

Remoting Patterns

Uwe Zdun, Markus Völter, Michael Kircher
zdun@acm.org, voelter@acm.org, michael.kircher@siemens.com

Overview

- **Patterns and Pattern Languages**
- **Distributed Systems and Middleware**
- **Remoting Patterns**
 - Basic Remoting Patterns
 - Identification Patterns
 - Lifecycle Patterns
 - Extension Patterns
 - Extended Infrastructure Patterns
 - Asynchronous Invocation Patterns
- **Web Services Technology Projection**

Pattern Definition

- **Pattern definition by Alexander:**

A pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.

- **Heavily simplified definition. Bad example:**

Context	You are driving a car.
Problem	The traffic lights in front of you are red. You must not run over them. What should you do?
Solution	Brake.

- **Obviously this is not a pattern!**
- **Alexander's definition is much longer. Summary by Jim Coplien:**

Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

3

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Patterns and Pattern Languages

4

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Elements of a Pattern

- Name
- Context
- Problem
- Solution
- Forces
- Consequences
- Examples/Known Uses
- Pattern Relationships

5

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Software Patterns

- **Last couple of years: Patterns have become part of the mainstream of OO SW development**
- **Different kinds of patterns:**
 - Design patterns (GoF)
 - Software architecture patterns (POSA, POSA2)
 - Analysis patterns (Fowler, Hay)
 - Organizational patterns (Coplien)
 - Pedagogical patterns (PPP)
 - Many others
- **Most of the patterns in this tutorial can be seen as falling into two categories - design patterns and architectural patterns**

6

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

From Patterns to Pattern Languages

- **Single pattern = one solution to a particular, recurring problem**
- **However: "Real problems" are more complex**
- **Pattern relationships:**
 - Compound patterns
 - Family of patterns
 - Collection or system of patterns
 - Pattern languages
- **Pattern languages:**
 - Language-wide goal
 - Generative
 - Sequences → has to be applied in a specific order
 - Pattern defines its place in the language → context, resulting context

7

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Distributed Systems and Middleware

8

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Distributed Systems: Application Areas

- **Some Examples:**
 - Internet
 - Telecommunication networks
 - Business-to-business (B2B) collaboration systems
 - International financial transactions
 - Embedded systems
 - Scientific applications
- **Many more ...**

9

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Distributed Systems: Reasons

- **Reasons: Problem-related reasons**
 - "Real" distribution
- **Reasons: Property-related reasons:**
 - Performance and Scalability
 - Fault Tolerance
 - Service and Client Location Independence
 - Maintainability and Deployment
 - Security
 - Business Integration

10

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Distributed Systems: Challenges

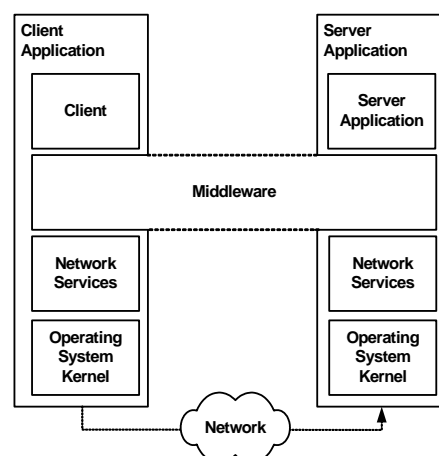
- Network Latency
- Predictability
- Concurrency
- Scalability
- Partial Failure

11

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Middleware

- **Dealing with low-level networking issues yields the following problems:**
 - not easy to scale,
 - cumbersome and error prone to use,
 - hard to maintain and change, and
 - does not provide transparency of the distributed communication.
- **Solution: Communication Middleware**

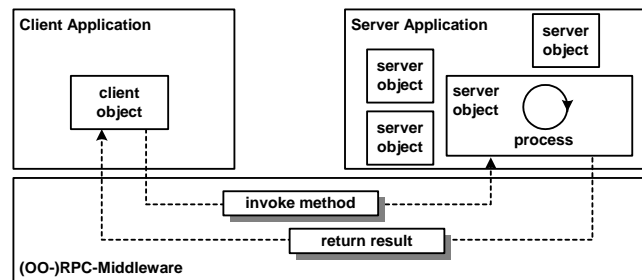


12

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Remoting Styles

- There are systems that
 - use the metaphor of a *remote procedure call* (RPC),
 - use the metaphor of posting and receiving *messages*,
 - utilize a *shared repository*, or
 - use continuous *streams* of data.
- We mainly focus on object-oriented variants of the RPC style:



13

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

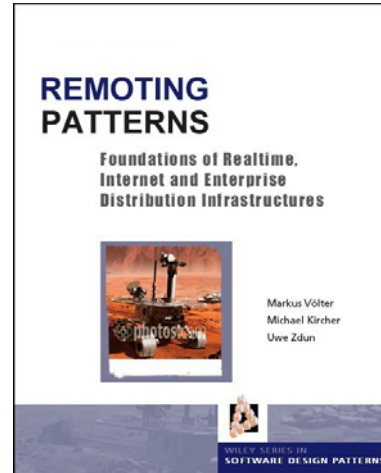
Remoting Patterns

14

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Remoting Patterns

- **Numerous projects**
 - *use,*
 - *extend,*
 - *integrate,*
 - *customize,* and
 - *build*
- **distributed object middleware**
- **Goals:**
 - illustrate the general, recurring architecture of successful distributed object middleware
 - illustrate more concrete design and implementation strategies
- **Book: to be published in Wiley's Pattern Series in 2004**



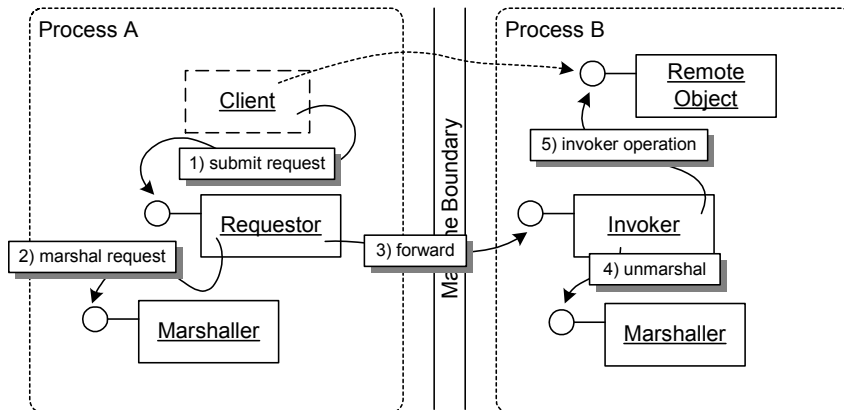
15 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Broker

- **Context:**
 - You are designing a distributed software system
- **Problem:**
 - Many challenges that do not arise in single-process software
 - communication across networks is unreliable
 - integration of heterogeneous components into coherent applications
 - efficient usage of networking resources
 - Developers should not lose their primary focus: to develop applications that resolve their domain-specific problems.
- **Solution:**
 - Separate communication-related concerns in a Broker
 - The Broker hides and mediates all communication between the objects or components of a system
- **Details realized by other patterns**

16 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

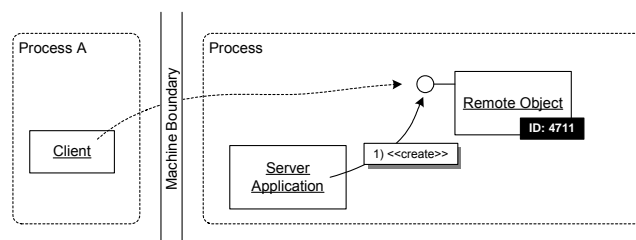
Pattern: Broker (2)



17 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Remote Object

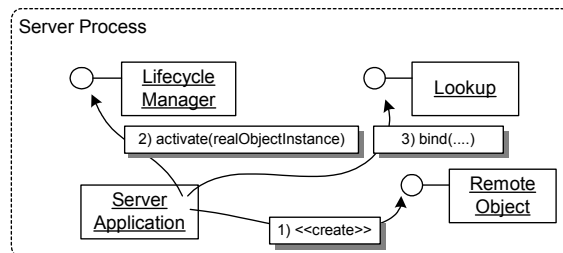
- **In many respects, accessing an object over a network is different from accessing a local object:**
 - Invocations have to cross process and machine boundaries
 - Network latency and network unreliability
 - Using memory addresses to define object identity will not work
 - ...
- **The application logic is provided in form of *remote objects***



18 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Server Application

- **Remote objects need a *server application* for the following tasks:**
 - Create and configure constituents of the distributed object middleware
 - Instantiate and configure remote objects
 - Advertise remote objects to clients
 - Connect individual remote objects to distributed applications
 - Destroy remote objects not needed anymore



19

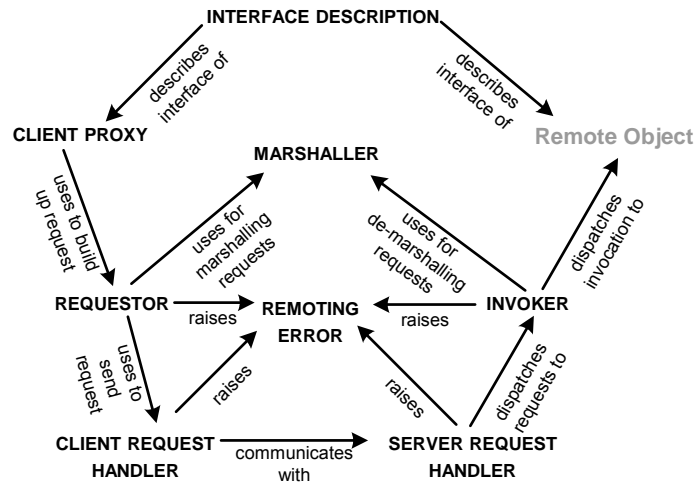
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Basic Remoting Patterns

20

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Basic Remoting Patterns



21

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

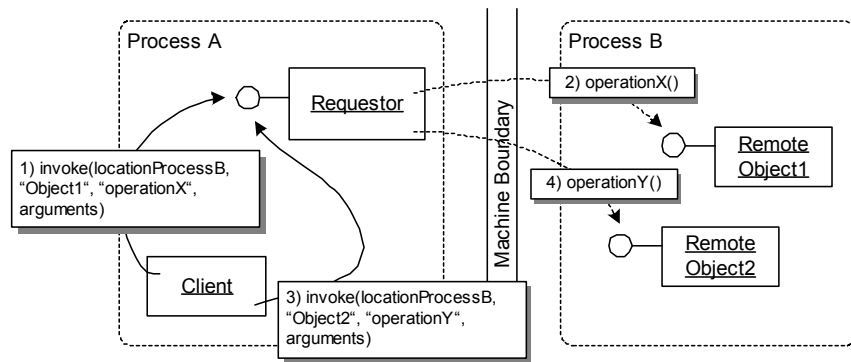
Pattern: Requestor

- **Context:**
 - A client needs to access one or more remote objects
- **Problem:**
 - For a remote invocation ...
 - Marshalling must be triggered
 - A connection needs to be established
 - The request information must be sent to the target remote object
- **Solution:**
 - In the client application ...
 - Use a Requestor for accessing the remote object
 - Supply it with the absolute object reference of the remote object, the operation name, and the arguments
 - The requestor constructs a remote invocation from these parameters and sends the invocation to the remote object

22

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Requestor (2)



23

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

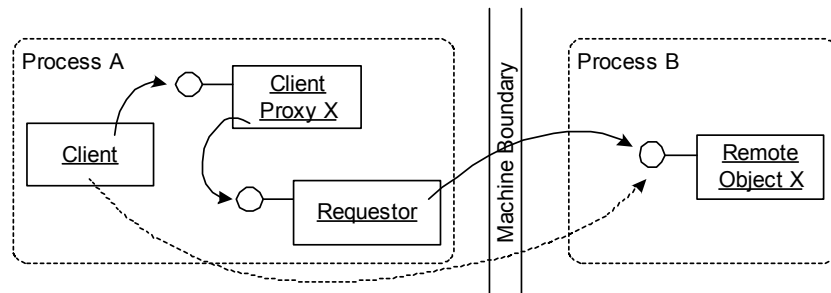
Pattern: Client Proxy

- **Context:**
 - A Requestor is provided by the distributed object middleware
- **Problem:**
 - Goal: support remote programming model similar to accessing local objects
 - A requestor solves part of this problem by hiding many network details
 - However: the methods, arguments, location and identification information have to be passed to the requestor for each invocation
 - Requestor supports no static type checking
- **Solution:**
 - Client proxy supports the same interface as the remote object
 - Clients only interact with the local client proxy
 - Client proxy translates the local invocation into parameters for the requestor, and triggers the invocation

24

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Client Proxy (2)



25

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

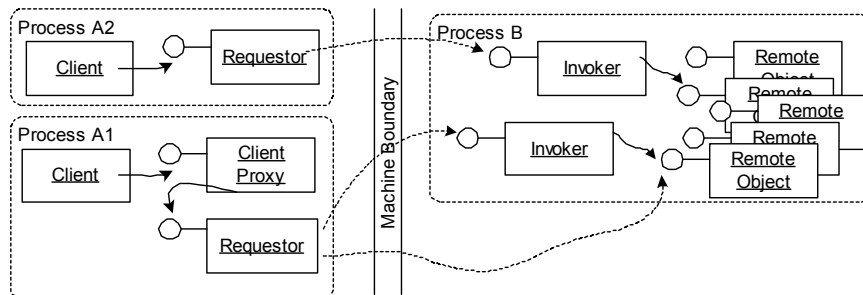
Pattern: Invoker

- **Context:**
 - A requestor sends a remote invocation for a remote object to a server
- **Problem:**
 - Somehow the targeted remote object on the server has to be reached
 - Simplest solution: remote object is addressed over the network directly
 - Large number of object → not enough network endpoints
 - Remote object would have to deal with network connections, receiving and demarshalling messages, etc.
- **Solution:**
 - Invoker(s) accept(s) invocations from requestors
 - Invoker reads the request and demarshalls it
 - Looks up the correct local object & operation and
 - Performs the invocation on the targeted remote object

26

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Invoker (2)



27

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

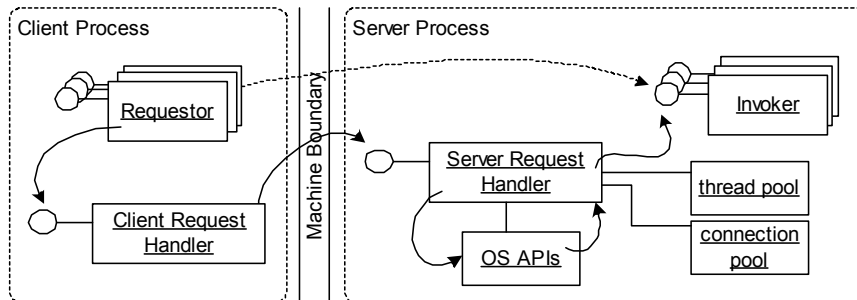
Pattern: Server Request Handler

- **Context:**
 - You are providing remote objects in a server application, and invokers are used for message dispatching
- **Problem:**
 - The request message has to be received from the network
 - Managing communication channels efficiently and effectively is essential
 - Network communication needs to be coordinated and optimized
- **Solution:**
 - Server request handler deals with all communication issues of a server application:
 - Receives messages from the network
 - Combines the message fragments to complete messages
 - Dispatches the messages to the correct invoker
 - Manages all the required resources (connections, threads, ...)

28

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Server Request Handler (2)



29

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

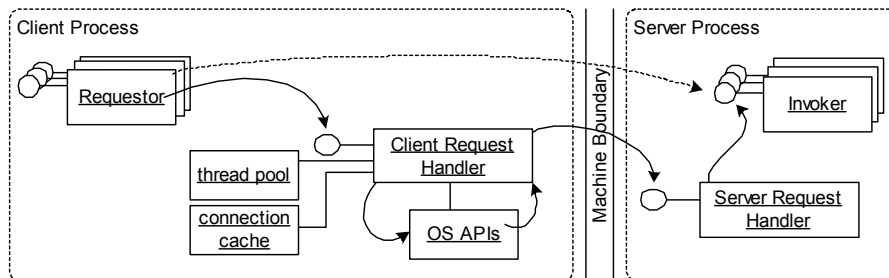
Pattern: Client Request Handler

- **Context:**
 - Requestor has to send requests to and receive responses
- **Problem:**
 - For a client request several tasks have to be performed:
 - Connection establishment and configuration
 - Result handling
 - Timeout handling
 - Error detection
 - Connection, threading, etc. need to be coordinated and optimized
- **Solution:**
 - Client request handler handles network connections for all requestors within a client:
 - Sending of requests
 - Receiving and dispatching of responses
 - Handling of timeouts, threading issues, and invocation errors

30

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Client Request Handler (2)



31

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

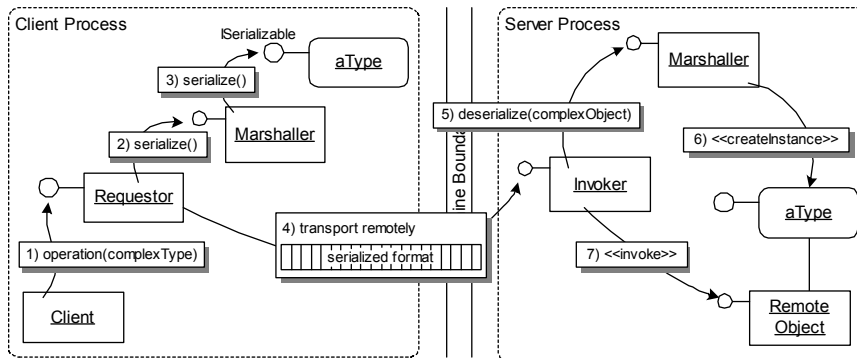
Pattern: Marshaller

- **Context:**
 - Request and response messages have to be transported over the network
- **Problem:**
 - The data to describe invocations consists of:
 - Object ID, operation name, parameters, return value, ...
 - Possibly other invocation context information
 - For transportation over the network, only byte streams are suitable
- **Solution:**
 - Use compatiblemarshallers on client and server side that serialize invocation information
 - Primitive types and non-primitive types are serialized
 - References to remote objects are serialized as absolute object references

32

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Marshaller (2)



33

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

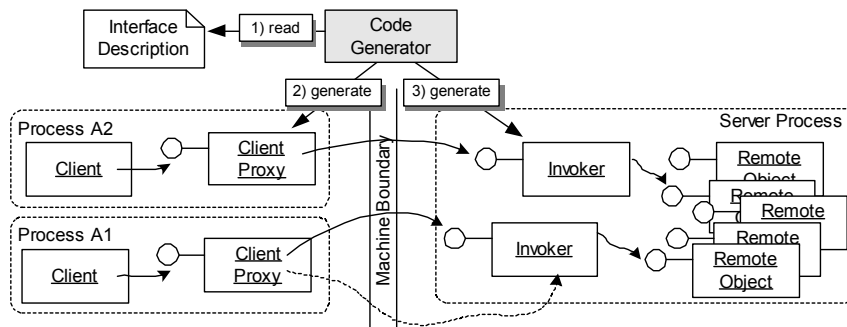
Pattern: Interface Description

- **Context:**
 - A client wants to invoke a remote object operation using a client proxy
- **Problem:**
 - Interfaces of client proxy and remote object need to be aligned
 - Marshalling and de-marshalling needs to be aligned
 - Client developers need to know the interfaces of the remote objects
- **Solution:**
 - Interface description describes the interface of remote objects
 - Interface description serves as the contract between client proxy and invoker
 - Client proxy and invoker use either code generation or runtime configuration techniques to adhere to that contract

34

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Interface Description (2)



35

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

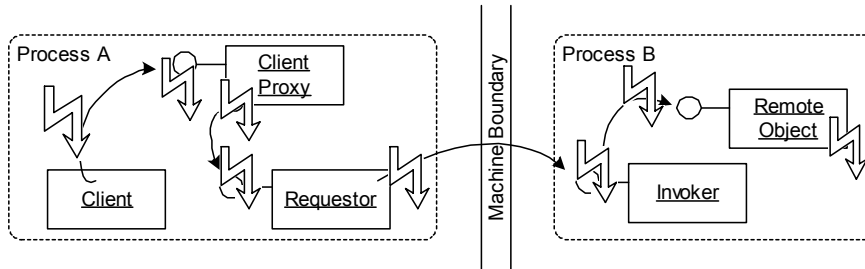
Pattern: Remoting Error

- **Context:**
 - Remote communication is inherently unreliable
- **Problem:**
 - Distributed invocations can never be completely transparent
 - Apart from errors in the remote object itself, new kinds of errors can occur, e.g.: network failures, server crashes, or unavailable objects
 - Clients need cope with such errors
- **Solution:**
 - Distinguish distribution-related errors and application-logic-related errors
 - Invoker, requestor, and request handlers detect and propagate remoting errors

36

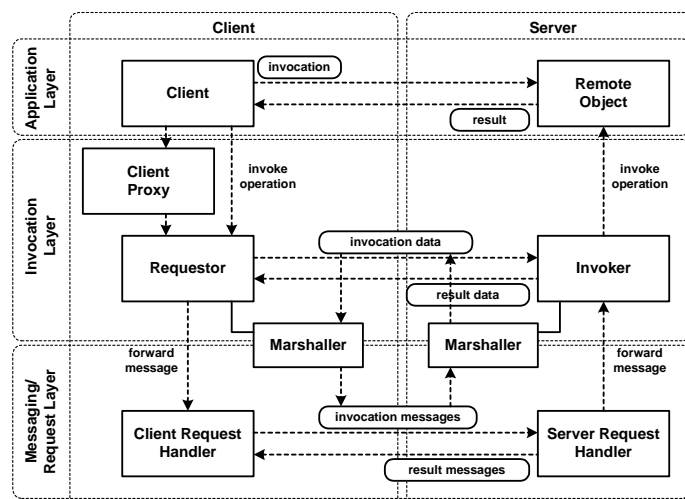
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Remoting Error (2)



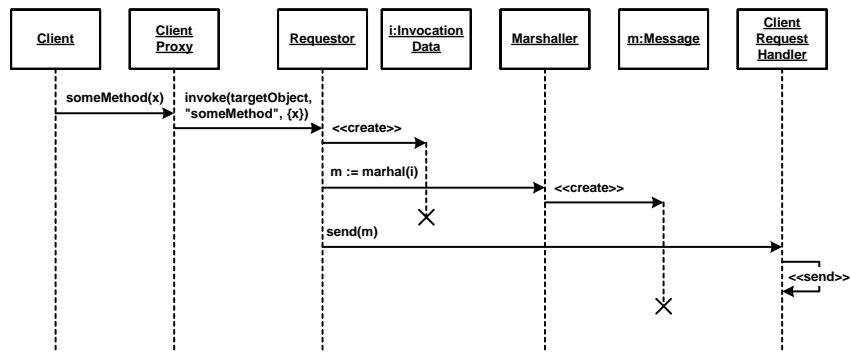
37 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions of the Patterns



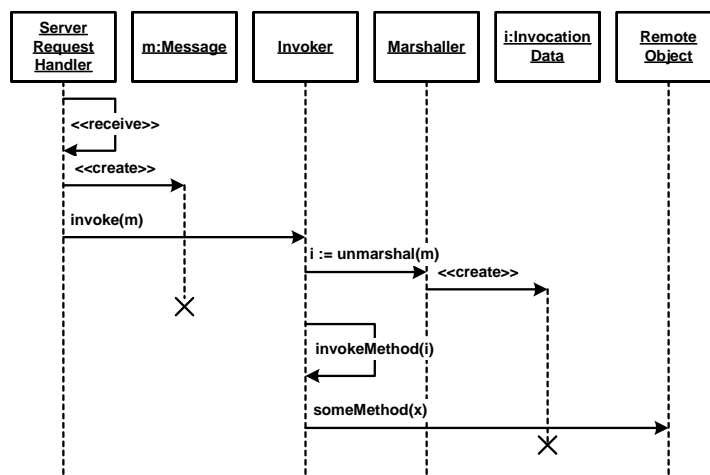
38 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Client-Side Invocation



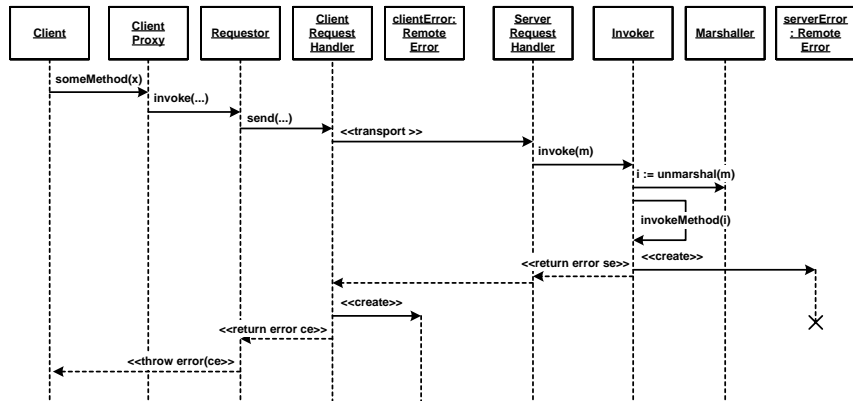
39 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Server-Side Invocation



40 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Remoting Error Propagation



41

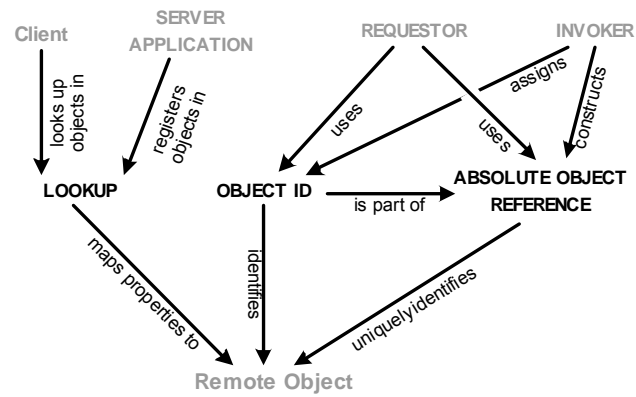
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Identification Patterns

42

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Identification Patterns



43

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

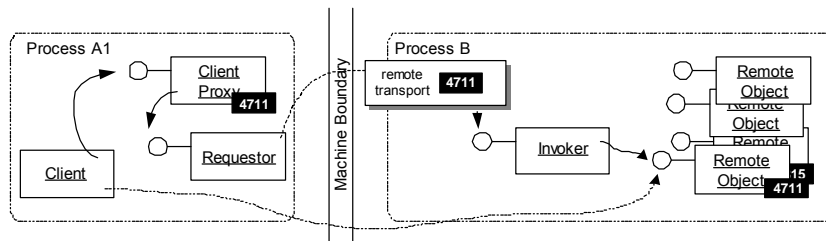
Pattern: Object ID

- **Context:**
 - The invoker has to select between registered remote objects to dispatch invocations
- **Problem:**
 - The invoker handles invocations for several remote objects
 - The invoker has to determine the remote object corresponding to a particular invocation.
- **Solution:**
 - Associate remote object instance with an object id that is unique in the context of the invoker
 - The client/client proxy have to provide the object id to the requestor

44

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Object ID (2)



45

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

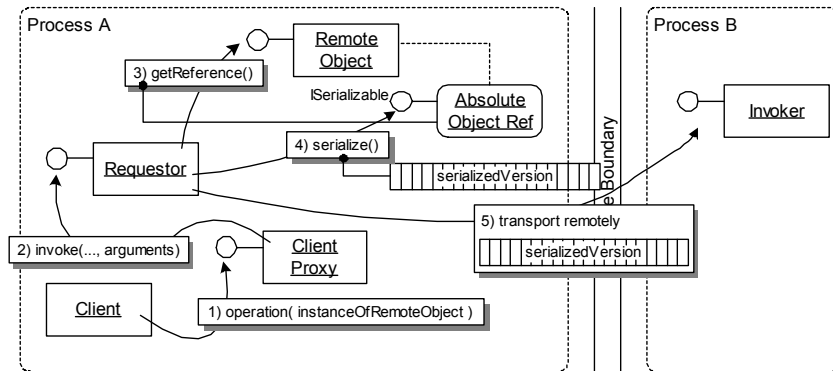
Pattern: Absolute Object Reference

- **Context:**
 - The invoker uses Object IDs to dispatch the invocations
- **Problem:**
 - Object ID allows the invoker to dispatch the remote invocation to the correct target object
 - But first the invocation has to be delivered to the correct server request handler
- **Solution:**
 - Absolute object reference uniquely identifies invoker and remote object:
 - Endpoint information (host, port)
 - ID of the invoker
 - Object ID
 - Clients exchange references to remote objects by exchanging the absolute object references

46

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Absolute Object Reference (2)



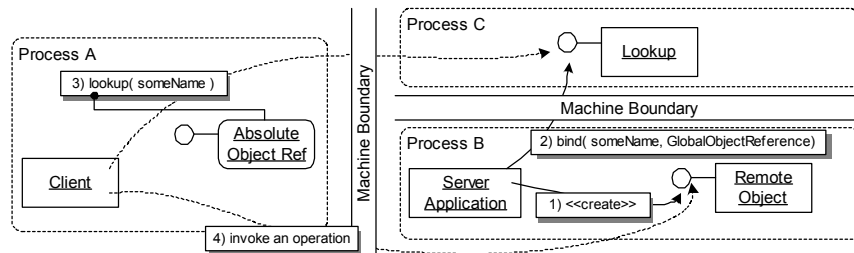
47 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Lookup

- **Context:**
 - Client applications want to use services provided by remote objects
- **Problem:**
 - Client has to obtain the absolute object references of the remote object
 - This remote object might change:
 - Serving instance changes
 - Location changes
 - Server restart
- **Solution:**
 - Lookup service is part of the distributed object framework
 - Server applications register references to remote objects
 - References are associated with properties (e.g. names)

48 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

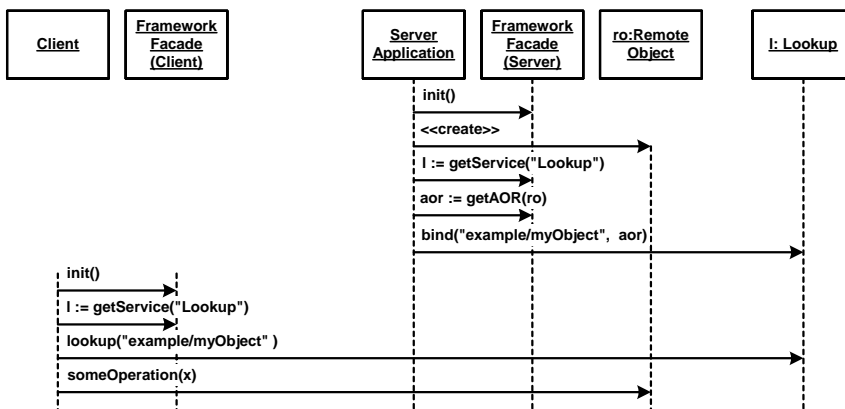
Pattern: Lookup (2)



49

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

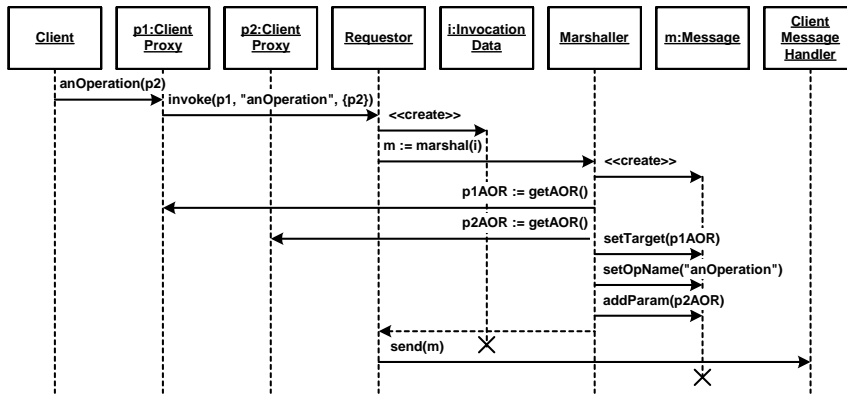
Interactions: Registration in Lookup



50

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Marshalling of Absolute Object References



51

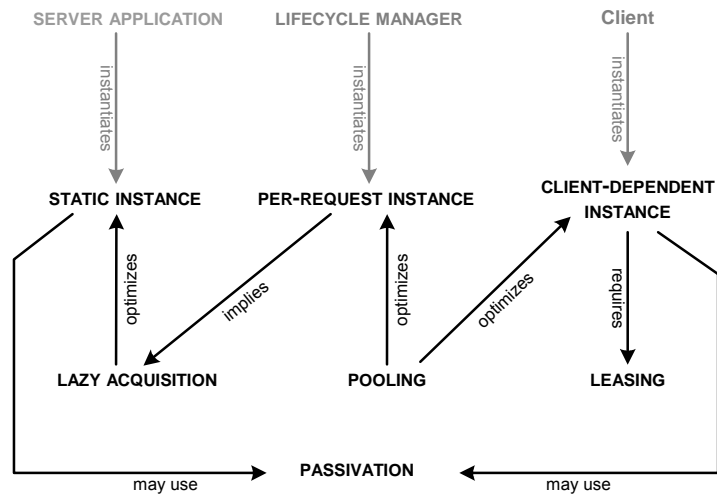
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Lifecycle Management Patterns

52

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Lifecycle Management Patterns



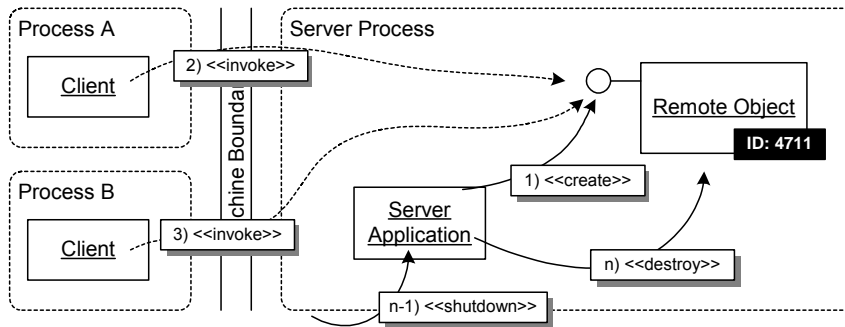
53 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Static Instance

- **Context:**
 - Remote object offers a service that is independent of specific clients
- **Problem:**
 - Fixed number of previously known remote objects instances
 - Remote objects must be available for a long period
 - No predetermined expiration timeout
 - Remote object's state must not be lost between individual invocations
- **Solution:**
 - Static instances are independent of the client's state and lifecycle
 - Activated before any client's usage, typically during application startup
 - Instances are registered in lookup after their activation

54 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Static Instance (2)



55

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

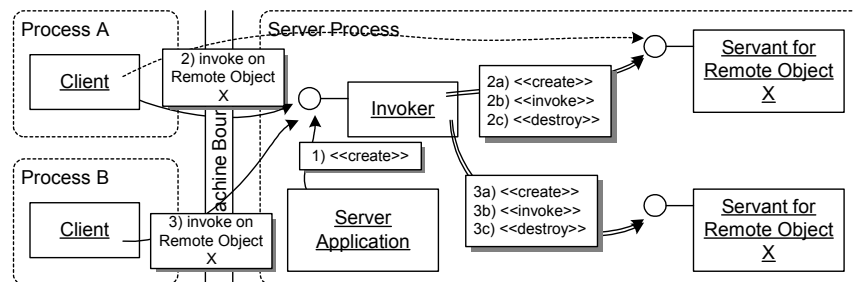
Pattern: Per-Request Instance

- **Context:**
 - Remote objects are stateless
- **Problem:**
 - Remote objects that are accessed by a large number of clients
 - Application needs to be scalable
 - Performance might decrease dramatically because of synchronization overhead
- **Solution:**
 - Let the distributed object framework instantiate a new servant for each invocation
 - Servant handles the request, returns the results, and is then deactivated again

56

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Per-Request Instance (2)



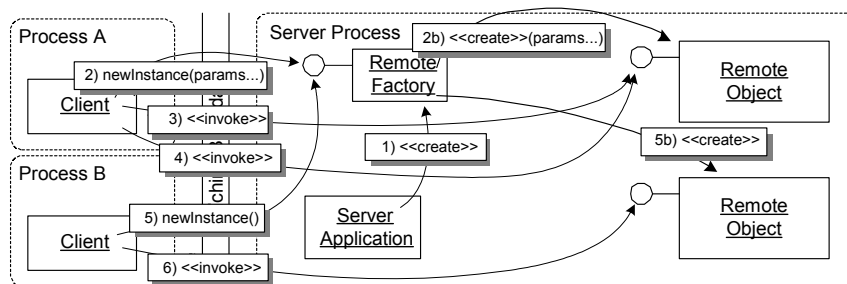
57 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Client-Dependent Instance

- **Context:**
 - Clients use services provided by remote objects
- **Problem:**
 - Remote object extends application logic of the client
 - Where to put the common state of both?
- **Solution:**
 - Provide remote objects whose lifetime is controlled by clients
 - Creation
 - Destruction
 - The client can consider the state of this instance to be private

58 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Client-Dependent Instance (2)



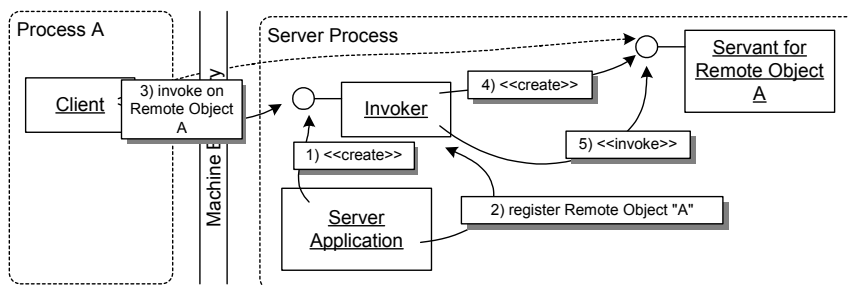
59 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Lazy Acquisition

- **Context:**
 - Static instances must be efficiently managed
- **Problem:**
 - Creating servants for all remote objects during server application startup can result in waste of resources
 - Instantiating all servants during server application startup leads to long startup times
- **Solution:**
 - Instantiate servants when they are invoked by clients
 - Let clients assume that remote objects are always available
 - Invoker triggers the lifecycle manager to lazily instantiate the servant

60 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Lazy Acquisition (2)



61

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

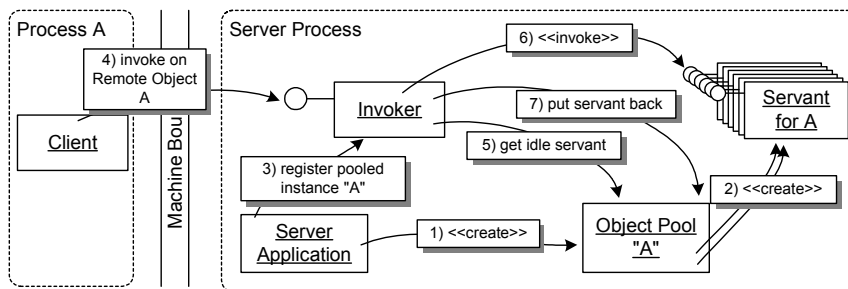
Pattern: Pooling

- **Context:**
 - Every remote object instance consumes server application resources
- **Problem:**
 - Instantiating and destroying servants causes a lot of overhead:
 - memory allocation
 - initializations
 - servants have to be registered with the middleware
- **Solution:**
 - Introduce a pool of servants for each remote object type
 - When an invocation arrives:
 - Take a servant from the pool
 - Initialize servant
 - Handle the request
 - Put it back to the pool

62

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Pooling (2)



63

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

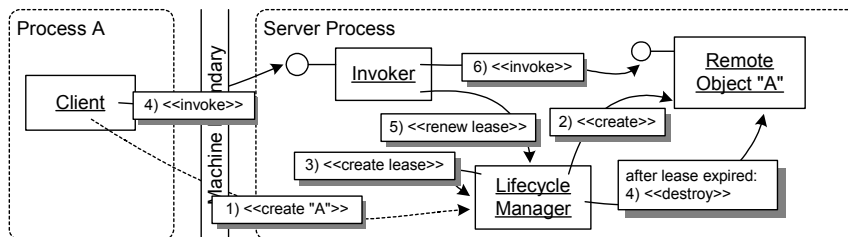
Pattern: Leasing

- **Context:**
 - Clients use server application resources
- **Problem:**
 - Resources no longer needed should be released
 - The lifecycle manager cannot determine when a particular remote object is not used anymore
- **Solution:**
 - Associate each client's use of a remote object with a lease
 - When the lease (or all leases) expire(s), the servant is destroyed by the lifecycle manager
 - Client can renew the lease
 - Call operation
 - Explicit renewal

64

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Leasing (2)



65

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

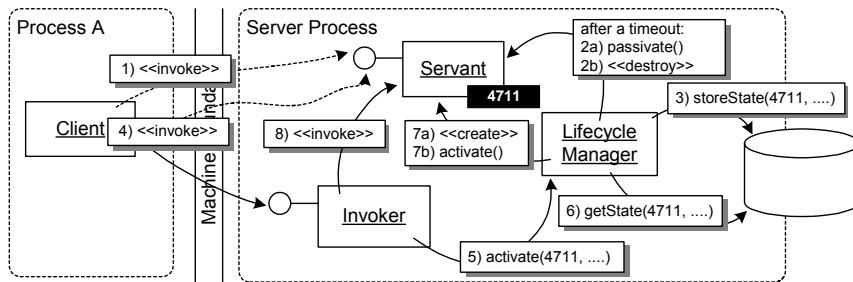
Pattern: Passivation

- **Context:**
 - The server application provides stateful remote objects
- **Problem:**
 - Remote objects may not be accessed by a client for a long time
 - Still their servants occupy server resources
 - Problems regarding performance and stability
- **Solution:**
 - Passivation:
 - Make remote objects – not accessed for a while – persistent
 - Then remove the unused servant from memory
 - Lifecycle Manager reactivates the object again upon the next invocation

66

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Passivation (2)



67

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

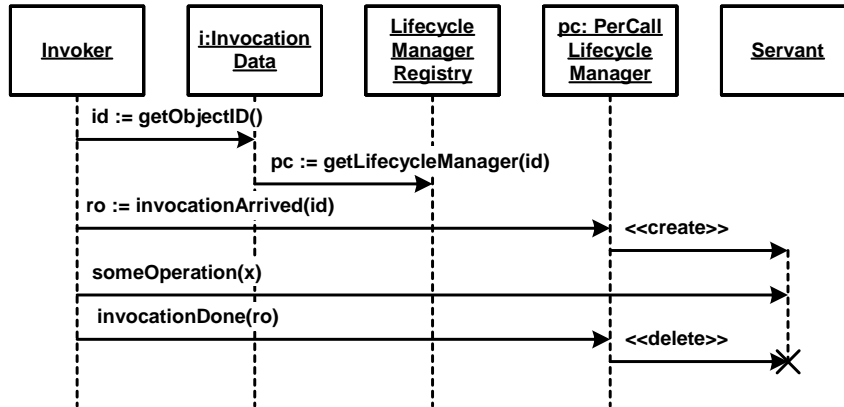
Interactions among the Patterns

	Static instance	Per-request instance	Client-dependent instance
Lazy acquisition	useful	implicitly useful	implicitly useful
Pooling	not useful	very useful	useful
Leasing	sometimes useful	not useful	very useful
Passivation	sometimes useful	not useful	very useful

68

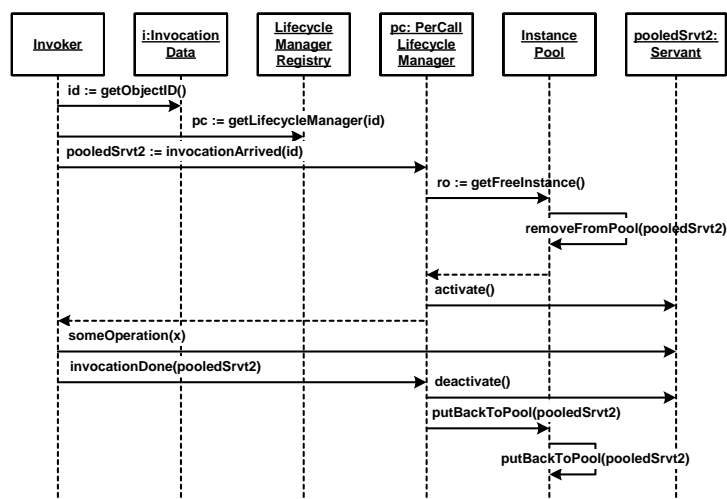
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Per-Request Instance



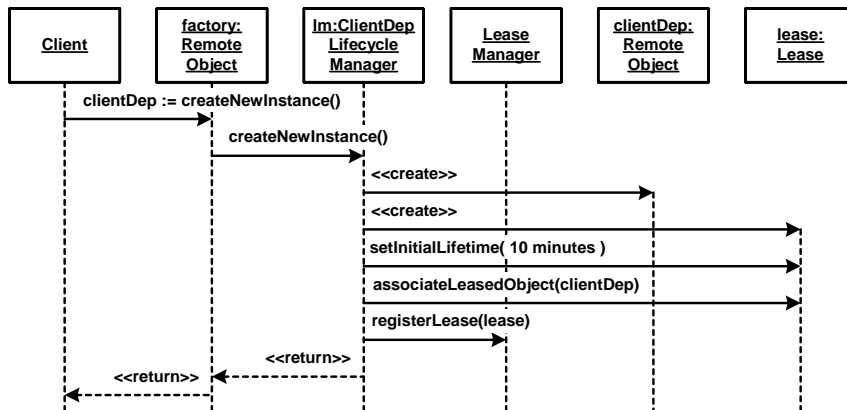
69 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Pooled Per-Request Instance



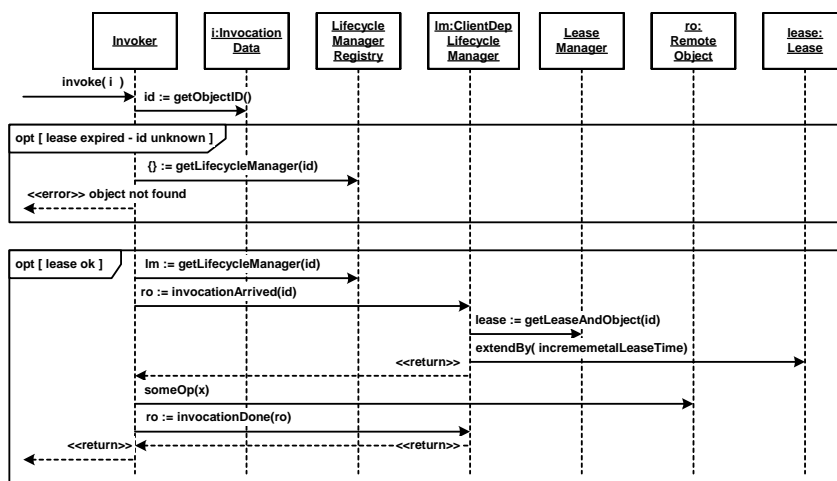
70 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Leased Creation



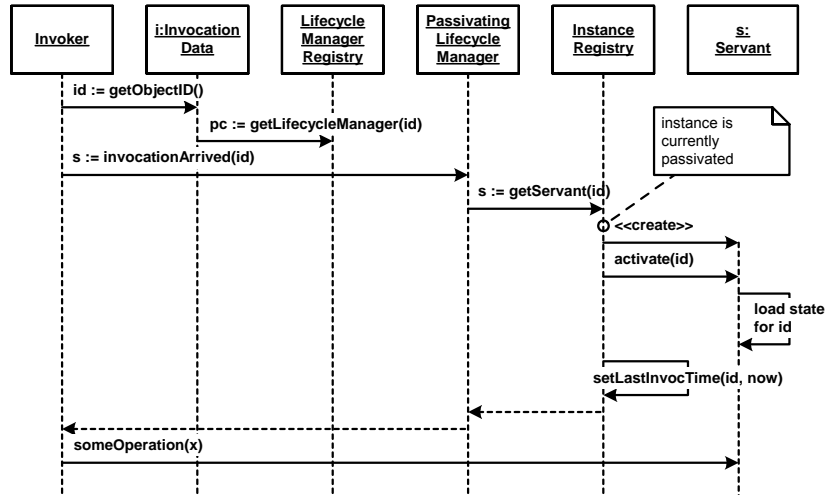
71 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Lease expires & Lease ok



72 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Passivation

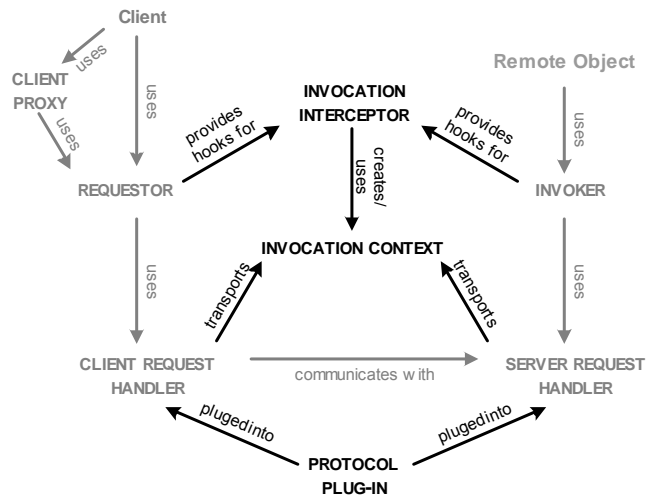


73 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Extension Patterns

74 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Extension Patterns



75

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

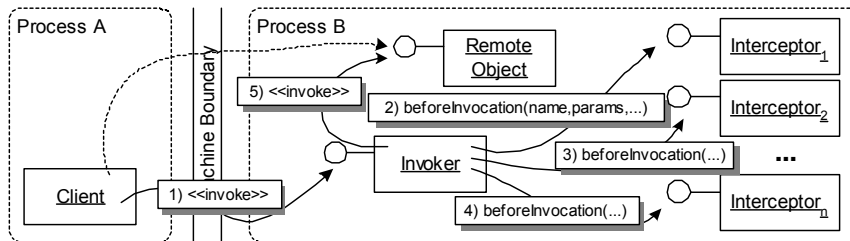
Pattern: Invocation Interceptor

- **Context:**
 - Applications need to transparently integrate add-on services
- **Problem:**
 - Client and server application have to provide add-on services, e.g.:
 - Transactions
 - Logging
 - Security
 - The clients and remote objects should be independent of add-ons
- **Solution:**
 - Hooks in the invocation path to plug in invocation interceptors
 - Invocation interceptors are invoked before and after each request and response message
 - Provide the interceptor with operation name, parameters, object id, and invocation context

76

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Invocation Interceptor (2)



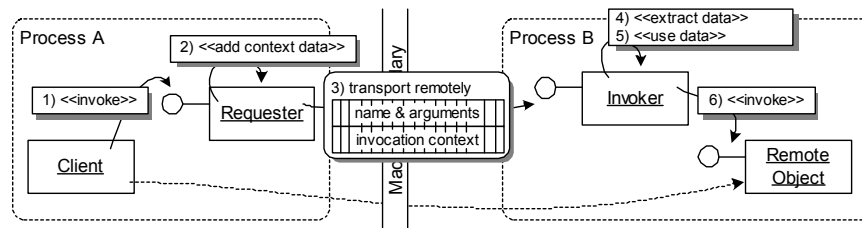
77 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Invocation Context

- **Context:**
 - Add-on services are plugged into the distributed object framework
- **Problem:**
 - Remote invocations only contain the necessary information, such as operation name, object id, and parameters
 - Invocation interceptors often need additional information in order to properly provide add-on services
 - Changing operation signatures for other reasons than business logic is tedious and error-prone
- **Solution:**
 - Bundle contextual information in an extensible invocation context
 - Invocation context is transferred between client and remote object with every remote invocation
 - Invocation interceptors can be used to add and consume this information

78 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Invocation Context (2)



79

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

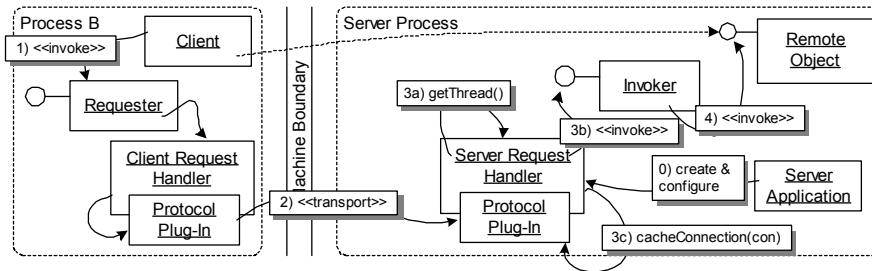
Pattern: Protocol Plug-In

- **Context:**
 - Client and server request handler adhere to the same protocol
- **Problem:**
 - The communication protocols should be configurable by developers, for instance because:
 - Multiple protocols need to be supported
 - Specialized protocols are needed
 - Existing protocols need to be optimized
- **Solution:**
 - Protocol plug-ins extend client request handlers and server request handlers
 - They provide a common interface to allow them to be configured from higher layers

80

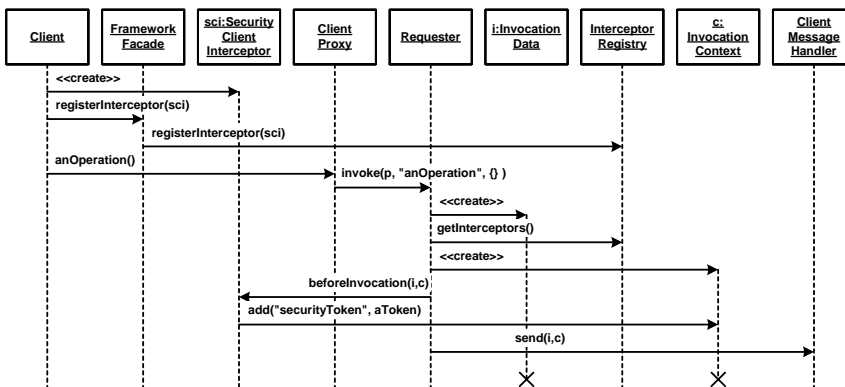
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Protocol Plug-In (2)



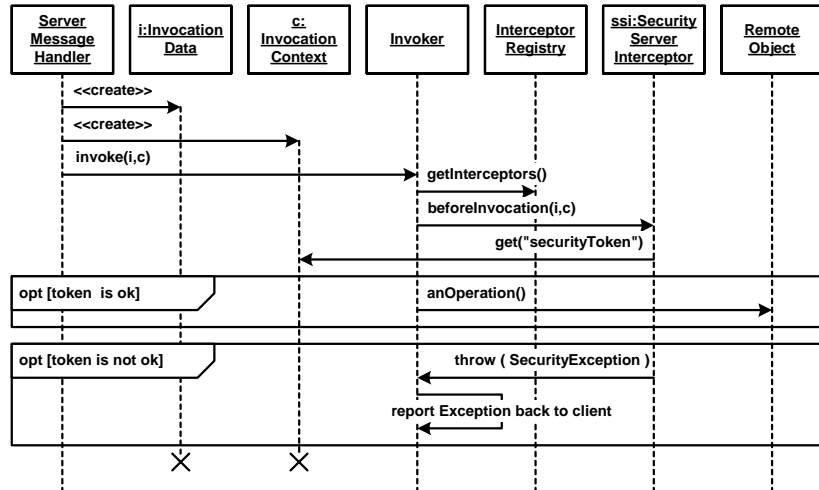
81 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Client-Side Security Interceptor



82 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Server-Side Security Interceptor

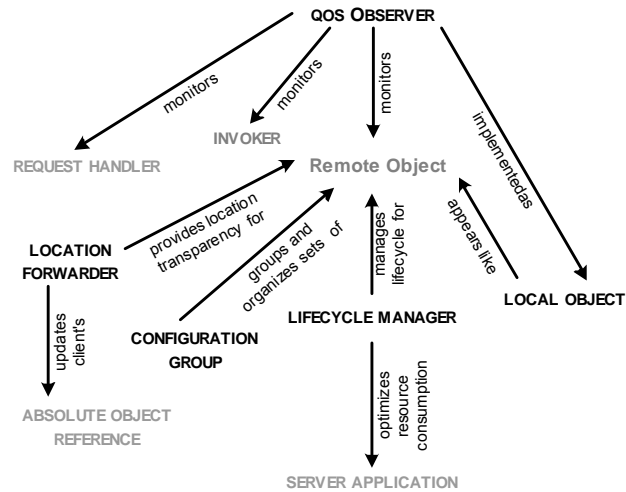


83 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Extended Infrastructure Patterns

84 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Extended Infrastructure Patterns



85

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

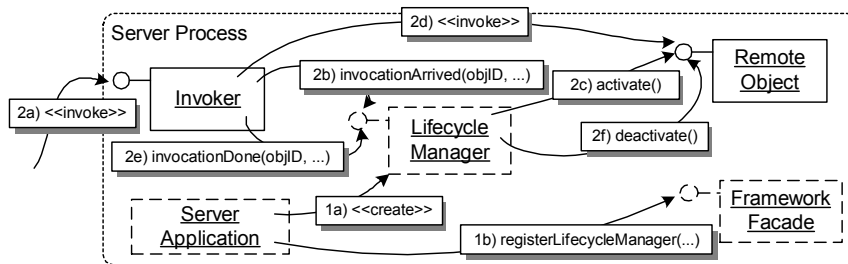
Pattern: Lifecycle Manager

- **Context:**
 - Different kinds of lifecycles of remote objects have to be managed
- **Problem:**
 - The lifecycle of remote objects needs to be managed by server applications
 - Based on configuration, usage scenarios, and available resources, servants have to be instantiated, initialized, or destroyed
 - All this has to be coordinated
- **Solution:**
 - Use a lifecycle manager to manage the lifecycle of remote objects
 - Lifecycle manager triggers lifecycle operations
 - Lifecycle manager implements configured lifecycle strategy

86

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Lifecycle Manager (2)



87

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

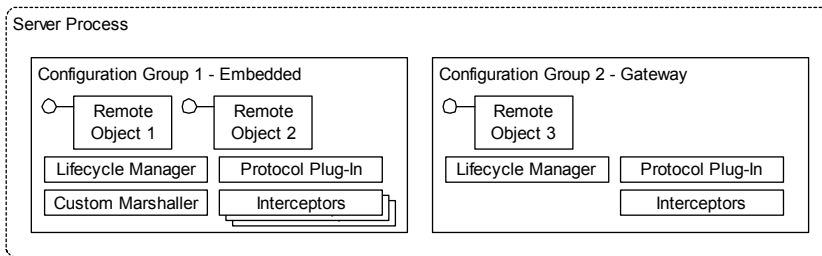
Pattern: Configuration Group

- **Context:**
 - Some remote objects require similar configurations
- **Problem:**
 - Remote objects need to be configured with various properties, such as:
 - quality of service (QoS)
 - Interceptor tasks
 - lifecycle management
 - protocol support.
 - Configuring per server application is not flexible enough
 - Configuring for each remote object leads to implementation overhead
- **Solution:**
 - Provide configuration groups which group remote objects with common properties

88

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Configuration Group (2)



89

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

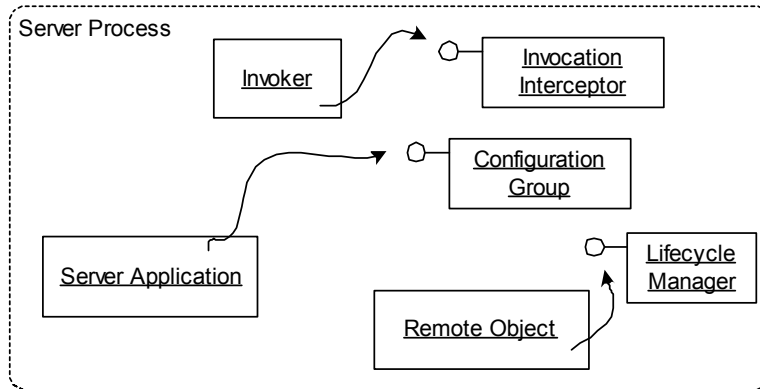
Pattern: Local Object

- **Context:**
 - Middleware constituents need to be configured
- **Problem:**
 - To configure protocol plug-ins, configuration groups, or lifecycle managers, interfaces are needed
 - These interfaces must not be accessible remotely
 - For consistency reasons, the interfaces should behave similarly as those of remote objects
- **Solution:**
 - Local objects allow application developers to access configuration and status parameters of the middleware
 - Local objects adhere to the same parameter passing rules, memory management, and invocation syntax as remote objects
 - They cannot be accessible remotely

90

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Local Object (2)



91

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

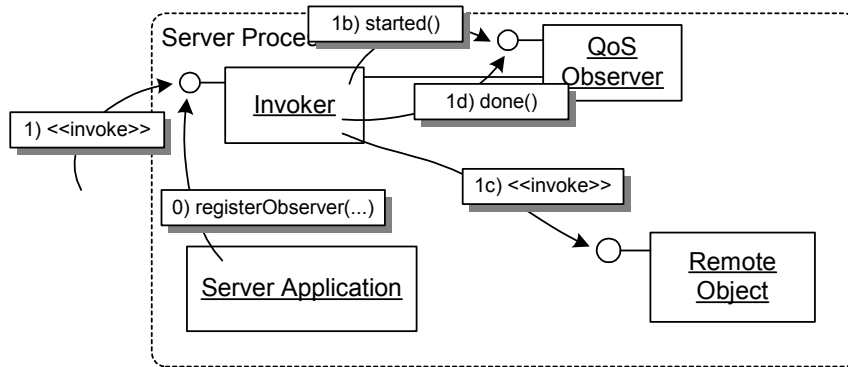
Pattern: QoS Observer

- **Context:**
 - You want to control application-specific quality of service properties, such as bandwidth, response times, or priorities
- **Problem:**
 - Request handlers,marshallers, protocol plug-ins, configuration groups, provide hooks to implement a variety of QoS characteristics
 - Applications might want to react on changes in QoS
 - The application-specific code to react on those changes should be decoupled from the framework itself
- **Solution:**
 - Provide hooks in the middleware constituents where application developers can register QoS Observers.
 - The observers are informed about relevant QoS parameters, such as message sizes, invocation counts, or execution times.
 - QoS observers contain code to react if service quality gets worse.

92

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: QoS Observer (2)



93

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

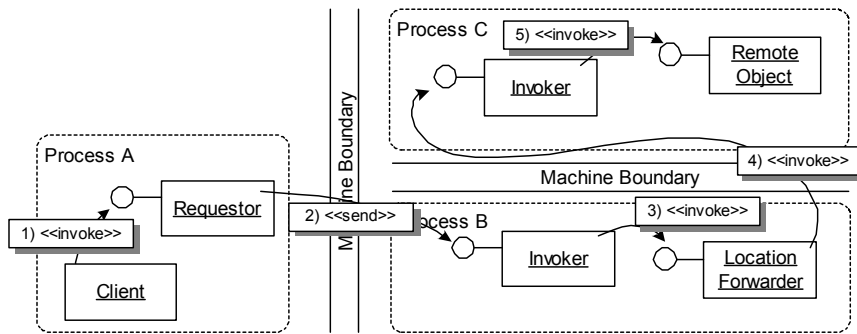
Pattern: Location Forwarder

- **Context:**
 - Invocations of remote objects arrive at the invoker to be dispatched
- **Problem:**
 - Remote objects might be located in a different server application than the one where the invocation arrives
 - Load balancing and fault tolerance
 - Remote objects were moved to other server applications
 - Invocations should transparently reach the correct remote object
- **Solution:**
 - A location forwarder can forward invocations to a remote object in other server applications
 - The location forwarder maps the object id to an absolute object references of another remote object

94

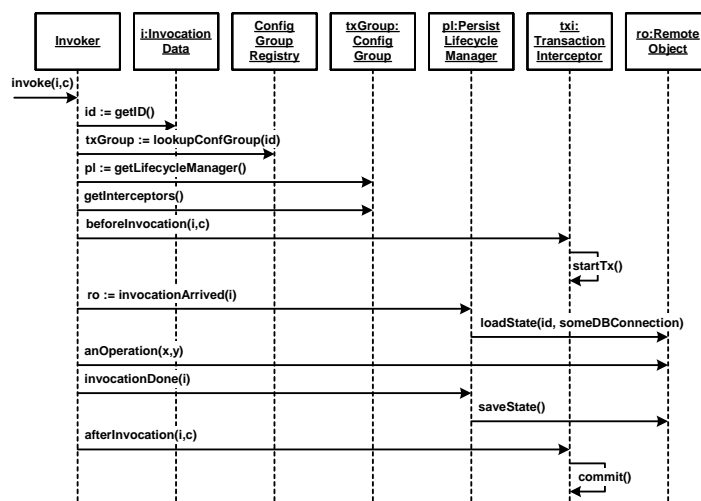
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Location Forwarder (2)



95 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Transaction Configuration Group

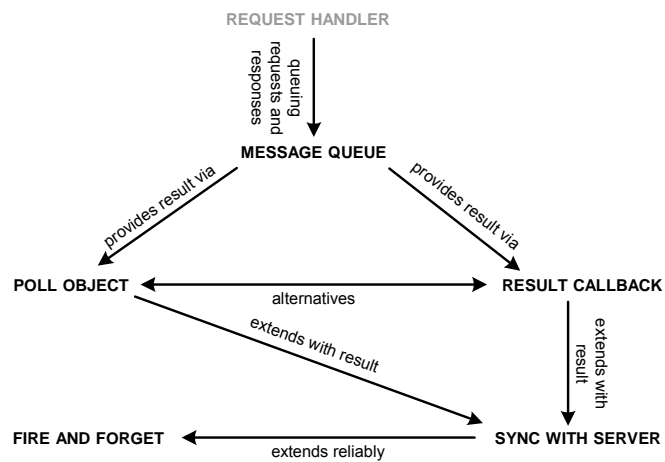


96 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Invocation Asynchrony Patterns

97 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Invocation Asynchrony Patterns



98 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

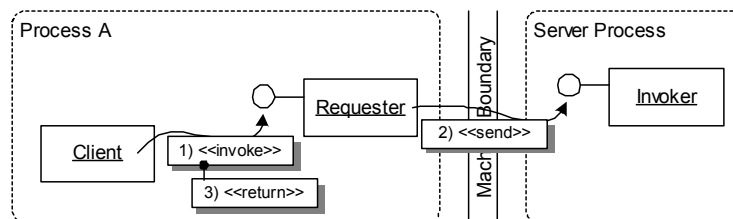
Pattern: Fire and Forget

- **Context:**
 - Operations have neither a return value nor report any exceptions
- **Problem:**
 - A client simply needs to notify the remote object of an event
 - The client does not expect any return value
 - Reliability of the invocation is not critical
- **Solution:**
 - For fire and forget operations, the requestor
 - sends the invocation across the network
 - returns control to the calling client immediately
 - The client does not get any acknowledgement from the remote object

99

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Fire and Forget (2)



100

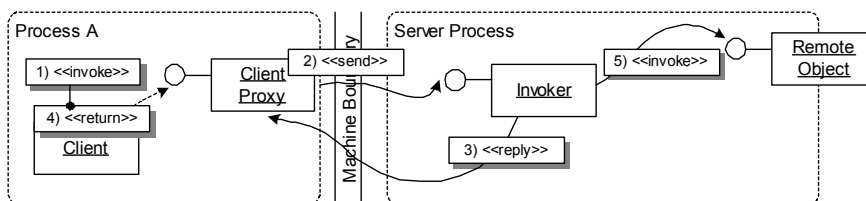
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Sync with Server

- **Context:**
 - Operations have neither a return value nor report any exceptions
- **Problem:**
 - Fire and forget is too unreliable
 - A synchronous call should not be used because it blocks the client until the server responds
- **Solution:**
 - Provide sync with server semantics for remote invocations
 - The client sends the invocation and waits for a reply from the server application informing it about the successful reception of the invocation
 - After the reply is received by the requestor, it returns control to the client and execution continues
 - The server application independently executes the invocation

101 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Sync with Server (2)



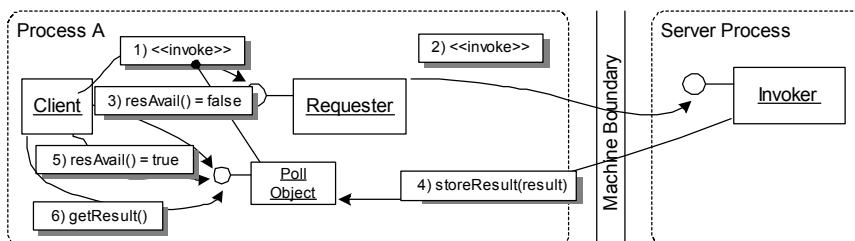
102 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Poll Object

- **Context:**
 - Invocations should execute asynchronously and clients depend on the results
- **Problem:**
 - The client does not necessarily need the results immediately to continue its execution
 - It can decide for itself when to use the returned results
- **Solution:**
 - Provide poll objects that receive the result of remote invocations on behalf of the client
 - The client subsequently uses the poll object to query the result:
 - query (“poll”) whether the result is available or
 - block on the poll object until the result becomes available
 - As long as the result is not available on the poll object, the client can continue with other tasks asynchronously.

103 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Poll Object (2)



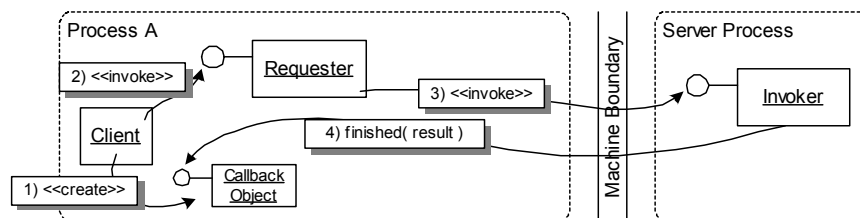
104 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Result Callback

- **Context:**
 - Invocations should execute asynchronously and clients depend on the results
- **Problem:**
 - If the result becomes available to the requestor, the client wants to be informed immediately to react on it
 - In the meantime the client executes concurrently.
- **Solution:**
 - Provide a callback-based interface for remote invocations on the client
 - The client passes a result callback object to the requestor
 - The invocation returns immediately after sending the invocation
 - Once the result is available, the distributed object framework invokes a predefined operation on the result callback object

105 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Result Callback (2)



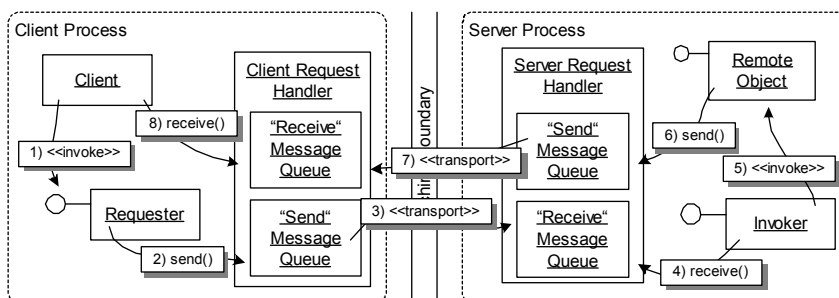
106 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Message Queue

- **Context:**
 - The result of the invocation can be handled asynchronously
- **Problem:**
 - Poll object and result callback require a separate process or thread for receiving the result asynchronously
 - Perhaps you want to deal with (temporal) problems of the networked environment
 - None of the synchronous or asynchronous invocation variants can handle the order of invoking or receiving messages.
- **Solution:**
 - Add message queues to the request handlers (or protocol plug-ins)
 - Perhaps as part of a larger Messaging system

107 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Pattern: Message Queue (2)



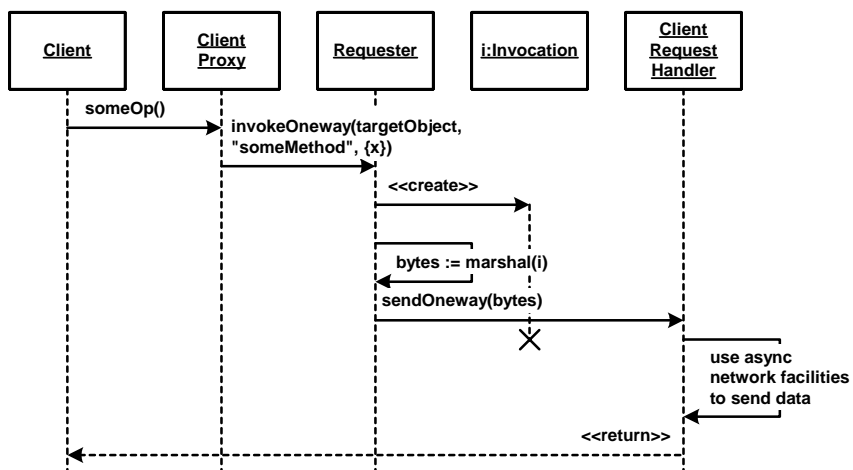
108 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Relations among the Patterns

	Acknowledgement to client	Result to client	Responsibility for result
Fire and forget	No	No	-
Sync with server	Yes	No	-
Poll object	Yes	Yes	Client gets the result
Result callback	Yes	Yes	The client is informed via callback
Message Queue	Yes	Yes	Sync, Poll Object or Result Callback

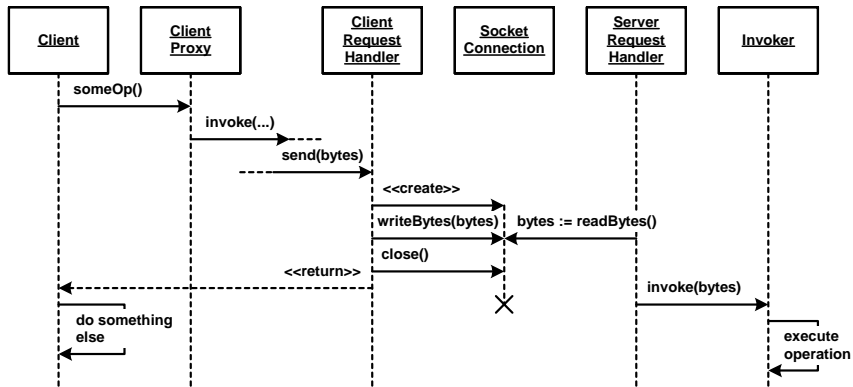
109 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Fire and Forget



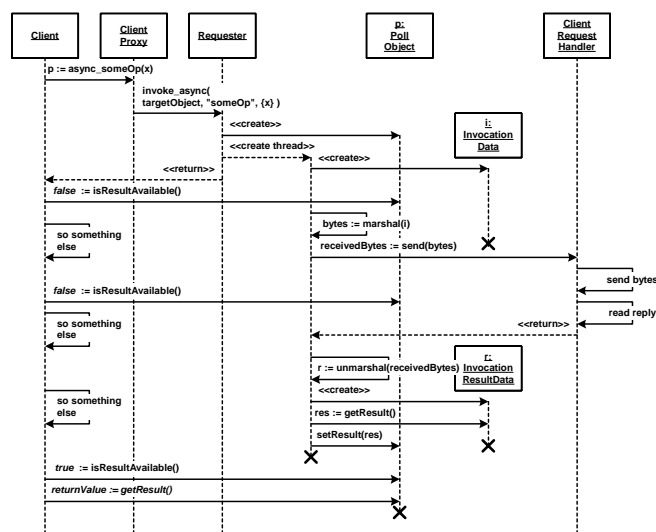
110 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Sync with Server



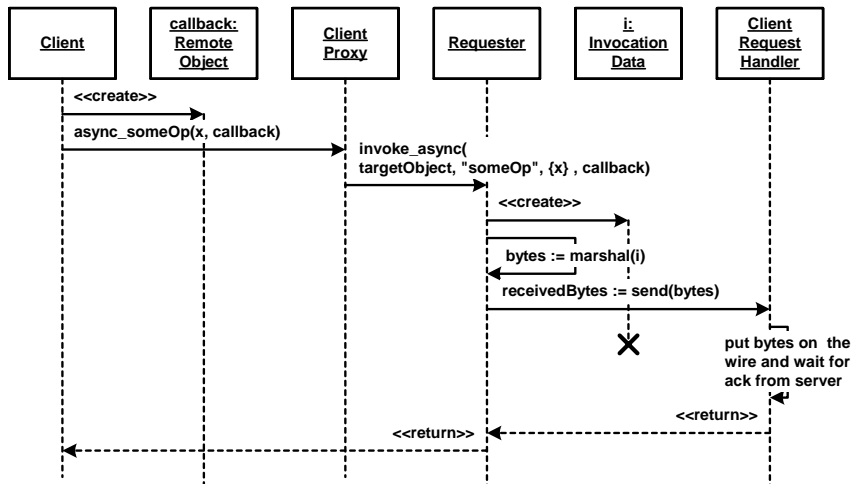
111 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Poll Object



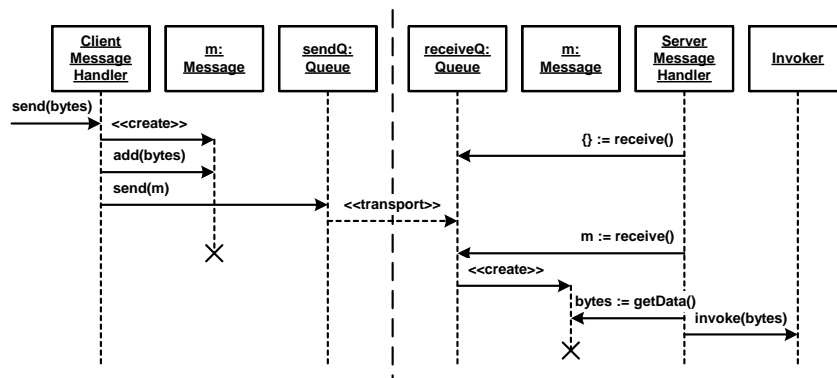
112 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Result Callback



113 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Interactions: Message Queue

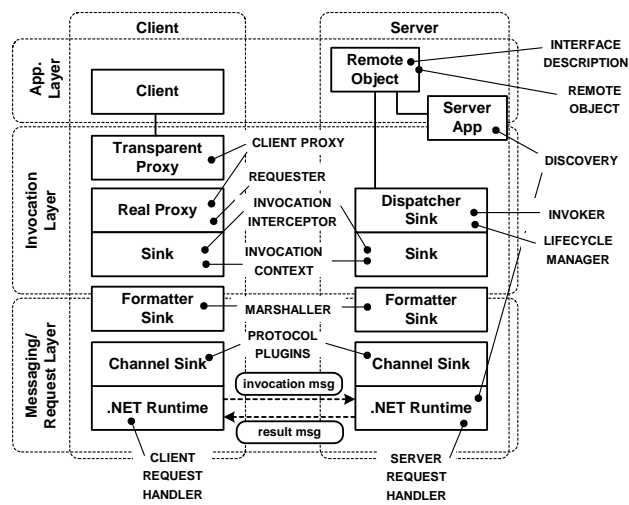


114 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Technology Projections

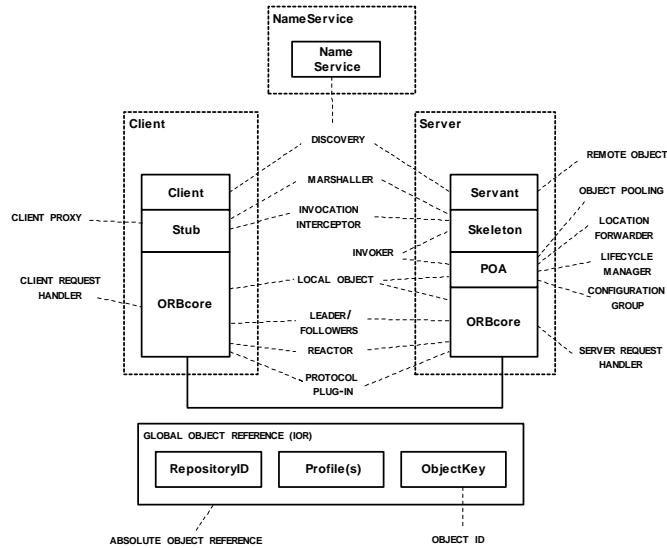
115 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

.NET Remoting Technology Projection



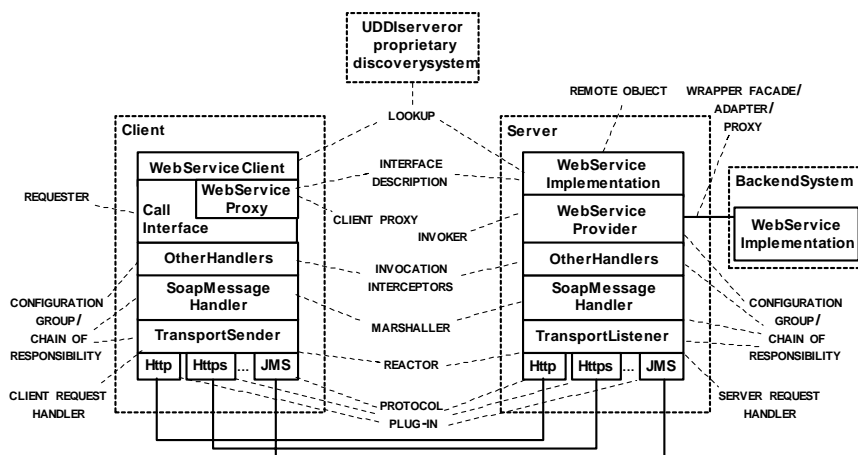
116 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

CORBA Technology Projection



117 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Web Services Technology Projection



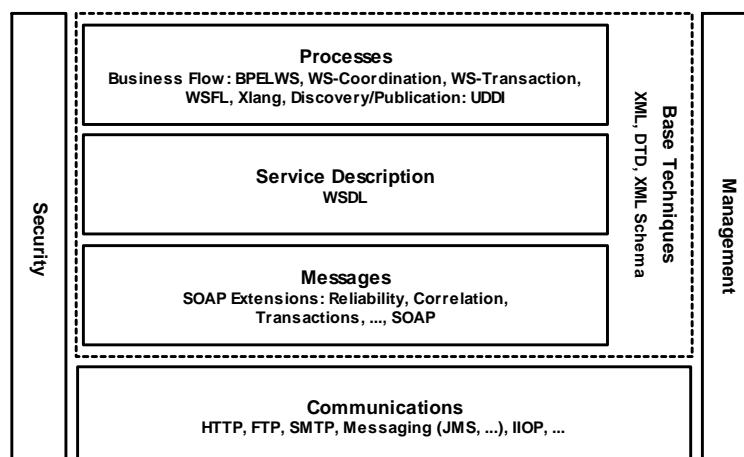
118 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Web Services

- **Definition:**
 - *A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network.*
 - *It has an interface described in a machine-processable format (specifically WSDL).*
 - *Other systems interact with the Web Service in a manner prescribed by its description using messages (specifically SOAP messages).*
 - *Messages are typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards, but any other communication protocol can be used for message delivery as well.*
- **Here: Apache Axis**

119 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Web Service Stack



120 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Dynamic Invocation on Server Side

Ordinary Java class as Remote Object:

```
public class DateService {
    public String getDate (String format) {...}
    public String getDate () {...}
}
```

Invoker is set up by XML configuration file (used for dynamic deployment):

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="DateService" provider="java:RPC">
    <parameter name="className"
      value="simpleDateService.DateService"/>
    <parameter name="allowedMethods" value="getDate"/>
  </service>
</deployment>
```

121 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Dynamic Invocation on Server Side (2)

- Service name is used as Object ID: **DateService**
- Per default: Per-Request Instance as activation strategy
- URL as Absolute Object Reference, e.g.:

`http://localhost:8080/axis/services/DateService`

- **Invoker performs mapping and invocation**
 - In the SOAP request the operation name and parameters are encoded
 - Before invocation:
 - check that invocation is an "allowed method"
 - check that the parameter number and types are valid
 - If so: return result as SOAP message
 - Otherwise: SOAP Remoting Error is returned to the client

122 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Constructing an Invocation with a Requestor on Client Side

```

Service service = new Service();
Call call = (Call) service.createCall();

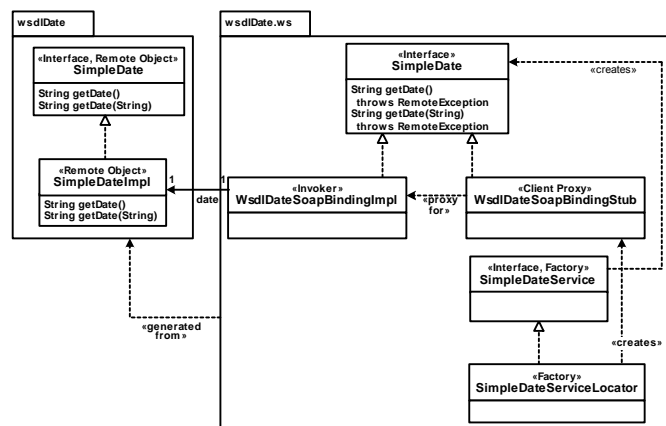
call.setTargetEndpointAddress(
    new java.net.URL(endpointURL));
call.setOperationName("getDate");
call.addParameter( "format",
    XMLType.XSD_STRING,
    ParameterMode.IN);
arguments = new Object[] { formatString };
call.setReturnType(
    org.apache.axis.encoding.XMLType.XSD_STRING);
String result =
    (String) call.invoke( arguments );
System.out.println("Date: " + result);

```

123 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Generating Invoker and Client Proxy Code with WSDL

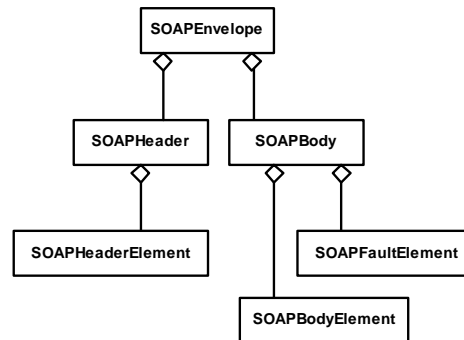
From an WSDL Interface Description we can generate Client Proxy and Invoker code:



124 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Marshalling

- **Conversion to SOAP**
 - Type conversion to/from strings, XSD types, ...
 - Automatic type conversion for standard types



125

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

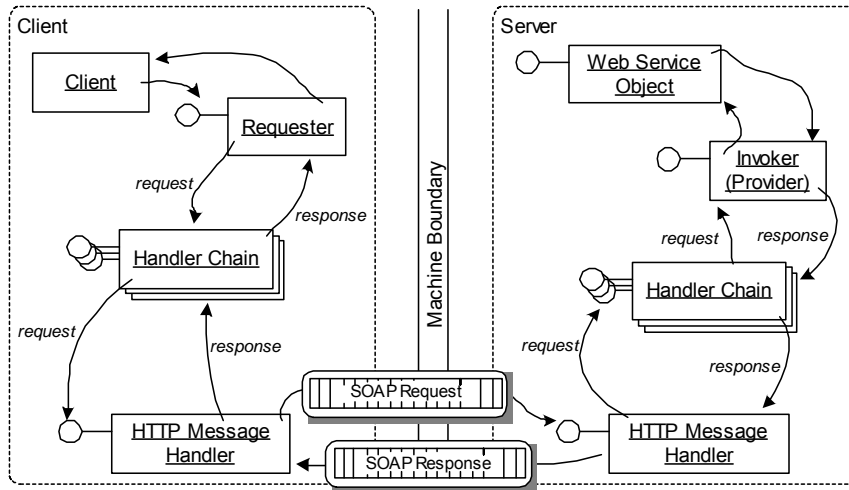
Message Processing

- **On client and server side:**
 - there are many different, orthogonal tasks to be performed for a message,
 - there is a symmetry of the tasks to be performed for request and response,
 - similar problems occur on client side and server side, and
 - the invocation scheme and add-ons have to be flexibly extensible.
- **Combination of the patterns**
 - requestor,
 - invoker,
 - invocation context,
 - invocation interceptor, and
 - client/server request handler
- **Interceptors as Commands are ordered in a Chain of Responsibility**

126

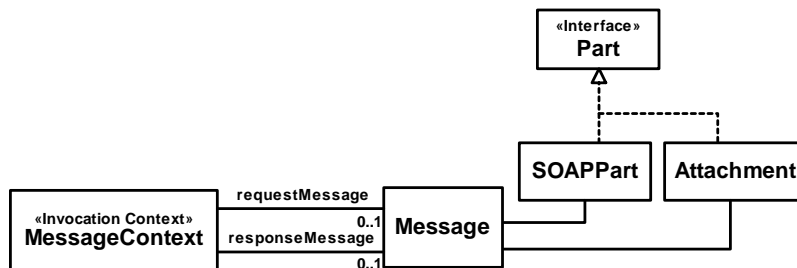
Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Message Processing (2)



127 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Message Context



128 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Example: Log Handler

```
public class LogHandler extends BasicHandler {
    ...
    public void invoke(MessageContext msgContext)
        throws AxisFault {
        ...
        if (msgContext.getPastPivot() == false) {
            start = System.currentTimeMillis();
        } else {
            logMessages(msgContext);
        }
        ...
    }
    ...
}
```

129 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Configuration Groups in Deployment Descriptors

```
<handler
  name="logger"
  type="java:org.apache.axis.handlers.LogHandler"/>
...
<chain name="myChain"/>
  <handler type="logger"/>
  <handler type="authentication"/>
</chain>
...
<service name="DateService" provider="java:RPC">
  ...
  <requestFlow>
    <handler type="myChain"/>
  </requestFlow>
</service>
```

130 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Protocol Integration

- **Heterogeneity of communication protocols of Web Service frameworks**
 - Most Web Service frameworks provide for some extensibility at this layer
 - Slightly different request handler/protocol plug-in architectures
- **In the default case HTTP is used as a communication protocol**
- **SOAP also allows for other communication protocols**
 - Implemented with protocol plug-ins
 - Same invoker can be used for all protocols
- **Axis supports protocol plug-ins for HTTP, Java Messaging Service (JMS), SMTP, and local Java invocations**
- **Protocol plug-ins are responsible for implementing a message queue, if needed (e.g. JMS-based messaging)**

131 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Lifecycle Handling and Identification

- **Axis supports the following lifecycle patterns using a scope option chosen in the deployment descriptor**
 - Per-Request Instance: Default, "request" scope
 - Static instance: "application" scope
 - Client-dependent instance: "session" scope,
 - Sessions are supported either by HTTP cookies or by - communication protocol independent - SOAP headers
 - Each session object has a timeout (which can be set to a certain amount of milliseconds). After the timeout expires, the session is invalidated. A method touch can be invoked on the session object, which renews the lease.

132 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Client-Side Asynchrony

- **Axis does not support client-side asynchrony patterns without using a messaging protocol**
- **Simple Asynchronous Invocation Framework for Web Services (SAIWS)**
 - Asynchrony layer on top of synchronous invocation layer provided by Axis
 - `http://saiws.sourceforge.net`

133 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

SAIWS - Requestors

Two kinds of requestors one for synchronous invocations:

```
SyncRequestor sr = new SyncRequestor();
String result =
    (String) sr.invoke(endpointURL, operationName,
                      null, rt
```

... and one for asynchronous invocations:

```
AsyncHandler ah = ...;
Object clientACT = ...;
AsyncRequestor ar = new AsyncRequestor();
ar.invoke(ah, clientACT, endpointURL, operationName,
          null, rt);
// ... resume work
```

134 Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

SAIWS - Example: Result Callback Invocation

```
AsyncRequestor requestor = new AsyncRequestor();
ResultCallback r = (ResultCallback) new
    ResultCallbackQueue();
for (int i = 0; i < 10; i++) {
    String id = "invocation" + i;
    requestor.invoke(r, id, endpointURL,
        operationName,
        null, rt);
}
```

137

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Lookup of Web Services: UDDI

- **Many possible ways to realize lookup with Web Services**
- **UDDI is an automated directory service that allows one to register services and lookup services**
 - All UDDI specifications use XML to define data structures
 - An UDDI registry includes four types of documents:
 - A business entity is a UDDI record for a service provider.
 - A business service represents one or more deployed Web Services.
 - A technical model (tModel) can be associated with business entities or services.
 - A binding template binds the access point of a Web Service and its tModel.
 - UDDI allows a service provider to register information about itself and the services it provides

138

Uwe Zdun, Markus Voelter, Michael Kircher - Remoting Patterns.

Other Web Service Frameworks Discussed

- **GLUE is a commercial Web Service implementation in Java that is optimized for ease-of-use and performance**
- **Microsoft's .NET Web Services:**
 - .NET remoting
 - ASP.NET Web Services
- **IONA's Artix**
- **Tcl SOAP**

Thanks for your attention!