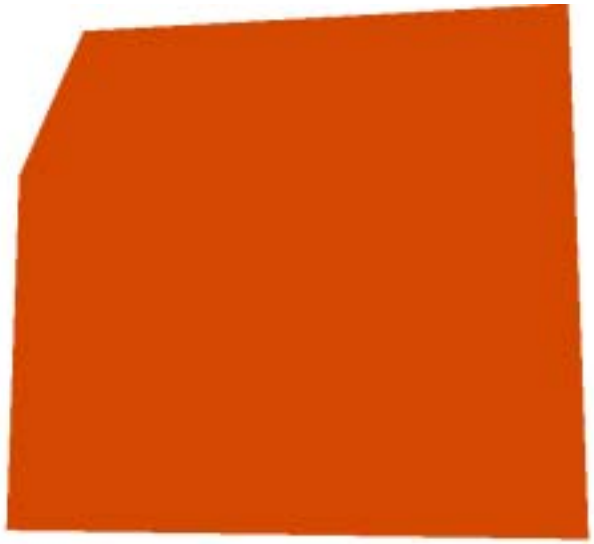


TECHNOLOGY CONSULTING INNOVATION

ELCA

ELCA

Interoperability with IIOP.NET

Agenda

Introduction

- What / Why IIOP.NET?

Interoperability

- Web Services vs. Remoting
- Type System Issues

IIOP.NET Tools

- Channel
- IDL Generator
- IDL Compiler

IIOP.NET Examples

- .NET to .NET
- .NET to J2EE
- .NET to CORBA

Conclusions

- Performance
- IIOP.NET State
- IIOP.NET Future

What is IIOP.NET?

IIOP.NET is an Open Source project (LGPL) developed by ELCA to connect .NET to J2EE/CORBA

IIOP.NET (marketing)

- *„Provide seamless interoperability between .NET and CORBA-based peers (including J2EE)“*

IIOP.NET (technical)

- .NET Remoting channel implementing the CORBA IIOP protocol
- Compiler to make .NET stubs from IDL definitions
- IDL definition generator from .NET metadata

Why IIOP.NET?

Reasons

- many EJB-based solutions
- interest in making .NET-based GUIs
- expected bi-polar world for distributed applications (.NET and J2EE)
- (ELCA only) bring together own Java and .NET frameworks
- provide a solution at Remoting / RMI (Component) level instead of WebService (Service) level
- Other solutions (at project start) were not appropriated
  - unsuitable abstraction (like WebServices)
  - unsuitable design (like JaNET)
  - missing piece in interoperability puzzle

Agenda

Introduction
- What / Why IIOP.NET?

Interoperability
- Web Services vs. Remoting
- Type System Issues

IIOP.NET Tools
- Channel
- IDL Generator
- IDL Compiler

IIOP.NET Examples
- .NET to .NET
- .NET to J2EE
- .NET to CORBA

Conclusions
- Performance
- IIOP.NET State
- IIOP.NET Future

Interoperability

Interoperability

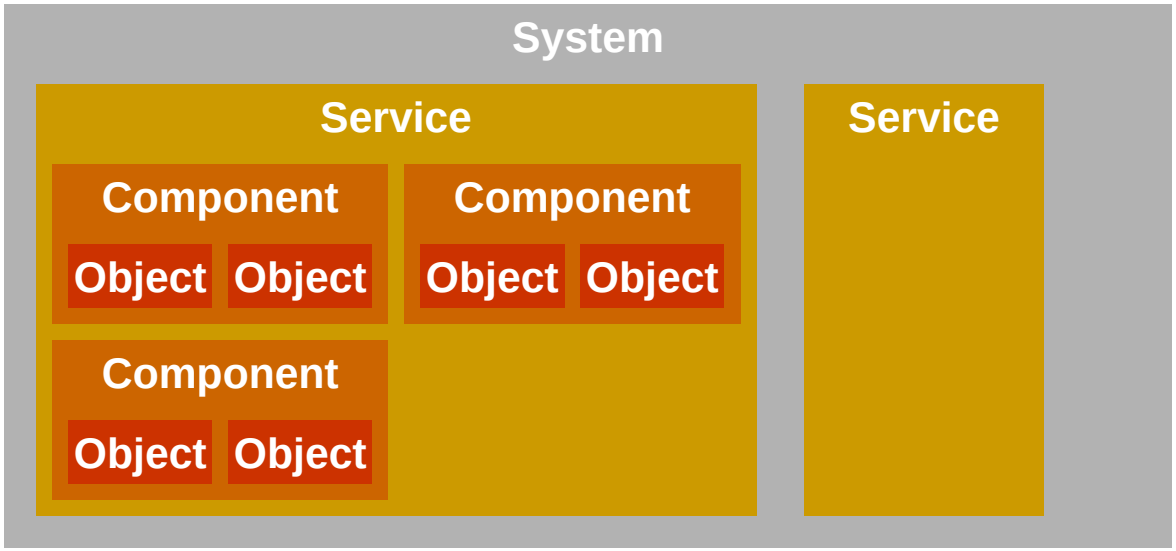
Interoperability is the ability of two or more software components to cooperate despite differences in language, interface, and execution platform. (P.Wegner)

Interoperability heavily depends on
the abstraction or context
you are working with....

Which Interoperability Level ?

Application	This is what we want, or is it ?
Services	Distributed Transaction Coordinator, Active Directory, …
Conversation	Activation model (EJB, MBR), global naming, distributed garbage collection, conversational state,…
Contextual Data Interception Layer	SessionID, TransactionID, cultureID, logical threadID …
Message Format	RPC, IIOP, HTTP, SOAP, proprietary binary format, messages, unknown data (exceptions), encryption
Data Model	Type system, mapping and conversion issues
Communication Protocols	TCP/UDP, Byte stream, point-to-point communication .NET's TCP channel transport header issue

Interop. Granularity and Level



- If interoperability granularity ↘, interaction complexity ↗, common type subset ↗ and required interoperability level ↗.

Granularity	**Service**	**Component**	**Object**
Coupling, Interaction	**Message-based Interface, Stateless**	**Strongly-typed Interface, Stateless or Stateful**	**Implementation Dependency, Stateful**

Loose Coupling in .NET

- Web Service
  - `XmlElement ProcessXml(XmlElement xe)`
  - The type existence is optional at both client and server sides (`[XmlAnyElement]`, `[XmlAnyAttribute]`, literal only)
  - More flexible, less typed.
  - Allows ignoring unknown parts of messages.
  - Type (XSD) information can be included in instances (XML)

- MarshalByRefObject, EnterpriseJavaBean
  - `Object ProcessObject(Object o)`
  - The type existence is mandatory at client and at server side (issue with unknown exceptions, and unknown contextual data)
  - Less flexible, strongly typed.
  - Libraries must be synchronized.

.NET Serializations

- XmlSerialization
  - Used by ASP.NET for Web Services method parameters
  - Objects attributed with `[Xml/SoapType]`
  - Basic customization with `[Xml/SoapElement]`, `[Xml/SoapAttribute]`, `[Xml/SoapIgnore]`, `[XmlAnyElement]`, `[XmlAnyAttribute]`
  - Advanced customization with `IXmlSerialization`

- (Object) Serialization
  - Used by the Remoting for MBR method parameters
  - Objects attributed with `[Serializable]`
  - Basic customization with `[NonSerializable]`
  - Advanced customization with `ISerializable` interface

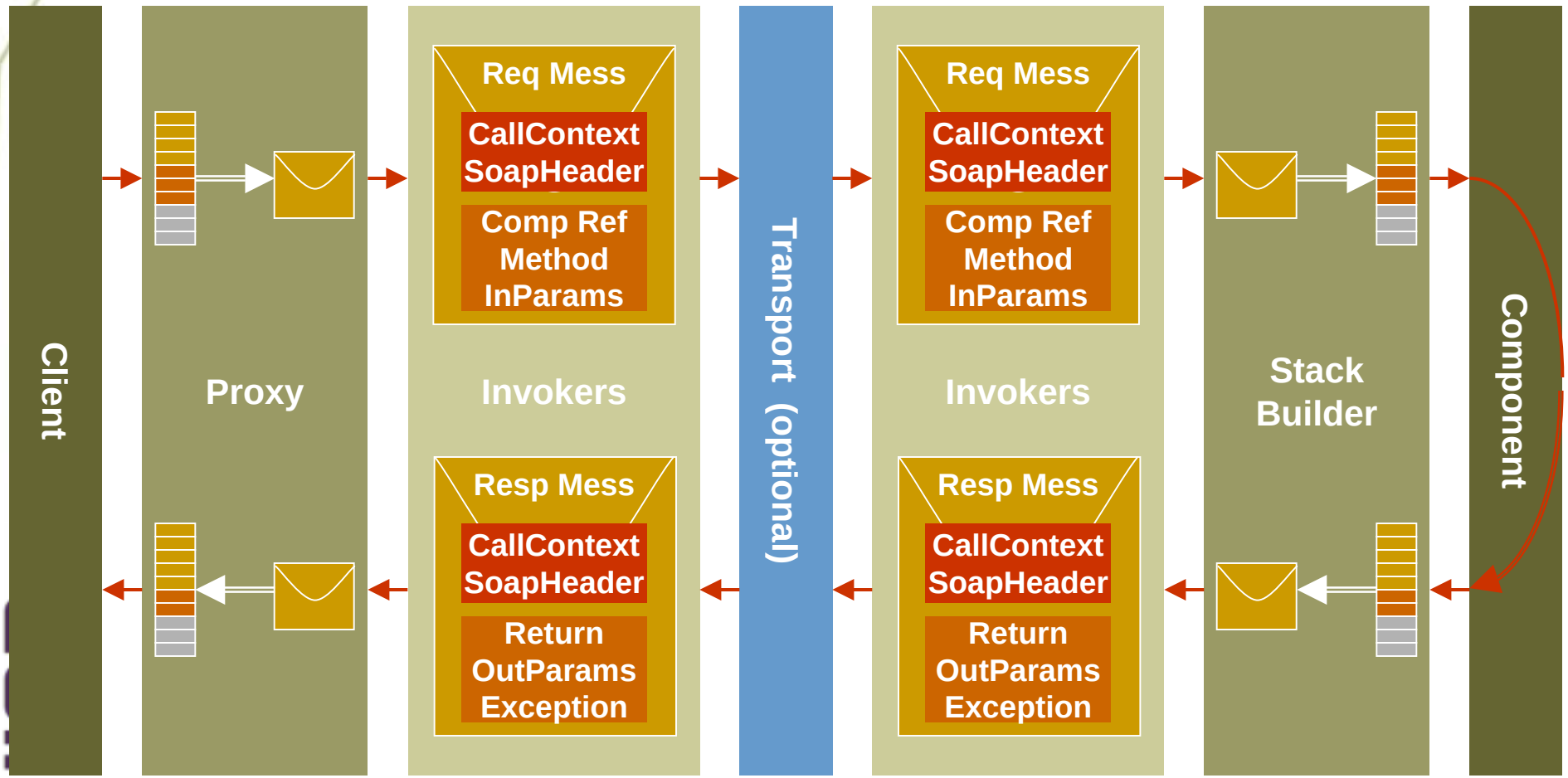
.NET XML Serialization

- Information Level
  - Structural only, semantic loss (methods), requires mapping
  - External view of object (public fields and read-write properties)
- Literal & Encoded formats
  - Have no dynamic polymorphism
  - Have no remote references
- Literal Format
  - Schema-based type system
  - Reference equality loss
  - Untyped
  - Cycles forbidden
- (SOAP) Encoded Format
  - Soap encoding-based Type System
  - Keep reference equality
  - Strongly typed

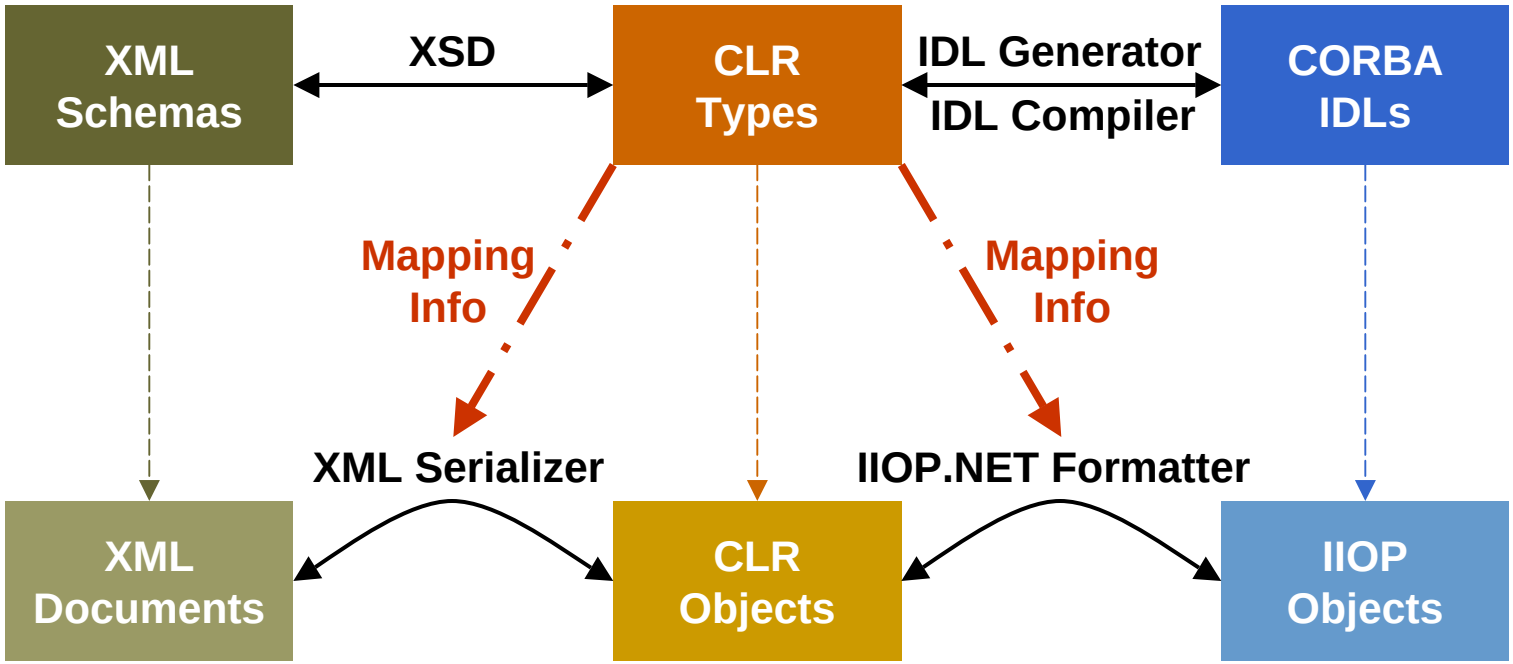
.NET (Object) Serialization

- Information Level
  - Structural and semantic (methods)
  - Internal state of object (public and private fields)
- SOAP, Binary, and IIOP.NET Formatters
  - Keep reference equality
  - Have a dynamic polymorphism
  - Have remote references
  - Have strong typing
- SOAP (-like) Formatter
  - CLR Type System
- Binary Formatter
  - CLR Type System
- IIOP.NET Formatter
  - IDL Type System
  - Semantic (methods) loss, requires mapping

Interception and Contextual Data



CLR Metadata-driven Conversions



Which Interoperability Level ?

Application	This is what we want, or is it ?
Services	Distributed Transaction Coordinator, Active Directory, …
Conversation	Activation model (EJB, MBR), global naming, distributed garbage collection, conversational state,…
Contextual Data Interception Layer	SessionID, TransactionID, cultureID, logical threadID …
Message Format	RPC, HTTP, SOAP encoding variants, proprietary binary format, messages, unknown data (exceptions), encryption
Data Model	Type system, mapping and conversion issues
Communication Protocols	TCP/UDP, Byte stream, point-to-point communication .NET's TCP channel transport header issue

Conclusions

- Web Services do not address all interoperability issues.
- Web Services target service granularity, IIOP.NET targets component one.
- IIOP.NET allows reaching more existing EJBs than if they were published as Web Services.
- Coupling (loose or tight) is a technical matter, not a faith matter.
- In both cases, only the object structure remains, and some additional mappings have to be handled by hand (ArrayList, HashTable for Java ↔ .NET, contextual data).
- Open and widely-adopted model for contextual data (CallContext) and interception layer (Invokers) is not yet considered as an important step for application interoperability.

Agenda

Introduction
- What / Why IIOP.NET?

Interoperability
- Web Services vs. Remoting
- Type System Issues

IIOP.NET Tools
- Channel
- IDL Generator
- IDL Compiler

IIOP.NET Examples
- .NET to .NET
- .NET to J2EE
- .NET to CORBA

Conclusions
- Performance
- IIOP.NET State
- IIOP.NET Future

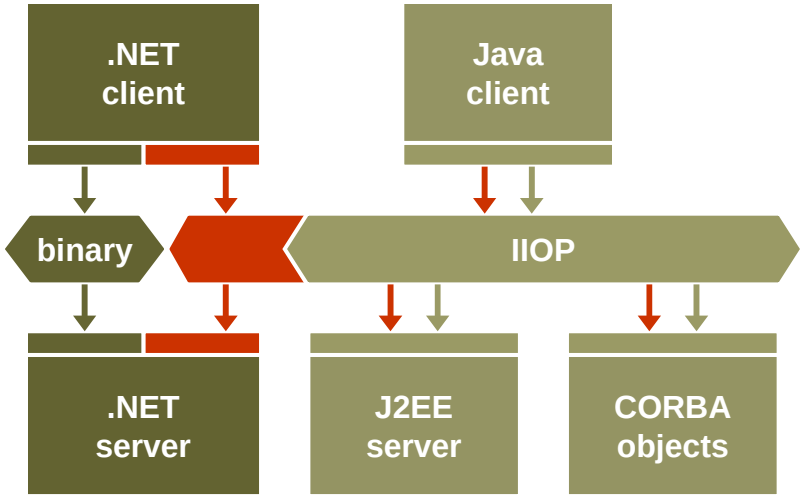
IIOP.NET: Overview

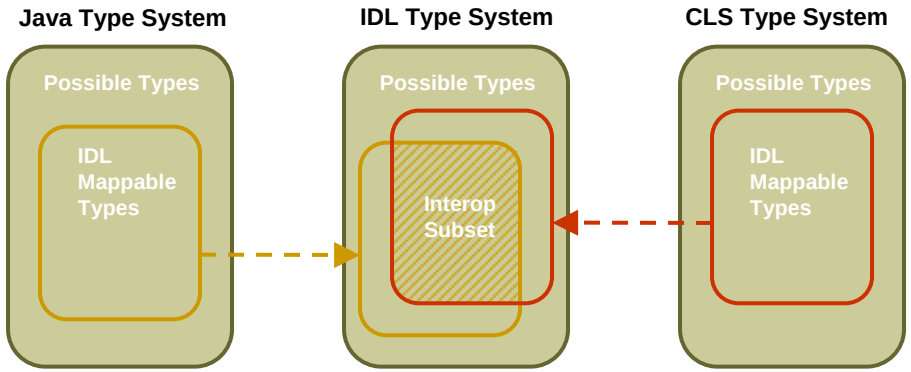
The IIOP.NET package consists of...

- IIOP.NET Channel
  - marshalling of method invocations
  - serialization of data
  - integrate IIOP in .NET remoting
- IDLToCLSCompiler
  - create proxies classes from IDL definitions
- CLSToIDLGenerator
  - create IDL definitions from .NET metadata

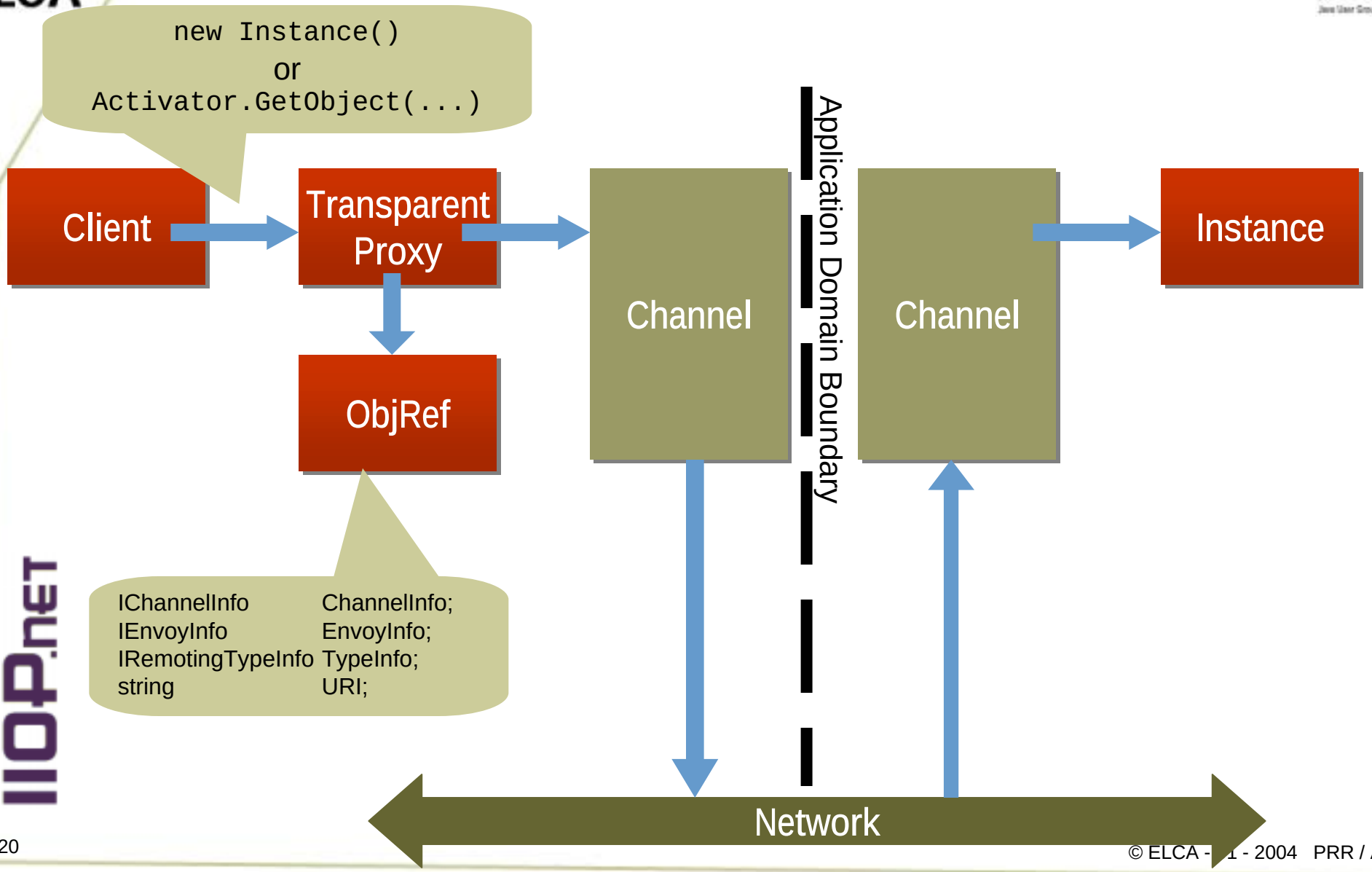
IIOP.NET Technical Overview

- IIOP rather than SOAP
  - transparent reuse of existing servers
  - tight coupling
  - object-level granularity
  - efficiency
- Runtime: standard .NET remoting channel for IIOP
  - transport sink
  - formatter
  - type-mapper

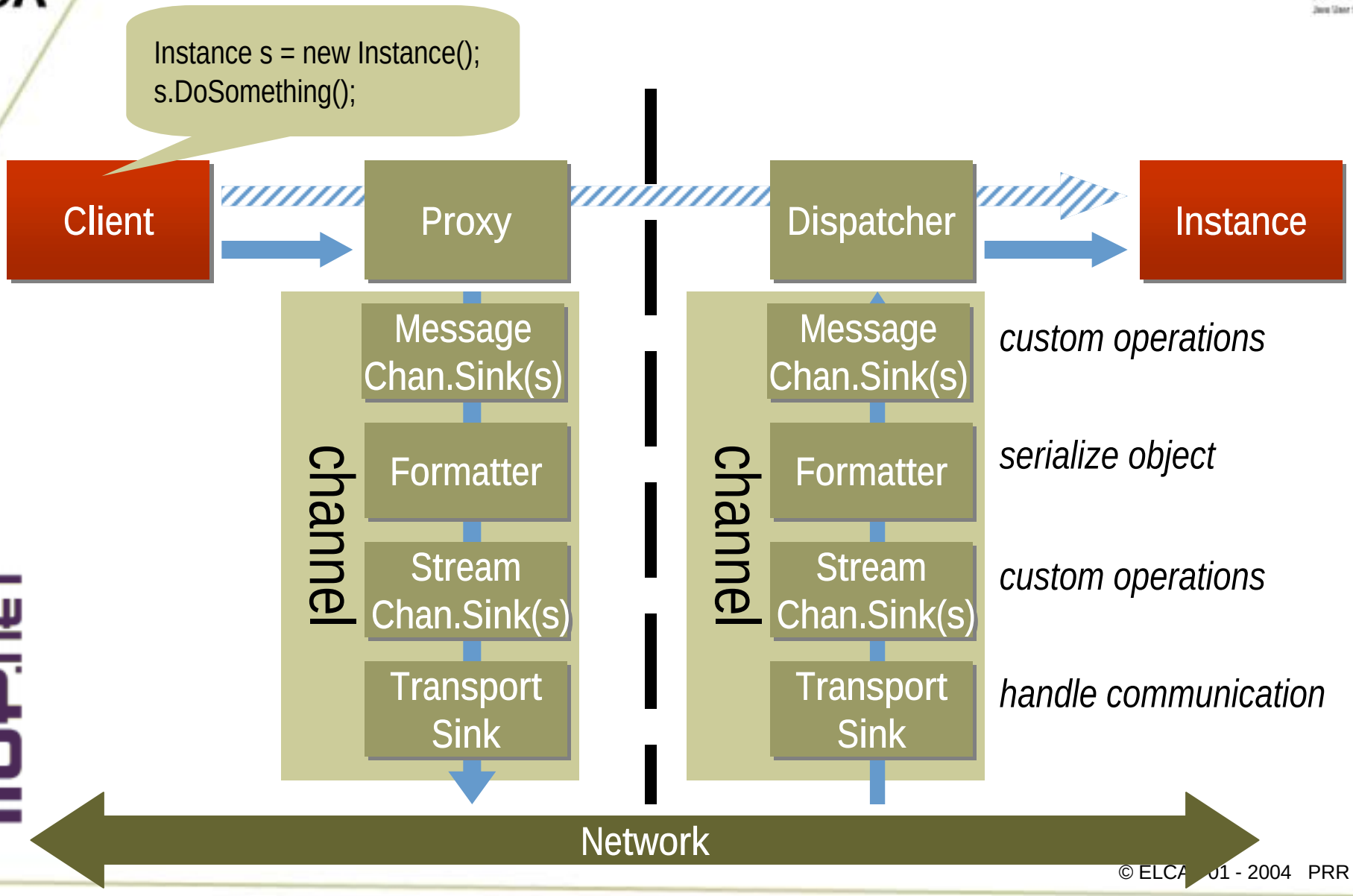




Remoting: Overview



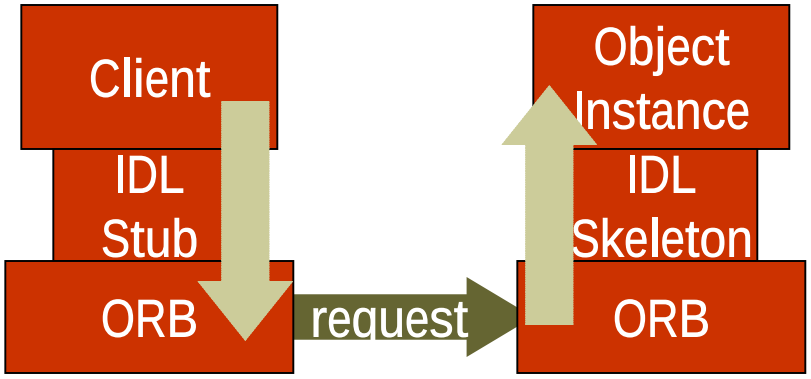
Remoting: Channels



CORBA

CORBA is a standard from OMG

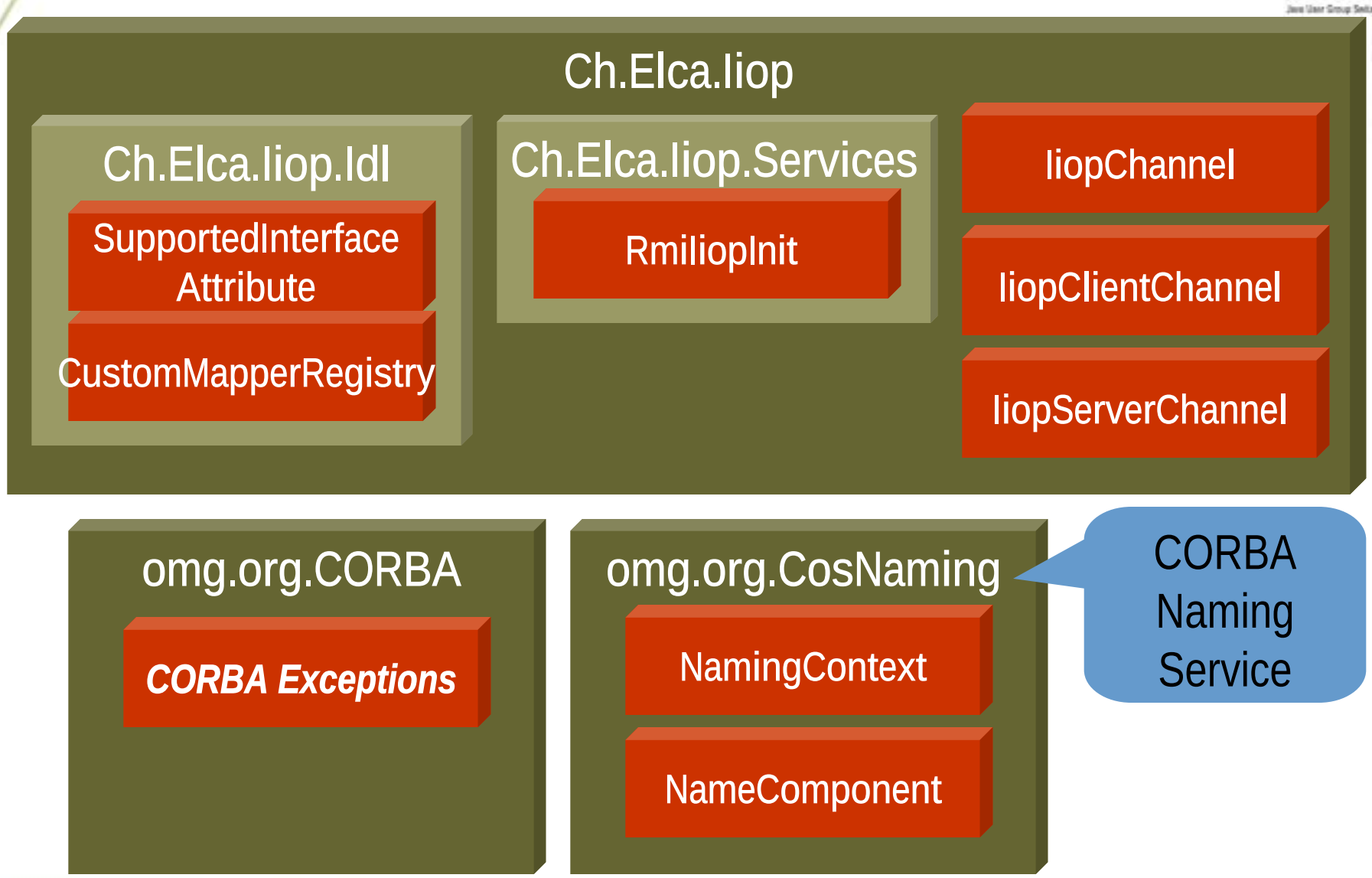
- **O**bject **M**anagement **G**roup
- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture



CORBA defines...

- an object-oriented **type system**
- an interface **definition language** (IDL)
- an object request **broker** (ORB)
- an inter-orb **protocol** (IIOP) to *serialize* data and *marshall* method invocations
- object **references** (IOR)
- language **mappings** from Java, C++, Ada, COBOL, Smalltalk, Lisp, Phyton
- ... and many additional standards and interfaces for distributed **security**, **transactions**, ...

IIOP.NET Relevant Classes



Using IIOP.NET Channel

Server-side

```
// register channel
IiopChannel chan = new IiopChannel(ServerPort);
ChannelServices.RegisterChannel(chan);
```

Client-side

```
// register client channel
IiopClientChannel chan = new IiopClientChannel();
ChannelServices.RegisterChannel(chan);

// register channel (allows callbacks)
IiopChannel chan = new IiopChannel();
ChannelServices.RegisterChannel(chan);
```

Corba Naming

Provide access to Corba Naming Service

```
// Fetch remote naming service object
NamingContext ns = RemotingServices.Connect(
     typeof(NamingContext),
     "corbaloc::localhost:3528/JBoss/Naming/root")
     as NamingContext;

// Compose remote object's name
NameComponent[] name = new NameComponent[] {
     new NameComponent("Demo"),
     new NameComponent("complexoperations") };

// get reference to the remote object
proxy = (ComplexOperationsHome) ns.resolve(name);
```

URI is ORB dependent

application dependent

SupportedInterfaceAttribute

Situation

- Use .NET as server (or a client with callback object)

Problem

- Server: serialized object reference contains **exactly one** type (the IDL interface supported by the remote object)

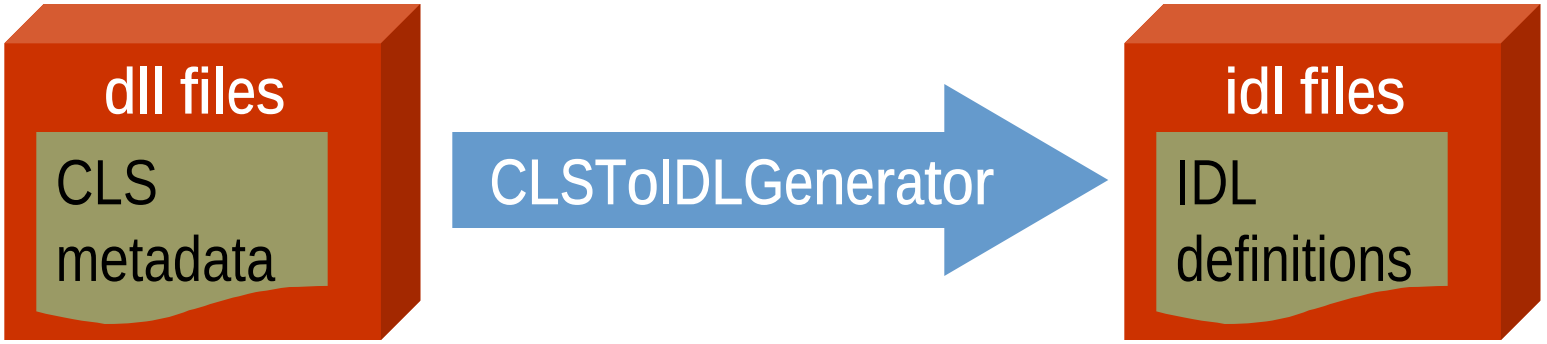
Solution

- `SupportedInterface(t)` forces server to send the type info `t` instead of the implementation class type

```
[SupportedInterface(typeof(IChatroom))]
public class ChatroomImpl: MarshalByRefObject, IChatroom, ...{
```

serialization: use type info „IChatroom“ instead of „ChatroomImpl“

IIOP.NET: IDL Generator




```
CLSToIDLGenerator [options] type assembly
```

type
fully qualified type name (e.g Demo.Complex)

assembly
assembly containing type to map

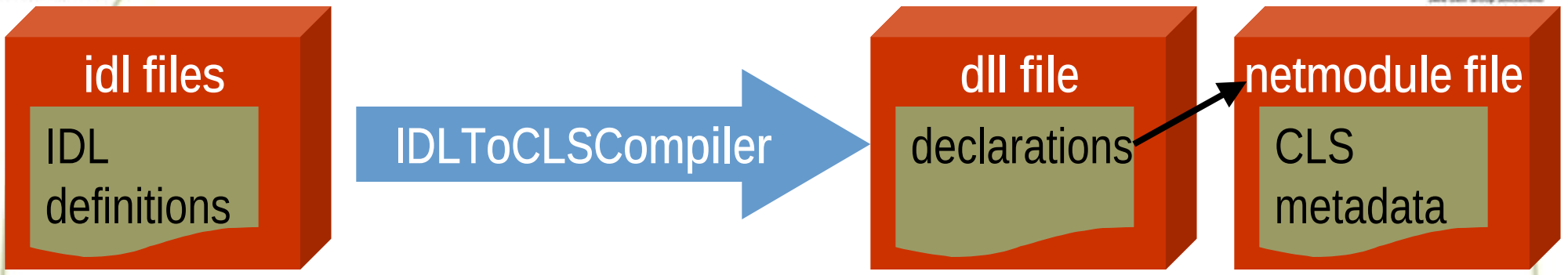
options

`-o dir`	output directory
`-c xmlfile`	use custom mapping

notes

- may import other assemblies recursively
- generates an IDL per type

IIOP.NET: IDL Compiler



`IDLToCLSCompiler [options] assembly idl_file`$_1$ **`[idl_file`$_2$ **`…]`**

`assembly`	target assembly
`idl_file`	IDL definition source
`options`	
-o dir	output directory
-c xmlfile	use custom mapping
-d symbol	define preprocessor symbol
-idir dir	alternate IDL directories
-r assembly	check assemblies for already mapped types

notes

- may import other idl files recursively (locally or from the –idir directories)
- generates a DLL and a netmodule file (restriction in Reflection.Emit)
- generates abstract class for valuetypes; own implementation required!

IIOP.NET: IDL Compiler (valuetypes)

IDL contains only type definitions

- compiler maps **valuetypes to abstract classes**
- if type is used, a **local implementation** must be provided for
  - declared methods
  - declared properties
  - default constructor (for deserialization)
- implementation can contain additional code
- implementation class for ***T*** is named ***TImpl***

```
[Serializable]
[ImplClass(„ComplexImpl“)]
public abstract Complex {
  public float re;
  public float im;

  public abstract float alpha{
      get; set;
  }
  public abstract float radius{
      get; set;
  }
}
```

```
[Serializable]
public ComplexImpl: Complex {
  public ComplexImpl() {...}

  public override float alpha {
      get {..implementation..}
      set {..implementation..}
  }
  public override float radius{
      get {..implementation..}
      set {..implementation..}
  }
}
```

IIOP.NET Limitations

IIOP Limitations

- no client-activated objects (use factory!)
- no call-context support
- no .NET transport header support

Implementation Limitations

- no SSL or HIOP
- sponsor support

Others

- Intellisense bug
  - Intellisense for C# does not handle netmodules (Intellisense for VB does)

Agenda

Introduction
- What / Why IIOP.NET?

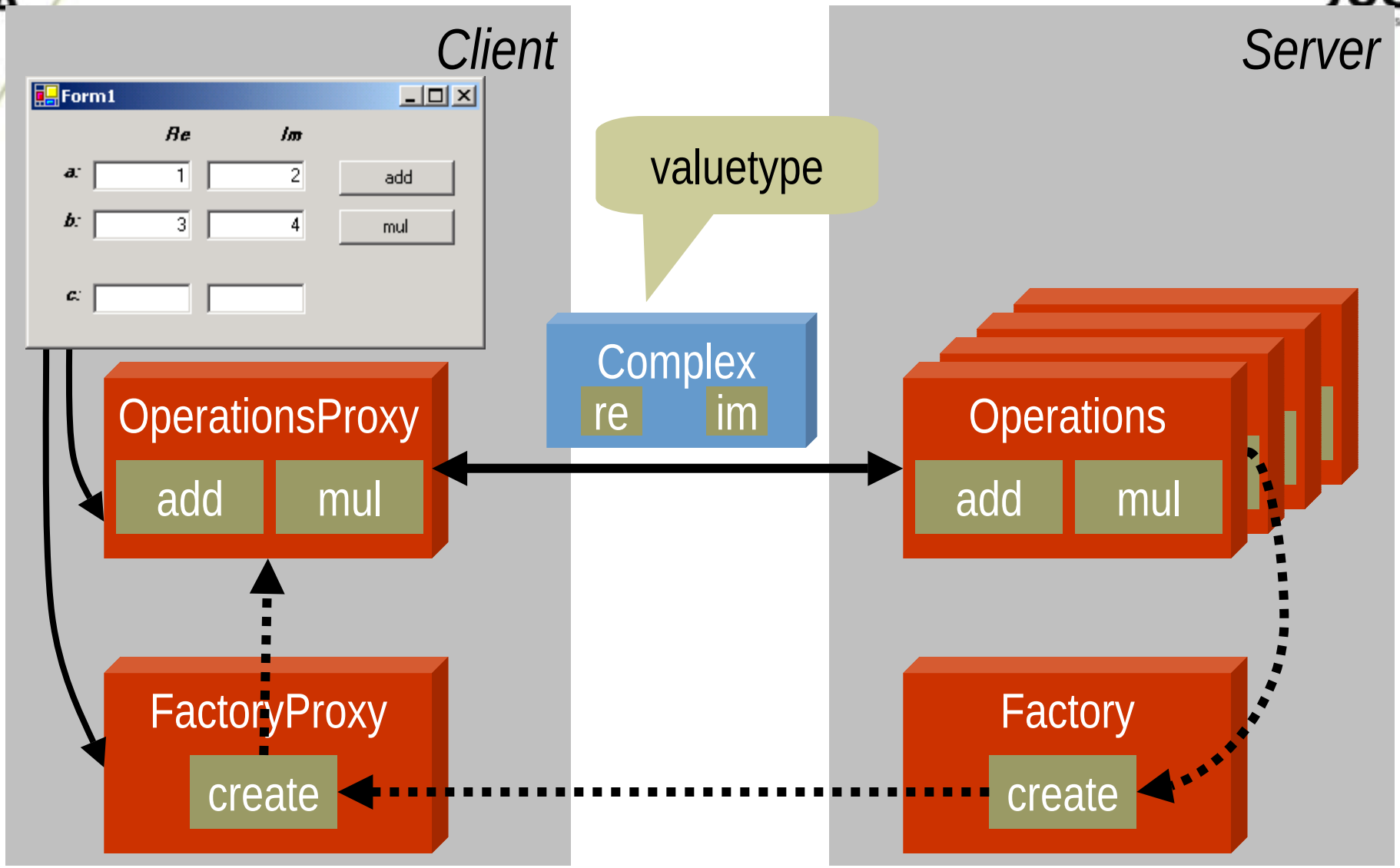
Interoperability
- Web Services vs. Remoting
- Type System Issues

IIOP.NET Tools
- Channel
- IDL Generator
- IDL Compiler

IIOP.NET Examples
- .NET to .NET
- .NET to J2EE
- .NET to CORBA

Conclusions
- Performance
- IIOP.NET State
- IIOP.NET Future

Demo: Overview



Demo: .NET to .NET (shared DLL)

Steps

1. Create Shared Definitions
2. Compile Definitions
3. Create Server
4. Compile Server
5. Create Client
6. Compile Client
7. Run

Notes

- no IDL involved;
- metadata shared in common dll
- IIOP has no support for client-activated objects, use factory instead
- access factory with Activator and URI

Demo: .NET to .NET (shared IDL)

Steps

1. Create IDL
2. Compile IDL (proxies)
3. Implement Valuetypes
4. Implement Client
5. Compile Client
6. Run

Notes

- the client is created from IDL, no shared dll
- must provide local implementation of valuetype
- access factory with Activator and URI

Demo: .NET to J2EE (EJB/JBoss)

Steps

1. Implement Server (EJB)
2. Build Server
3. Generate IDL
4. Deploy Server

5. Compile IDL
6. Implement Valuetypes
7. Implement Client
8. Build Client
9. Run

Notes

- server is JBoss 3.2.3
- use EJB home as factory class
- must provide local implementation of valuetype
- some server configuration required (ejb-jar.xml, jboss.xml)
- get EJBHome reference using CORBA naming service
- JBoss naming service is on port 3528

Demo:.NET to CORBA (omniORB)

Steps

1. Create definition in IDL
2. Compile IDL for omniORB
3. Implement Servant
4. Build Servant

5. Compile IDL for .NET
6. Implement Valuetype
7. Implement Client
8. Build Client
9. Run

Notes

- uses omniORB 4.0.3
- Complex is a struct because omniORB doesn‘t support valuetypes (no local implementation needed)
- Configuration in program code
- Non-trivial servant registration due to C++ model
- get Factory reference using CORBA naming service
- naming service on port 11356

Agenda

Introduction
- What / Why IIOP.NET?

Interoperability
- Web Services vs. Remoting
- Type System Issues

IIOP.NET Tools
- Channel
- IDL Generator
- IDL Compiler

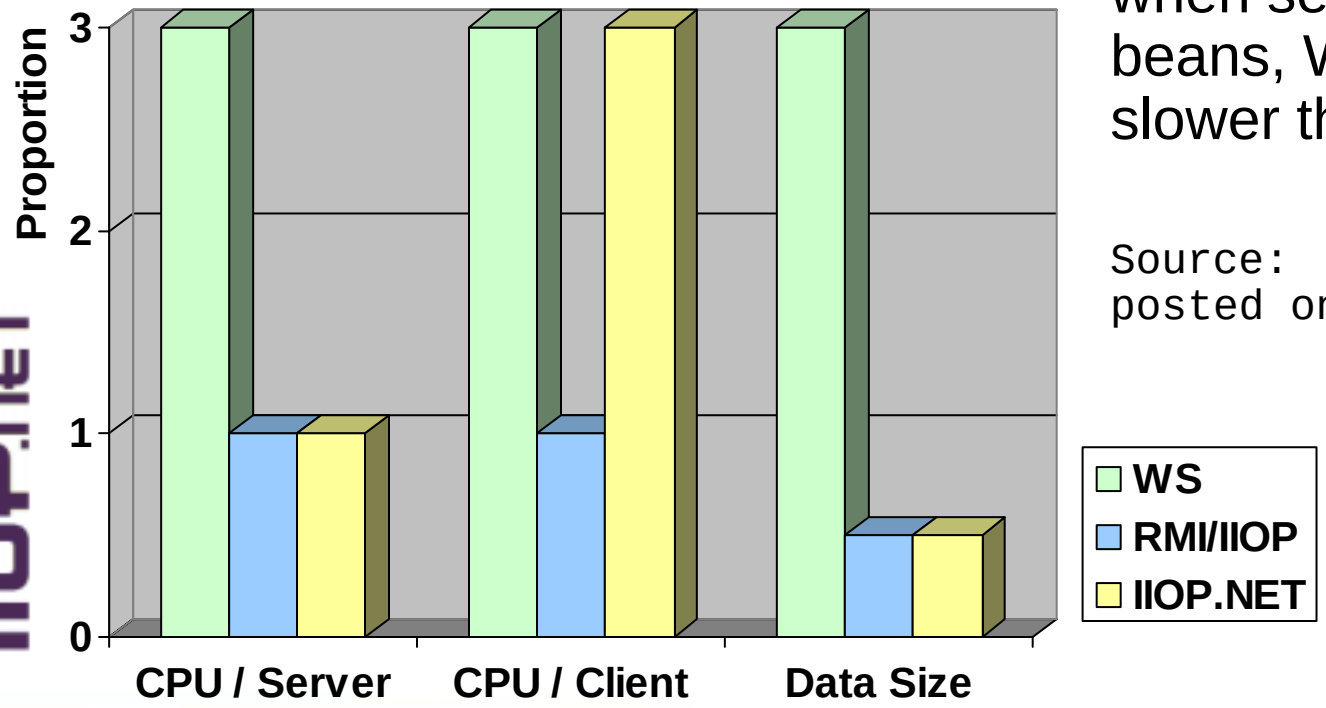
IIOP.NET Examples
- .NET to .NET
- .NET to J2EE
- .NET to CORBA

Conclusions
- Performance
- IIOP.NET State
- IIOP.NET Future

IIOP.NET Performance

Test Case:

- WebSphere 5.0.1 as server
- Clients
  - IBM SOAP-RPC Web Services
  - IBM Java RMI/IIOP
  - IIOP.NET



Response time

- receiving 100 beans from server
  - WS: 4.0 seconds
  - IIOP.NET: 0.5 seconds
- when sending many more beans, WS are then 200% slower than IIOP.NET

Source:
posted on IIOP.NET forum

IIOP.NET Performance

Comments

- IIOP.NET is not optimized (yet)
  - uses a lot of (slow) reflection
  - cache reflected data
  - dynamically generated proxies
- Some .NET problems
  - first call is slow (must compile code, initialize system)
- IIOP requires less bandwidth than SOAP
  - less bandwidth needed for same information
  - parsing time often proportional to data size

IIOP.NET Project State

Platform
- .NET Framework v1.1

Protocol
- CORBA 2.3.1

Peers
- BEA WebLogic 6.1 / 8.1
- IBM WebSphere 4.0 / 5.0
- JBoss 3.2.1 / 3.2.3
- MICO
- TAO
- OmniORB 4.0
- SUN JDK 1.4

IIOP.NET Community
- worldwide
- IIOP.NET
  - > 1500 downloads
  - > 50000 hits
- CodeProject Articles
  - 1: > 25000 hits
  - 2: > 8000 hits
- Awards
  - best c# article
  - SDNUG tool of month

IIOP.NET Future

Code Consolidation
- Support Mono
- Support more ORBs
- Optimizations
- Installation
  - Binary releases
  - Configuration

Features
- More CORBA services standards

Free Support
- bug fixing
- documentation

Commercial / ELCA added value services (on-demand)
- Integrated Solutions
- LEAF.NET Framework
- Specific Development
- Optimized Versions
- Advanced Support
- Consulting, Expertise
- Visual Studio.NET integration

ELCA

Thank you for your attention

ELCA

Questions?