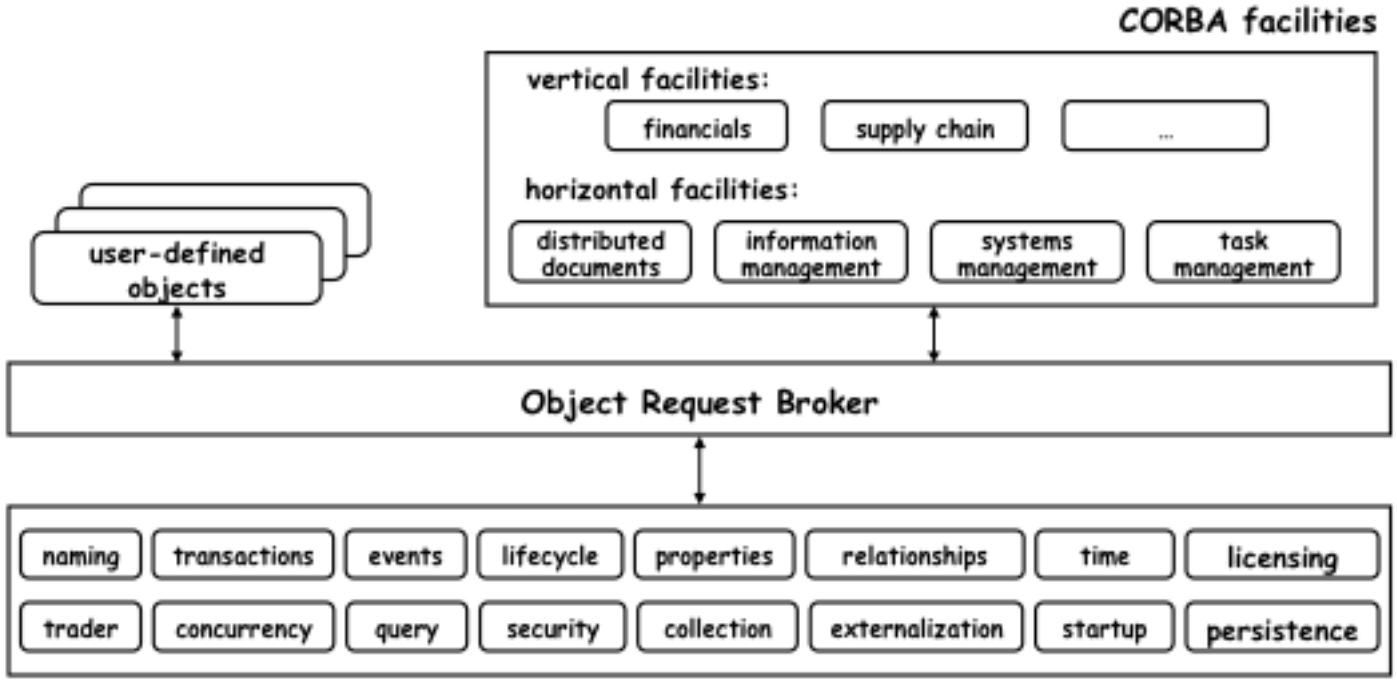
STI · INNSBRUCK

- Object Broker represents a middleware to support interoperability between objects.
- They appeared in 1990s as the natural evolution of RPC to cope with the increasingly popular object-oriented programming paradigm.
  - They also provide services that simplify the development of distributed OO applications.
- At the beginning they offered exactly the same functionality as RPC provides
  - Hiding abstraction of distributed calls.
  - Features of the object-oriented paradigm (inheritance and polymorphism) are making the process a bit more complex.
  - Later they have been added location transparency, dynamic binding, object lifecycle management, and persistence.

- Common Object Request Broker Architecture (CORBA) is the best-known example of object broker paradigm
  - Architecture and specification, no reference implementation.
  - It gained tremendous popularity in 1990s.
  - Developed in early 1990s by Object Management Group (OMG).
- Other popular solutions include:
  - Distributed COM by Microsoft
  - .NET by Microsoft
  - Java 2 Enterprise Edition (J2EE) by Sun Microsystems (now Oracle).
- OMG is trying to align CORBA with recent trends and to extend it with richer feature set provided by J2EE.

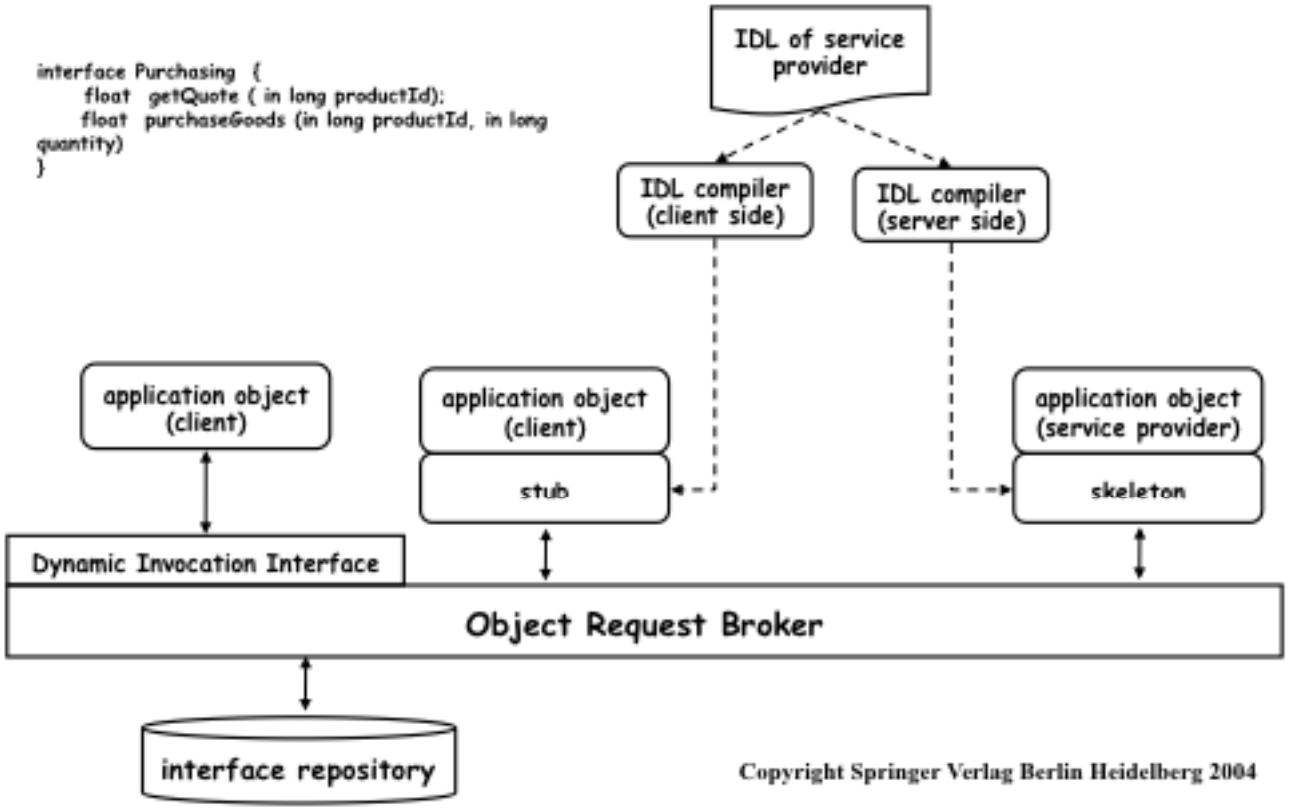
Object Brokers
System Architecture

STI · INNSBRUCK



- CORBA-based system is made of 3 parts:
  - Object Request Broker
    - provides basic object interoperability functionality.
  - CORBA services
    - A set of services accessible through APIs
    - Provide common and standardized functionality – persistence, security, life cycle, etc.
  - CORBA facilities
    - High-level services needed by applications rather than individual objects like document management, i18n, mobile agent support, etc.

Object Brokers
How CORBA works?



- In order to be accessible through an ORB object needs to declare interface
  - Similarly to RPC we rely on CORBA's IDL.
  - Supported OO features such as inheritance and polymorphism.
  - Compiler generates stubs and skeletons.

- All programmer needs to know is server's IDL interface.

- Alongside with of static interface binding CORBA allows dynamic discovery and invocation of objects by relying on two components:
  - Interface repository
    - Stores IDL definitions of all objects associated to an ORB.
    - Applications can access the repository to browse, edit and delete IDLs.
  - Dynamic invocation interface
    - Provides operations such as get_interface and create_request for repository browsing and dynamic method invocation construction.
- References to service objects are retrieved through CORBA's Naming and Trading services
  - Trading service enables search for service objects based on their properties
  - Naming service resolves object names into object references.
  - Semantics is an issues – ontologies shared between clients and service objects are missing.

- Encapsulation refers to the possibility to hide the internals of an object implementation from clients.
- It's present to some extent already in RPC and TP Monitor approaches, but it fits perfectly with CORBA.
- In CORBA there is no need to use same programming languages and operating systems
  - Client and server are totally detached.
  - IDL represents the connecting point.
- CORBA standardizes mappings between IDL and various programming languages
  - This feature is eliminating the heterogeneous behavior of IDL compilers.
- Location independence contributes to encapsulation
  - Reference to a server object is just an identifier.

Technical Solution

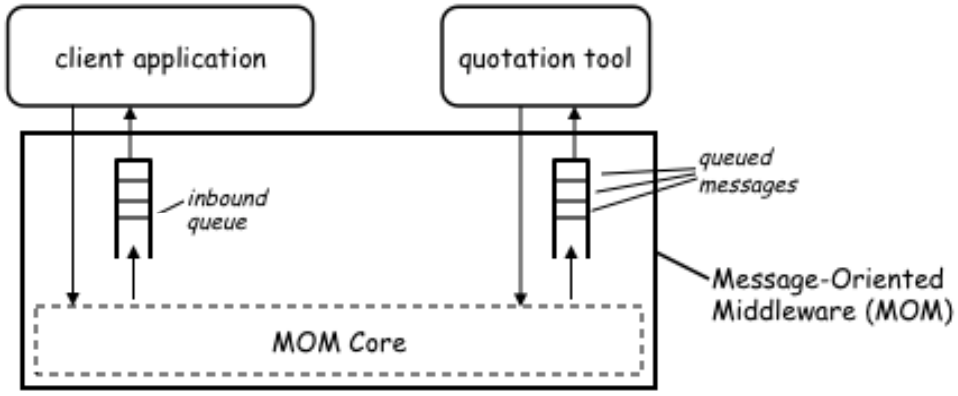
Message-Oriented Middleware

- Asynchronous communication is in the heart of Message-Oriented Middleware (MOM)
  - RPC also had an extension to cope with asynchronous calls
  - TP Monitors introduced queues to implement message-based interactions.
- Modern MOM is descendent of queuing systems found in TP Monitors
  - Used for batch processing but later switched to cope with the interoperability issues in the context of multiple heterogeneous systems and programming languages.
- Major approach to integrate information systems and applications today is by relying on MOM
- Current solutions in the area include IBM Web Sphere MQ, Microsoft Message Queuing (MSQM), … but also CORBA has it's service.

- It refers to an interaction paradigm where clients and service providers communicate by exchanging messages.
- Message is a structured data set characterized by its type and a set of pairs consisting of names and values.
- Client and service provider must agree on the set of message types exchanged during the communication.
- MOM supports message-based interoperability.
- Difference between client and server is here blurred.

STI · INNSBRUCK

- Messages sent by a MOM client are placed into a queue.
- Queue is identified by a name and possibly bound to a specific intended recipient.
- Recipient picks up and processes the message when suitable.
- More robust on failures, flexible in terms of performance optimizations:
  - Queues may be shared between applications.
  - Messages may have priorities.



STI · INNSBRUCK

- Transactional queuing = reliable messaging.
- Messages will be delivered once and only once even if MOM goes down between delivery
  - Messages are persisted.
- The system also copes with failures
  - Recipients mat bundle a couple of messages into atomic the units.
  - In cased of failure at consumer side the messages are placed back in the queue to be consumed again.
  - From the producer point of view messages in the queue inside the transactional brackets are valid and visible only until execution of atomic unit is completed. In case of some unrecoverable failures they must be deleted from the queue.
- Putting back messages in the queue may be problematic
  - Messages may be trapped between the failing receiver and queue.
  - Sender mat not be notified about the problem.

Illustration By a Larger Example

Illustration by a larger example
Java RMI

STI · INNSBRUCK

- Java Standard Edition comes bundled with the compilers and libraries needed to implement solutions based on object-oriented RPC
  - Java™ Remote Method Invocation (RMI)
- Java™ RMI enables development of Java2Java–based applications in which methods of remote Java objects can be invoked from other JVMs.
  - It uses object serialization to marshal and unmarshal parameters.
- The Hello World example
  - http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

STI · INNSBRUCK

- Steps needed to implement, compile and execute RMI application in Java:

1. Define the remote interface
2. Implement the server
3. Implement the client
4. Compile the source files
5. Start the Java RMI registry, server, and client

Illustration by a larger example
Define the Remote Interface

STI · INNSBRUCK

```
package example.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Illustration by a larger example
Implement the Server

STI · INNSBRUCK

```
package example.hello;

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
        return "Hello, world!";
    }

    public static void main(String args[]) {

        try {
            Server obj = new Server();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Illustration by a larger example
Implement the Client

STI · INNSBRUCK

```
package example.hello;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Illustration by a larger example
Compile the Source Files and Starting the System

STI · INNSBRUCK

Compiling the source files

```
javac -d destDir Hello.java Server.java Client.java
```

Starting the RMI Registry, Server Object and Client

```
rmiregistry &

java -classpath classDir -Djava.rmi.server.codebase=file:classDir/ example.hello.Server &

java  -classpath classDir example.hello.Client
```

Summary

Summary

STI · INNSBRUCK

- Middleware represents a set of solutions to the common problems occurring during the integration of systems and application in the multi-tier systems.

- During the previous years a number of middleware solutions have been proposed according to the different programming paradigms
  - RPC,
  - TP Monitors,
  - Object Brokers, and
  - Message-Oriented Middleware.

- As such middleware is today indispensable in course of creating robust, fail safe and complex distributed information systems.

References

References

STI · INNSBRUCK

- Mandatory reading
  - Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. Web Services - Concepts, Architectures and Applications. Springer-Verlag, 2004. **second chapter**

- Wiki and Web references
  - Middleware http://en.wikipedia.org/wiki/Middleware
  - IDL http://en.wikipedia.org/wiki/Interface_description_language
  - RPC http://en.wikipedia.org/wiki/Remote_procedure_call
  - Transaction processing http://en.wikipedia.org/wiki/Transaction_processing
  - CICS http://en.wikipedia.org/wiki/CICS
  - ACID http://en.wikipedia.org/wiki/ACID
  - Two phase commit http://en.wikipedia.org/wiki/2PC
  - CORBA http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
  - ORB http://en.wikipedia.org/wiki/Object_request_broker
  - MOM http://en.wikipedia.org/wiki/Message-oriented_middleware
  - J2EE http://en.wikipedia.org/wiki/J2EE

Next Lecture

STI · INNSBRUCK

#	Title
1	Distributed Information Systems
2	Middleware
3	**Web Technologies**
4	Web Services
5	Basic Web Service Technologies
6	Web 2.0 Services
7	Web Service Security

Questions?

STI · INNSBRUCK