CPE 545

Distributed objects

Distributing Objects - Highlights

- Application designers may take advantage of the expressiveness, abstraction, and flexibility of an object model.
- Encapsulation allows an object's implementation to be placed on any site.
- Legacy applications may be reused by encapsulating them in objects, using the “wrapper” pattern.
- Scalability is enhanced by distributing processing power over a network of servers.

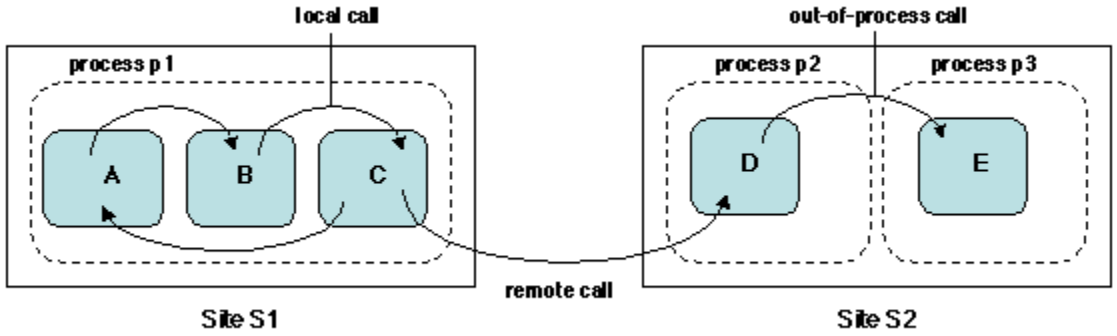
Distributing Objects

- ***Local invocation:*** The calling and called objects are in the same process.

- ***Out-of-process invocation****:* The calling and called objects are executed by different processes on the same site.

- ***Remote invocation:*** The calling and called objects are on different nodes.

Distributing Objects



Models for distributing objects

- The *fragmented objects* model, in which an object may be split in several parts, located on different nodes, and cooperating to provide the functionality of the object.

- The *replicated objects* model, in which several copies of a given object may coexist to increase availability and to improve performance.

- The *migratory* (or *mobile*) objects model, in which an object may move from one node to another one to improve performance through load balancing, and to dynamically adapt applications to changing environments.

Models for distributing objects

- Non-local forms of invocation rely on an *object request broker* (ORB), a middleware that supports distributed objects.

- An ORB has the following functions:
  - Identifying and locating objects.
  - Binding client to server objects.
  - Performing method calls on objects.
  - Managing objects' life cycle (creating, activating, deleting objects)

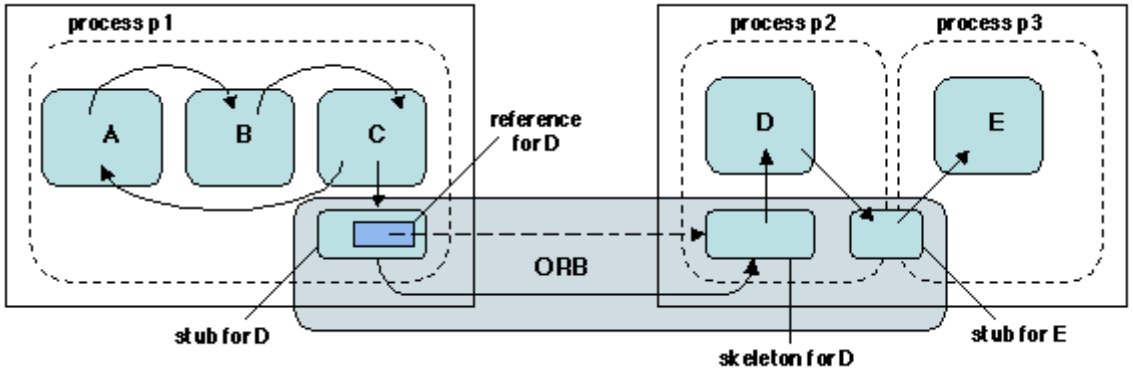
Remote Object Call: a First Outline

- A client-server model: a process or thread executing a method of a client object sends a request to remote server object in order to execute a method of that object.

- It relies on a stub-skeleton pair. In contrast with RPC, the stub and the skeleton are objects in their own right.

Remote Object Call: a First Outline

- The stub for *D*, on client *C*'s site has the same interface as *D*. It forwards the call to *D*'s skeleton on *D*'s site, which performs the actual method invocation and returns the results to *C*, via the stub.



Remote Object Call: a First Outline

- In order to be able to forward the invocation, *D*'s stub contains a reference to *D*'s skeleton.
- A *reference* to an object is a name that allows access to the object; it contains information allowing the object to be located (port, address).
- An *out-of-process* call on the same node (e.g. from object *D* to object *E*) could in principle be performed as a remote invocation using shared memory.

Remote Object Call: a First Outline

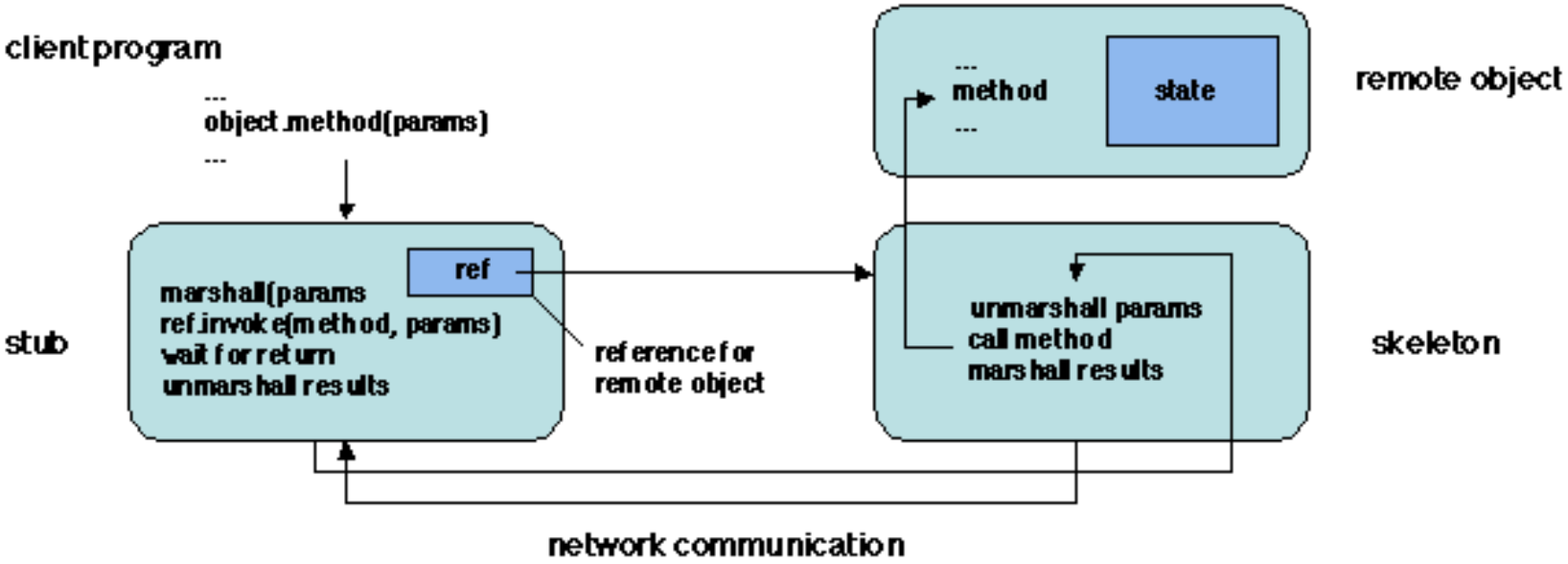
- The client process invokes the method on the local stub of the remote object
- The stub marshalls the parameters and constructs a request,
- Determines the location of the remote object using its reference,
- Sends the request to the remote object's skeleton.

Remote Object Call: a First Outline

- On the remote object's site:
  - The skeleton performs unmarshalling parameters,
  - Dispatching the call to the address of the invoked method
  - Marshalling returned values, and sending them back to the stub.
- On the local site:
  - The stub unmarshalls the returned values and delivers them to the client process, thus completing the call.

Remote Object Call: a First Outline

- Important difference with RPC is that objects are dynamically created

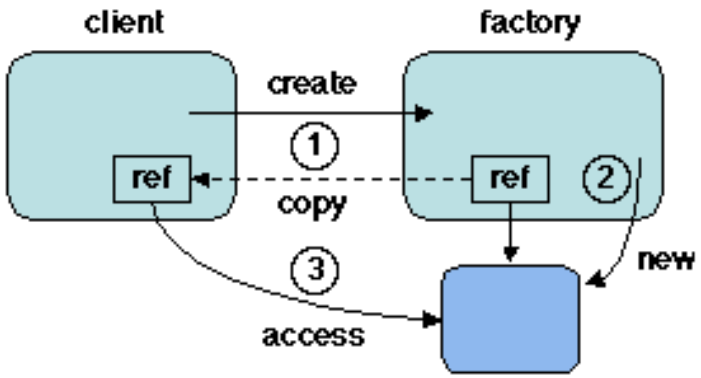


The User's View

- Remote object infrastructures attempt to hide distribution from the user, however, some aspects of distribution remain visible.
- The main distribution-aware aspect is object creation and location.
- Creating a remote object cannot be done through the usual object instantiation mechanism, which involves memory allocation and is not directly applicable to a remote node.

The User's View – Create Object

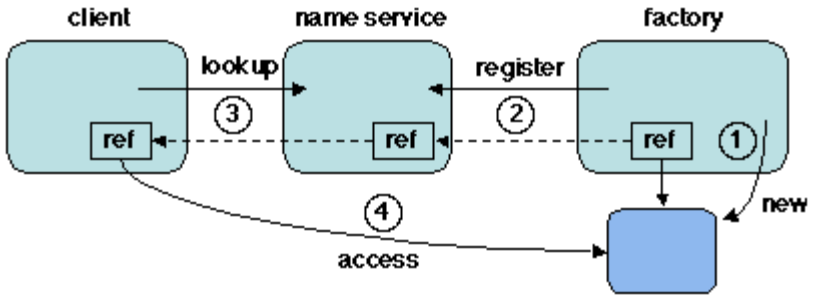
- Creating a remote object is done through an object factory , which acts as a server that creates objects of a specified type.
- The reference of the created object is returned to the client .



(a) Initiated by client

The User's View – Create Object

- An object may also be created at the server's initiative. In that case the server usually registers a reference to the object in a name service, to be later retrieved by the client



(b) Initiated by server

The User's View – Access Object

- A remote object may only be accessed by a client program through a reference
- A reference may be obtained as through a name service.

The User's View – a closer view

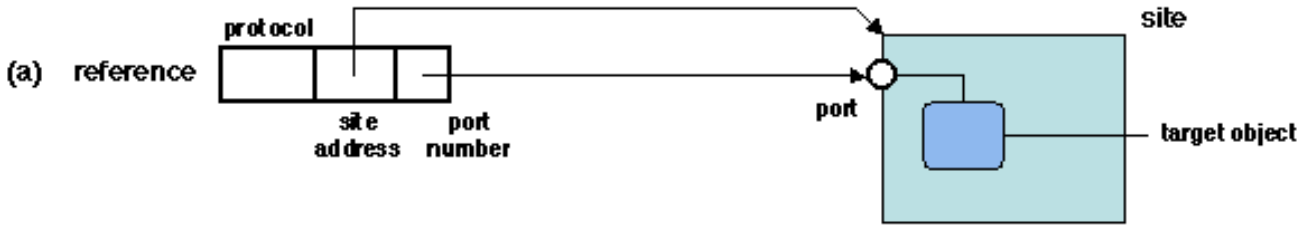
- What information should be contained in an object reference?
- How is the remote object activated, i.e how is it associated with a process or thread that actually performs the call?
- How are parameters passed, specially if the parameters include objects?

Object References

- An object reference must refer to the object's interface.
- The reference must provide all the information that is needed to perform a remote access to the object.

Object References

- The reference may contain the network address of remote location. Lacks flexibility, since it does not allow the target object to be relocated without changing its reference.

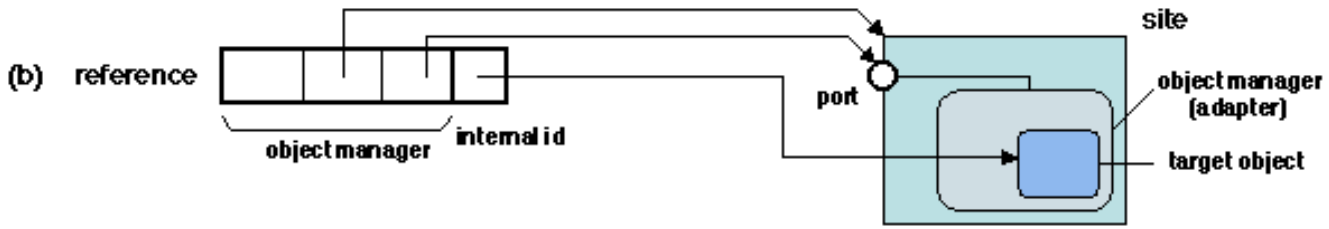


Object creates the reference

Object References

- The reference may contain the network address of an object adapter + an internal object identification on this adapter*

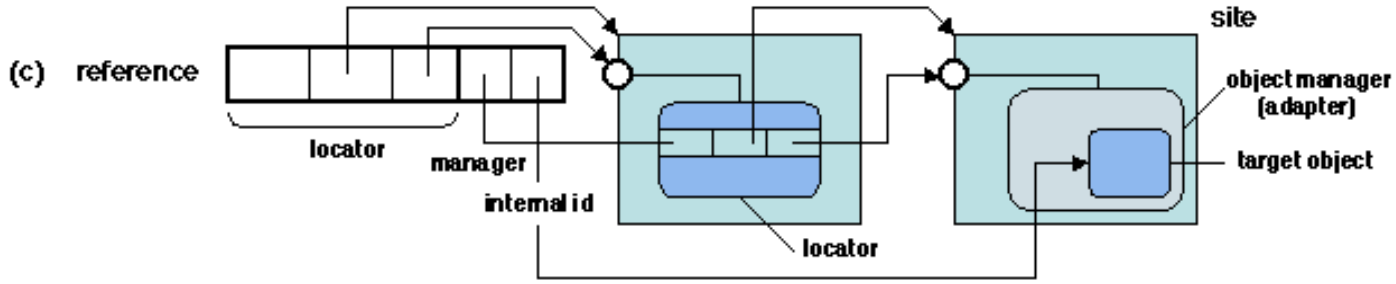
  * We will use this model for references through out the lecture notes



Adaptor creates the object reference

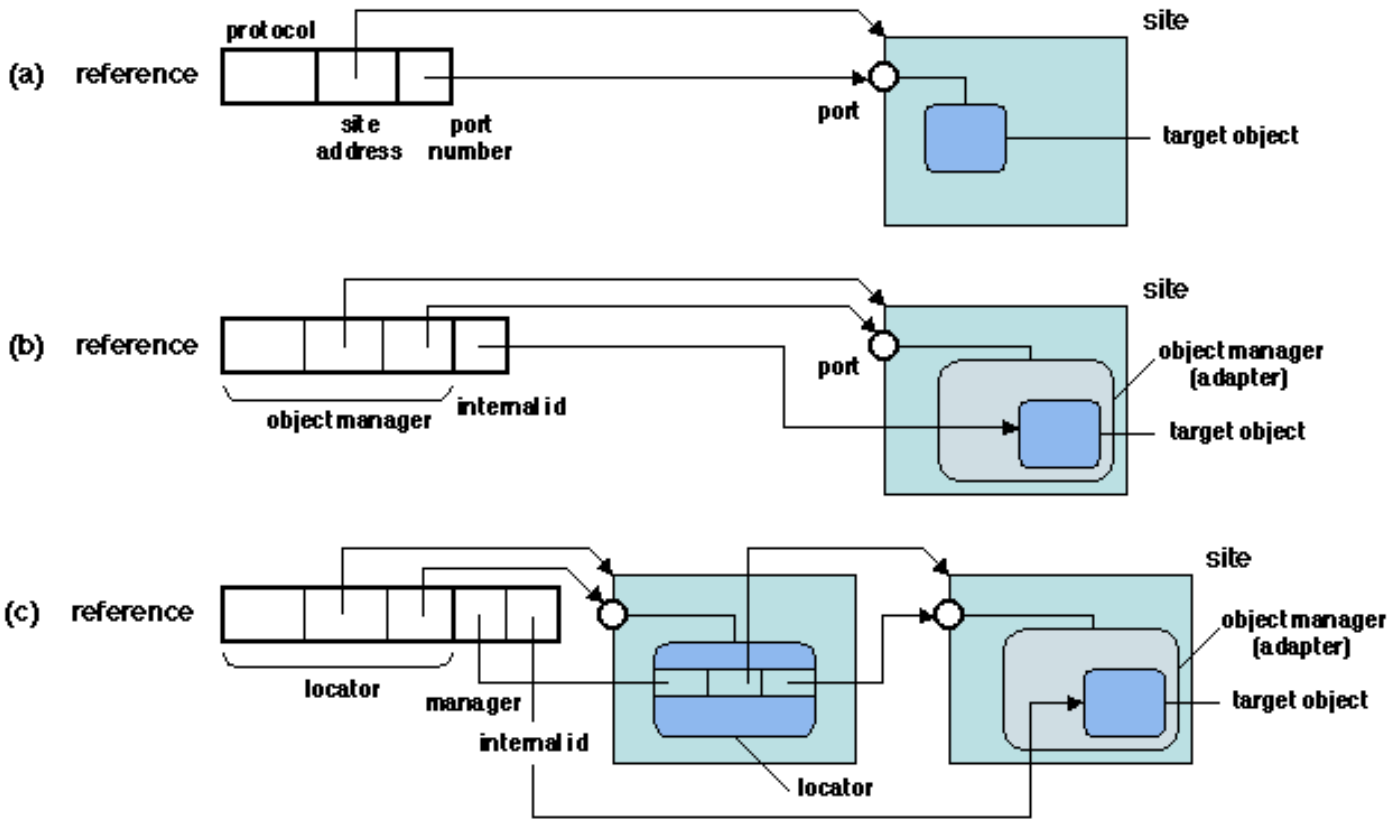
Object References

- The reference may contain the network address of a locator for the adapter + the identity of the adapter + the internal object identification. This allows the adapter to be transparently relocated on a different site, updating the locator information without changing the reference.



Adaptor creates the object reference which will be referenced by a locator

Object References

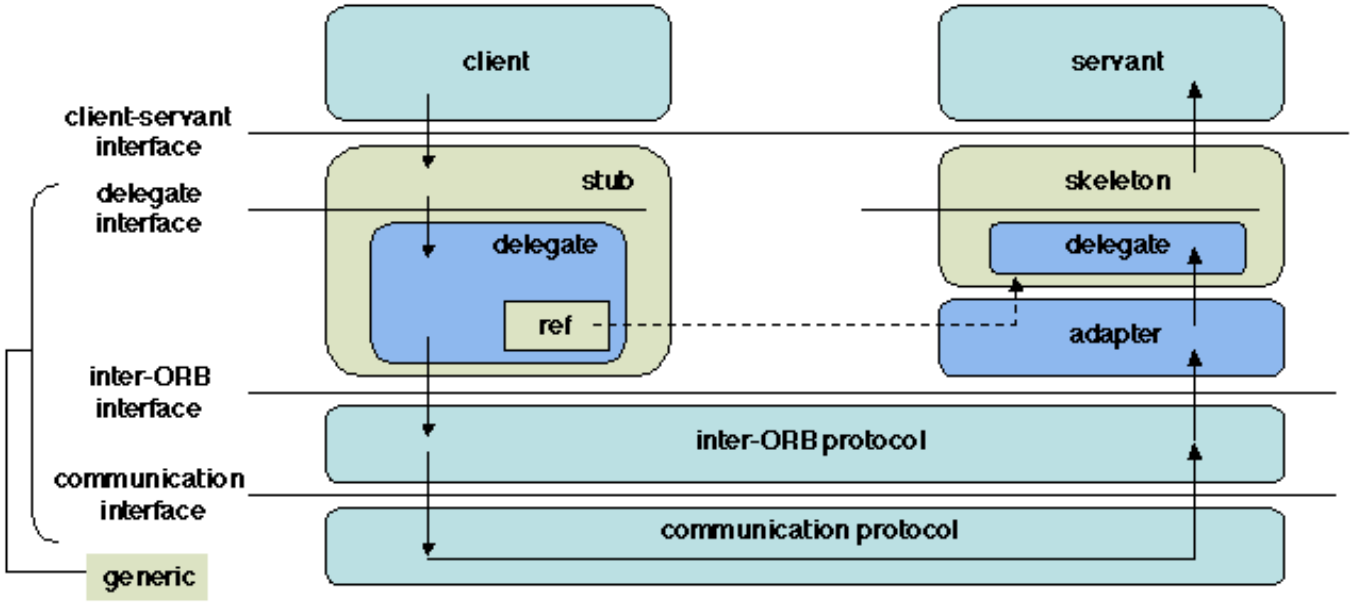


Remote Invocation – Server side
Abstraction, portability & interoperability

- The interface transformation between service and servant is done through a server delegate, as part of the skeleton.
- The skeleton (and thus the servant) is located through an *object adapter*.
- Delegates are part of the internal organization of a stub, and are never explicitly seen by a user,
- Adapters are usually visible and may be directly used by applications.

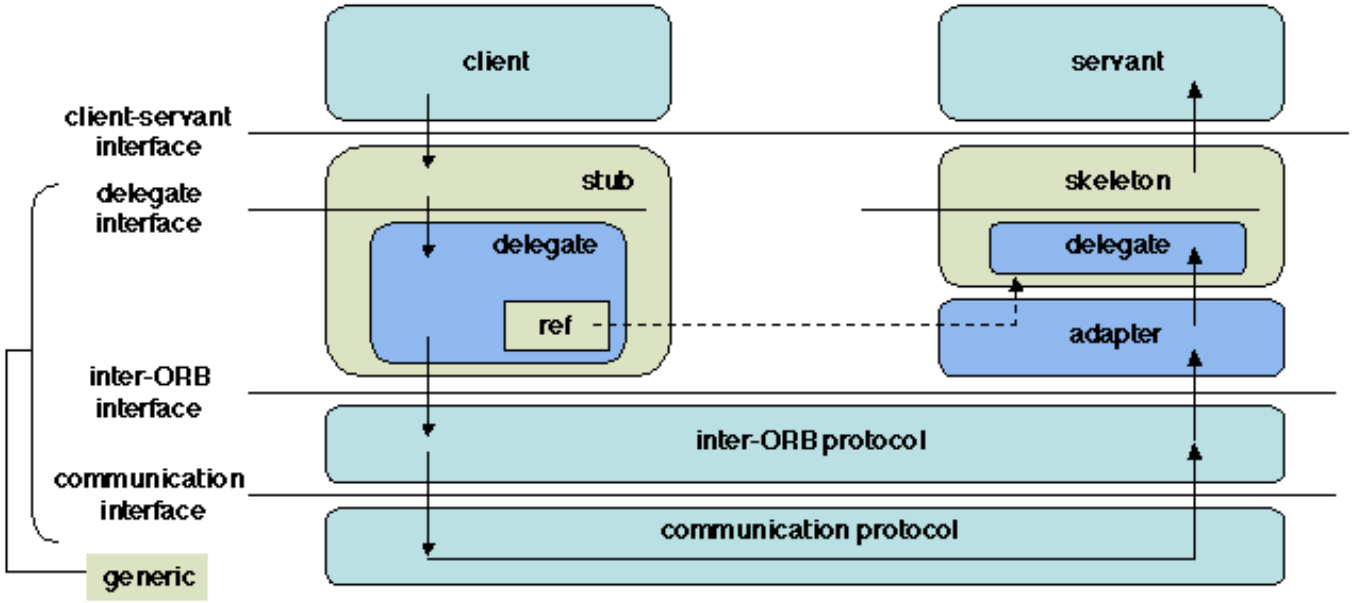
Remote Invocation – Client side
Abstraction, portability & interoperability

- Upper interface of the stub is application-specific.
  - A. Application interface to stub (Interface entry points and data format are standardized).
  - B. Delegate interface from Stub (not visible to users):
    - *create_request()* - constructing a invocation request in a standard form
    - *Invoke()* - actually performing the invocation.



Remote Invocation – Client side
Abstraction, portability & interoperability

- Lower interface of the stub, connects it to the network.
  - *C. ORB Interface from Delegate (not visible to users):*
    - *SendRequest()* – Parameters:
      - Reference of the called object,
      - the name of the method,
      - the description of the parameters of the called method.



Object Adapters - Functions

- Registers servant objects when they are created (installed) on the server;
- Creates object references for the servant objects (i.e. finds a servant object using its reference) – remember the string names (mp3, vlc, mp4,…) were used to find the correct object servant in media player example
- Activates a servant object when it is called, (i.e. associates a process with it).
- Creating/Installing Servant Object using object factory
- Used when an existing (or *legacy*) application, not written in object style, must be reused in an object-oriented setting.
- Passing parameters

Object Request Broker Standard

- *General Inter-ORB Protocol* (GIOP), allow different CORBA implementations to interoperate
- The GIOP specification covers three aspects.
  - A common format for the *data* being transmitted, called Common Data representation (CDR);
  - A format for the *messages* used for invoking remote objects;
  - Requirements on the underlying transport layer.

Object Invocation

- A Request message includes:
  - a reference for the target object,
  - the name of the invoked operation,
  - the parameters (in marshalled form),
  - an identifier for a reply holder (if the reply is expected).
- After sending the message, the calling thread is put to wait.
- At the receiving end, the servant is located using the reference and the request is forwarded to it.

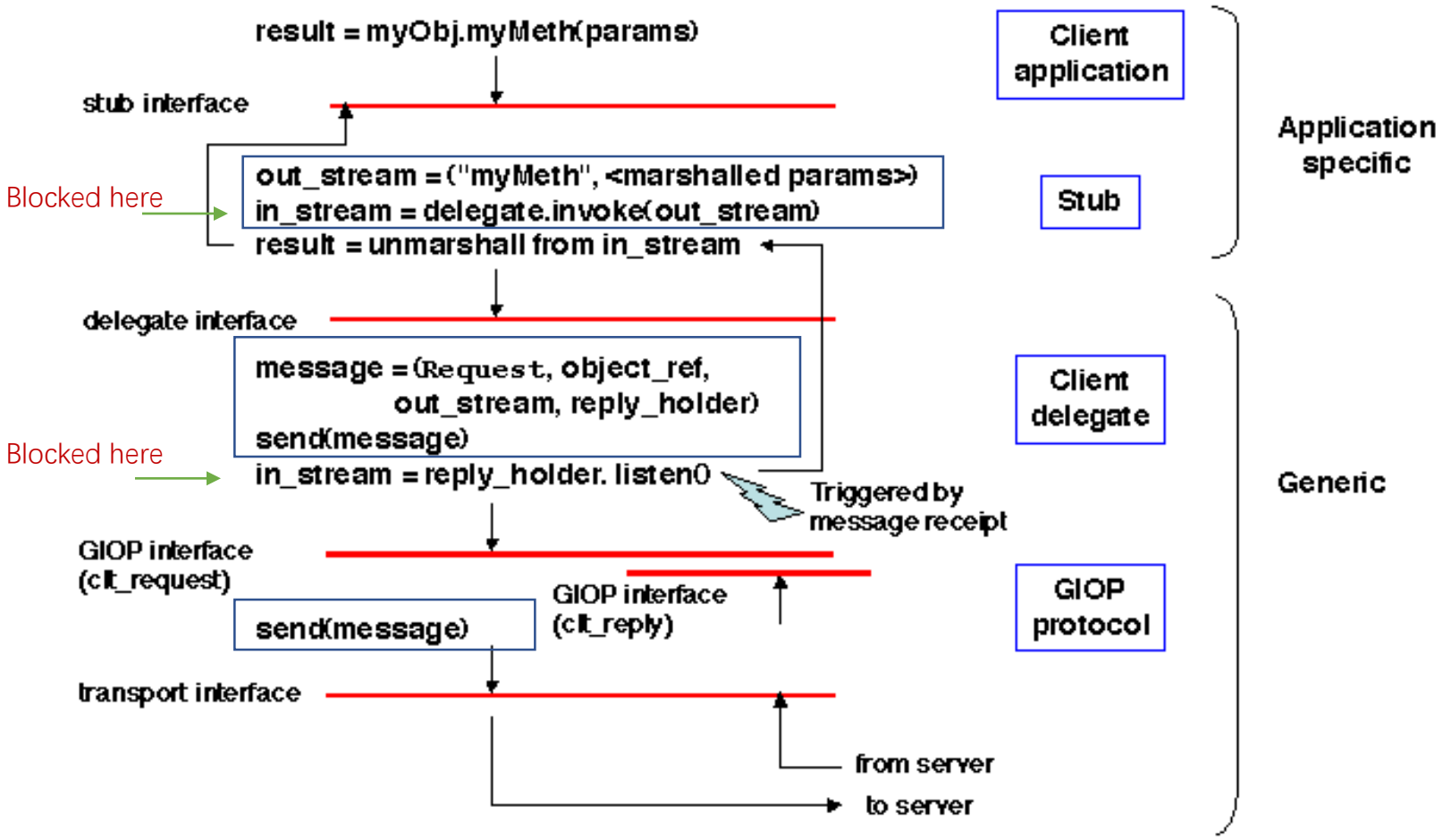
Object Invocation

- A Reply message, only created if the invoked operation returns a value (in marshalled form), together with the reply holder.
- When this message is received by the client, the waiting thread is activated and may retrieve the reply from the reply holder.

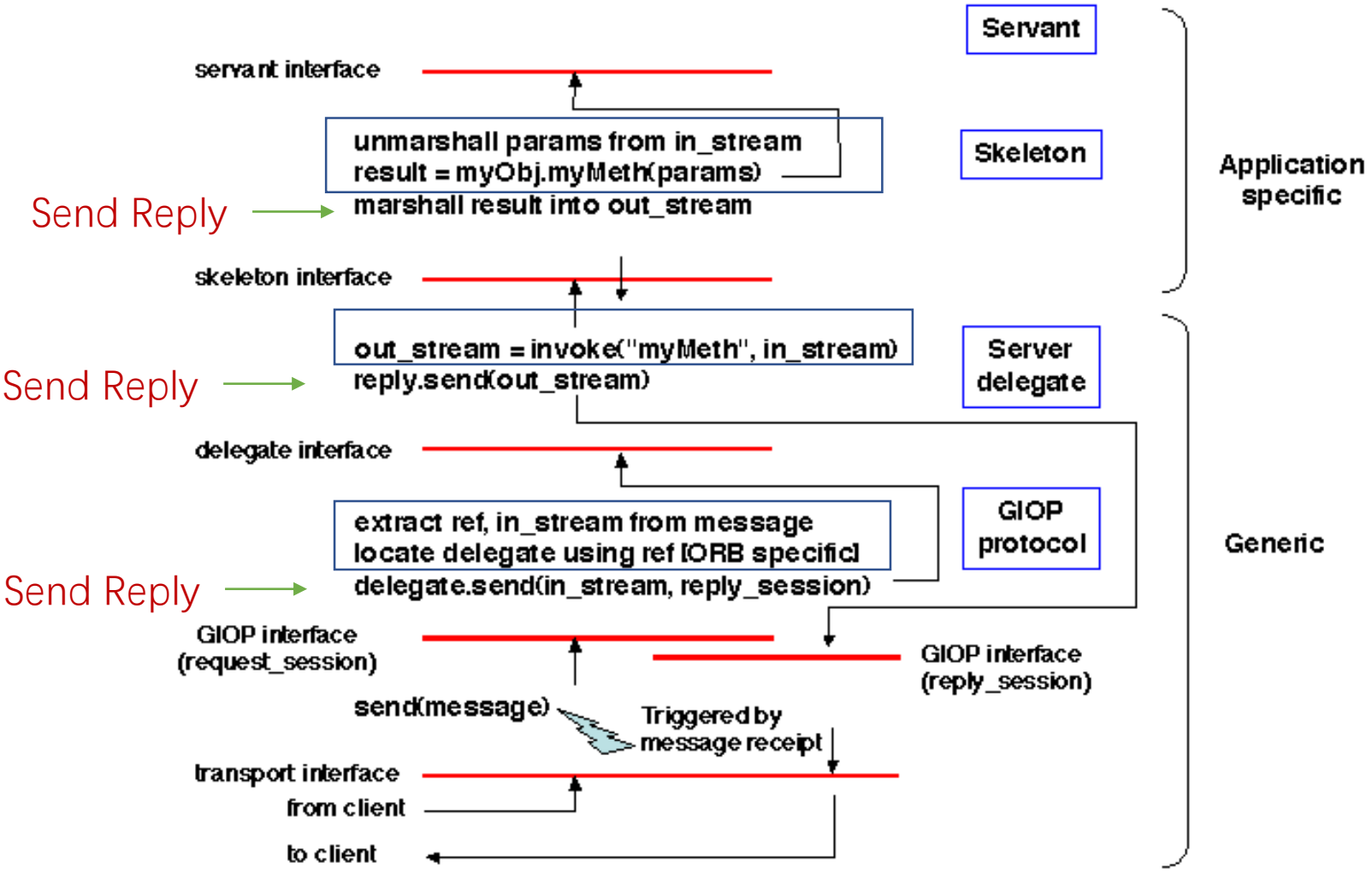
Object Invocation – Client side

- The invocation path goes through a stub (at the client end) and a skeleton (at the server end).
- The stub and the skeleton are separated into an (upper) application-specific part and a (lower) application-independent part called a *delegate*.

Object Invocation – Client side



Object Invocation – Server side



References

- Middleware Architecture with Patterns and Frameworks, Sacha Krakowiak
- Designing Embedded Communications Software, by T. Sridhar, ISBN: 157820125x, CMP Books
- IT Architectures and Middleware – 2nd edition. Chris Britton, Peter Bye. Addison-Wesley
- Tanenbaum & van Steen Distributed Systems: Principles and Paradigms, 2nd ed. ISBN: 0-132-39227-5. [Schmidt et al. 2000]Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. John Wiley & Sons. 666 pp.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.
- Völter, M., Kircher, M., and Zdun, U. (2004). *Remoting Patterns: Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware*. John Wiley & Sons.