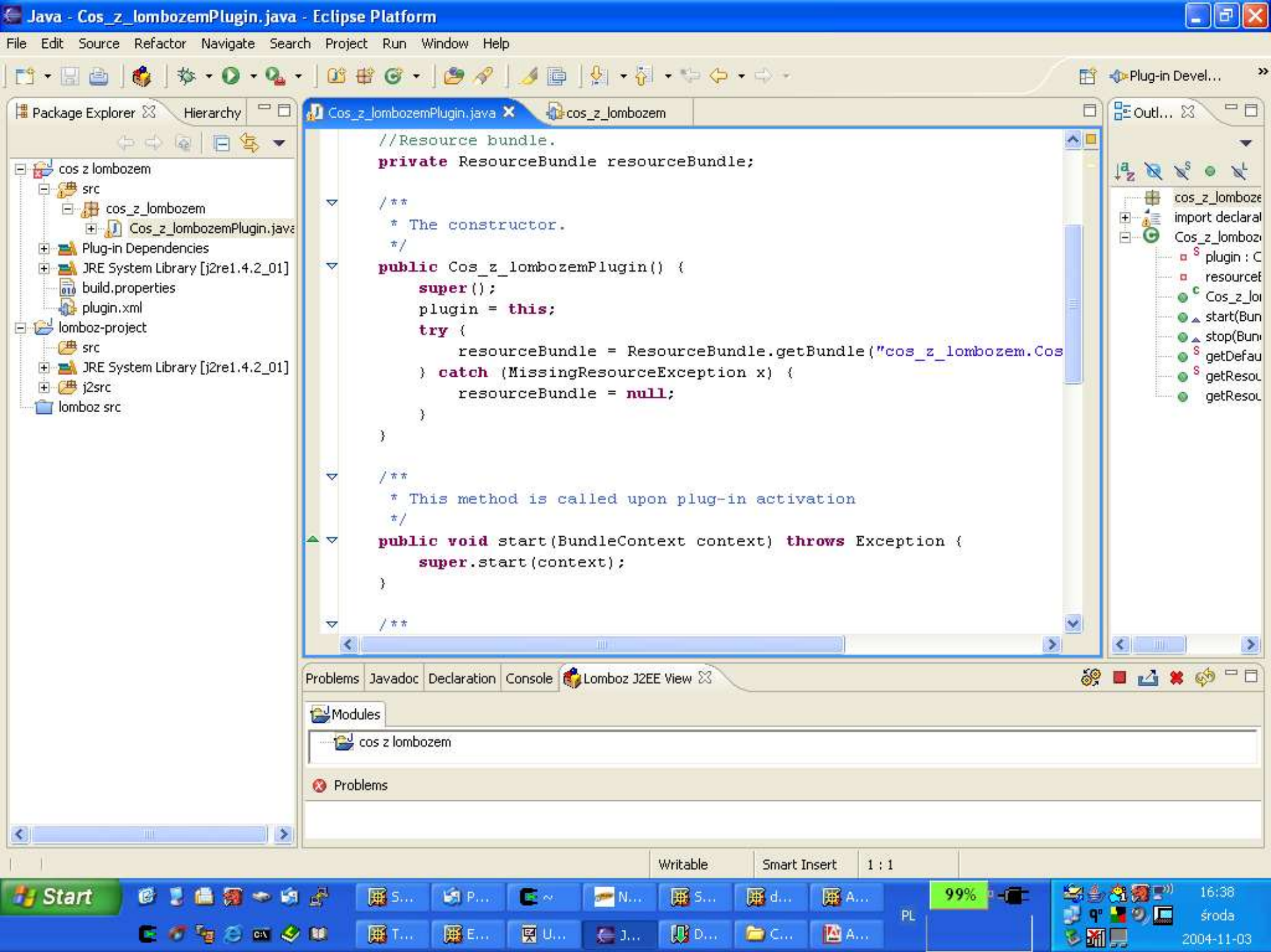
Protokoly w technologii obiektow rozproszonych - CORBA, RMI/IIOP, COM, SOAP

Paweł Kozioł
p.koziol@students.mimuw.edu.pl

Na początek - moja praca magisterska

- “Narzędzie dla środowiska Eclipse wspierające stosowanie wzorców projektowych J2EE”
- Prowdzący: dr Janusz Jabłonowski

- Narzędzie do tworzenia IDE
- Mechanizm wtyczek (plug-ins)
- Otwarty kod źródłowy (GPL), otwarta architektura
- Wsparcie dla tworzenia nowych wtyczek – Plug-in Development Environment (PDE)
- Java Development Tooling (JDT)


Java - Cos_z_lombozemPlugin.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Plug-in Devel...
Package Explorer
Hierarchy
cos z lombozem
src
cos_z_lombozem
Cos_z_lombozemPlugin.java
Plug-in Dependencies
JRE System Library [j2re1.4.2_01]
build.properties
plugin.xml
lomboz-project
src
JRE System Library [j2re1.4.2_01]
j2src
lomboz src
Cos_z_lombozemPlugin.java
cos_z_lombozem
//Resource bundle.
private ResourceBundle resourceBundle;
/**
* The constructor.
*/
public Cos_z_lombozemPlugin() {
super();
plugin = this;
try {
resourceBundle = ResourceBundle.getBundle("cos_z_lombozem.Cos
} catch (MissingResourceException x) {
resourceBundle = null;
}
}
/**
* This method is called upon plug-in activation
*/
public void start(BundleContext context) throws Exception {
super.start(context);
}
/**
Outl...
cos_z_lomboze
import declaral
Cos_z_lomboz
plugin : C
resourceE
Cos_z_lo
start(Bun
stop(Bun
getDefau
getResou
getResou
Problems
Javadoc
Declaration
Console
Lomboz J2EE View
Modules
cos z lombozem
Problems
Writable
Smart Insert
1 : 1
Start
99%
PL
16:38
środa
2004-11-03

Wzorce projektowe

- Wzór dobrego i sprawdzonego rozwiązania pewnego problemu w pewnym kontekście
- Opisują często stosowane techniki projektowania aplikacji
- Oprócz projektowych istnieją wzorce dla analizy, kodownia i testowania
- Również wiele specyficznych wzorców np. dla systemów współbieżnych, sztucznej inteligencji, projektowania interfejsu użytkownika

Przykład wzorca:

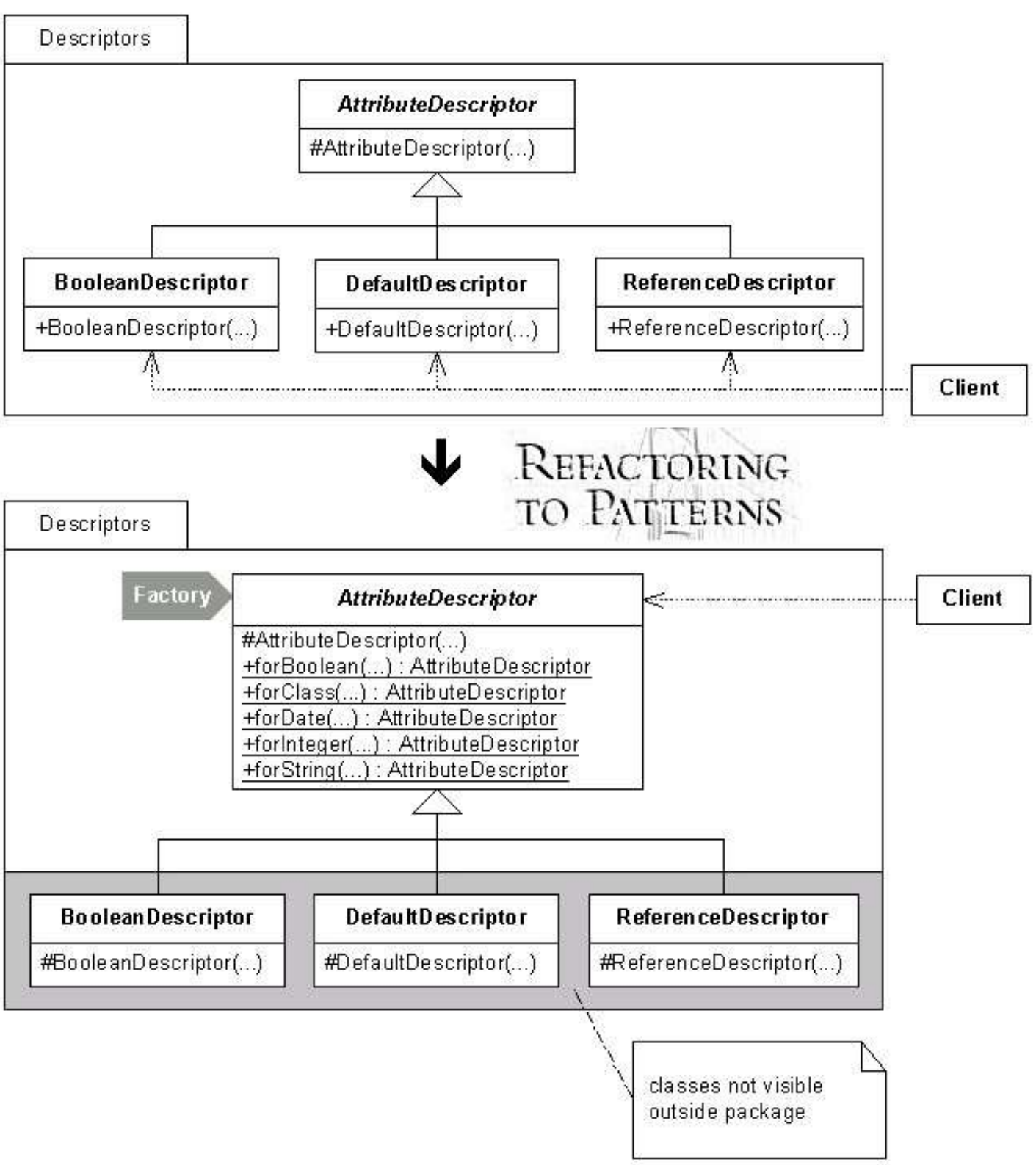
Compose Method

```
public void add(Object element) {
  if (!readOnly) {
    int newSize = size + 1;
    if (newSize > elements.length) {
      Object[] newElements =
        new Object[elements.length + 10];
      for (int i = 0; i < size; i++)
        newElements[i] = elements[i];
      elements = newElements;
    }
    elements[size++] = element;
  }
}
```

```
public void add(Object element) {
  if (readOnly)
    return;
  if (atCapacity())
    grow();
  addElement(element);
}
```

Przykład 2:

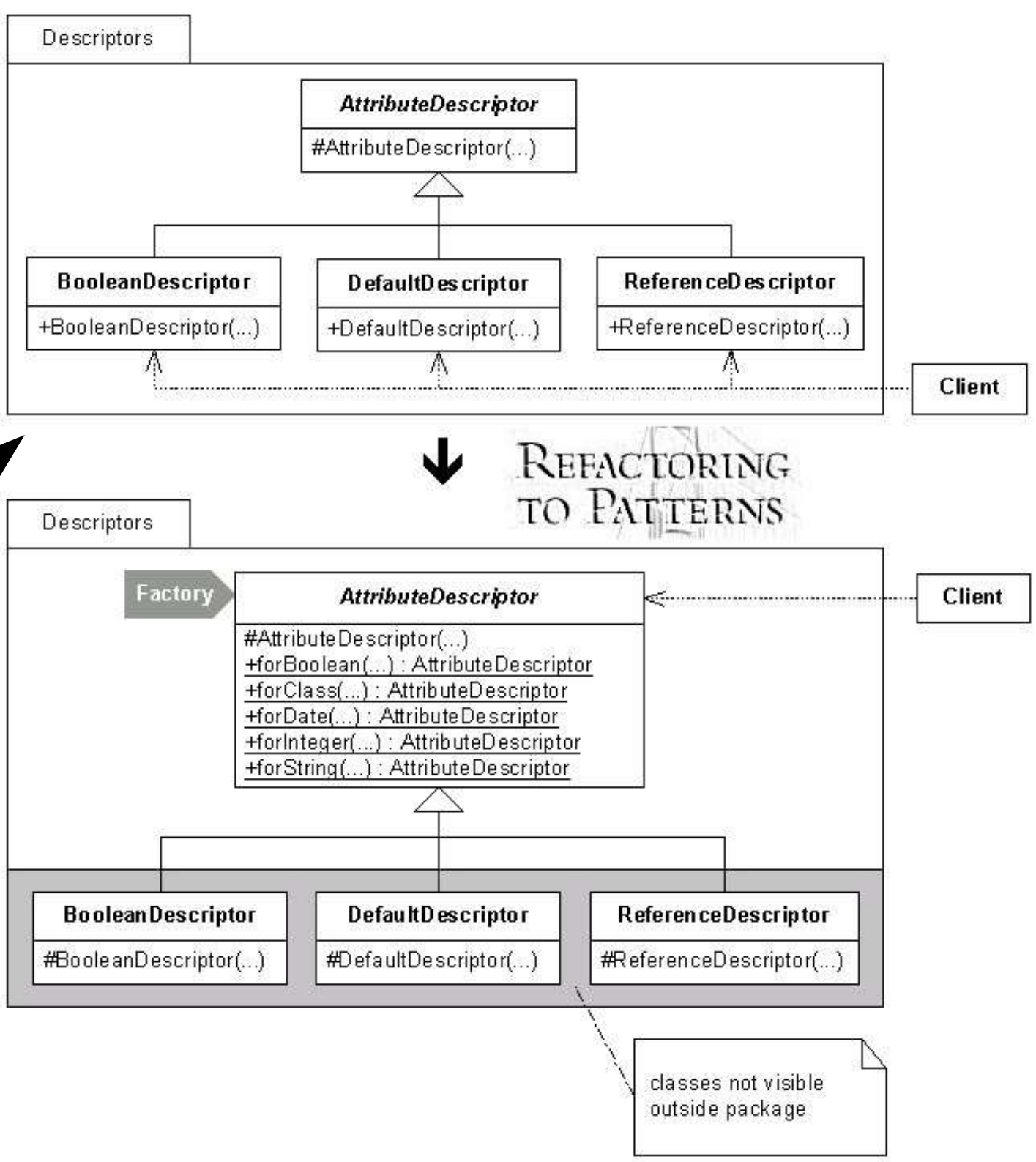
Encapsulate Classes with Factory



Przykład 2:

Encapsulate Classes with Factory

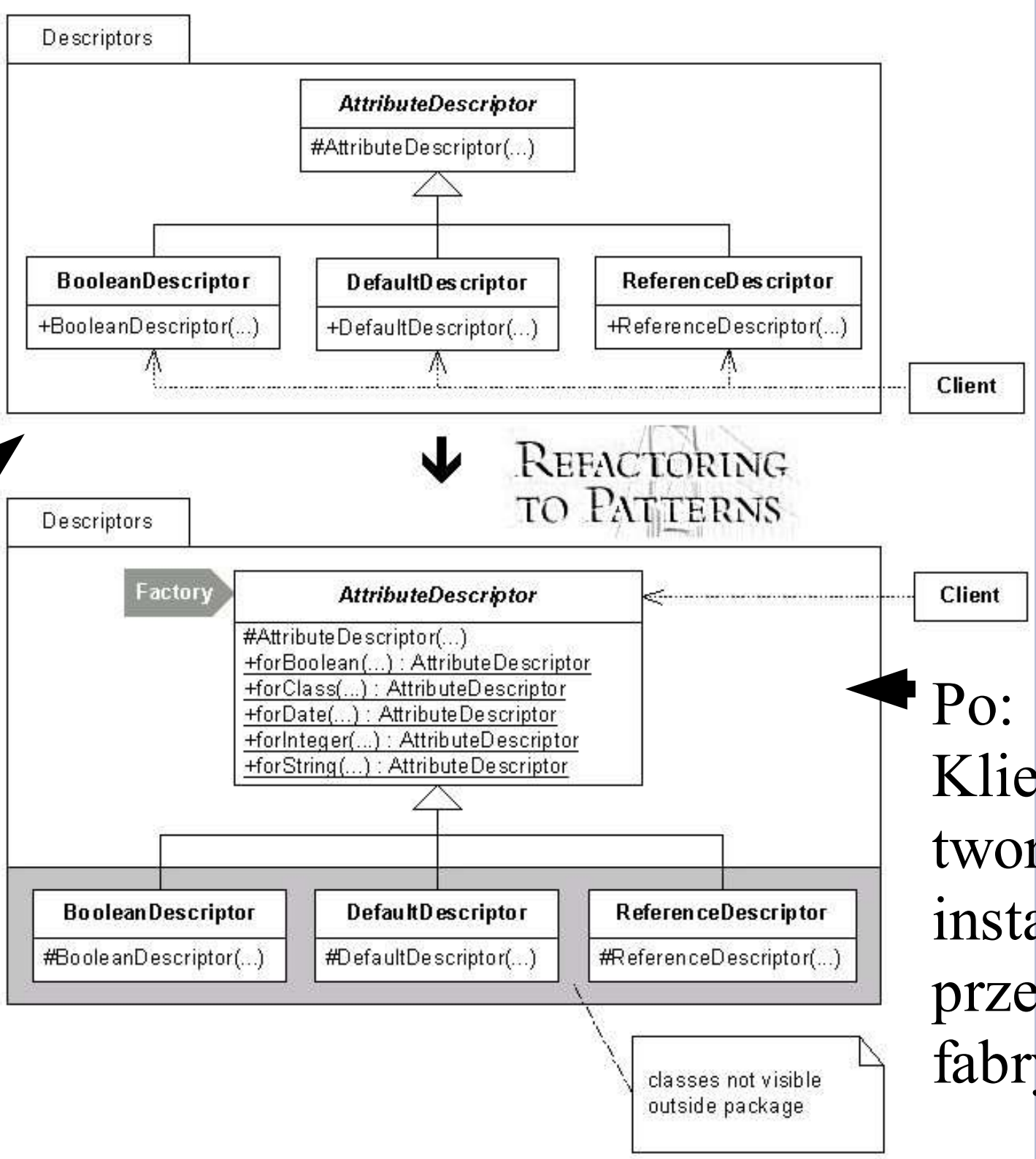
Przed: klient ma dostęp do wielu klas z pakietu, Wszystkie mają wspólny interfejs



Przykład 2:

Encapsulate Classes with Factory

Przed: klient ma dostęp do wielu klas z pakietu, Wszystkie mają wspólny interfejs



Po: Klient tworzy instancje przez fabrykę

Wzorce znane z Javy

- Przykładowe wzorce:
  Singleton, Iterator, Observer, Proxy, Facade, Adapter, Factory

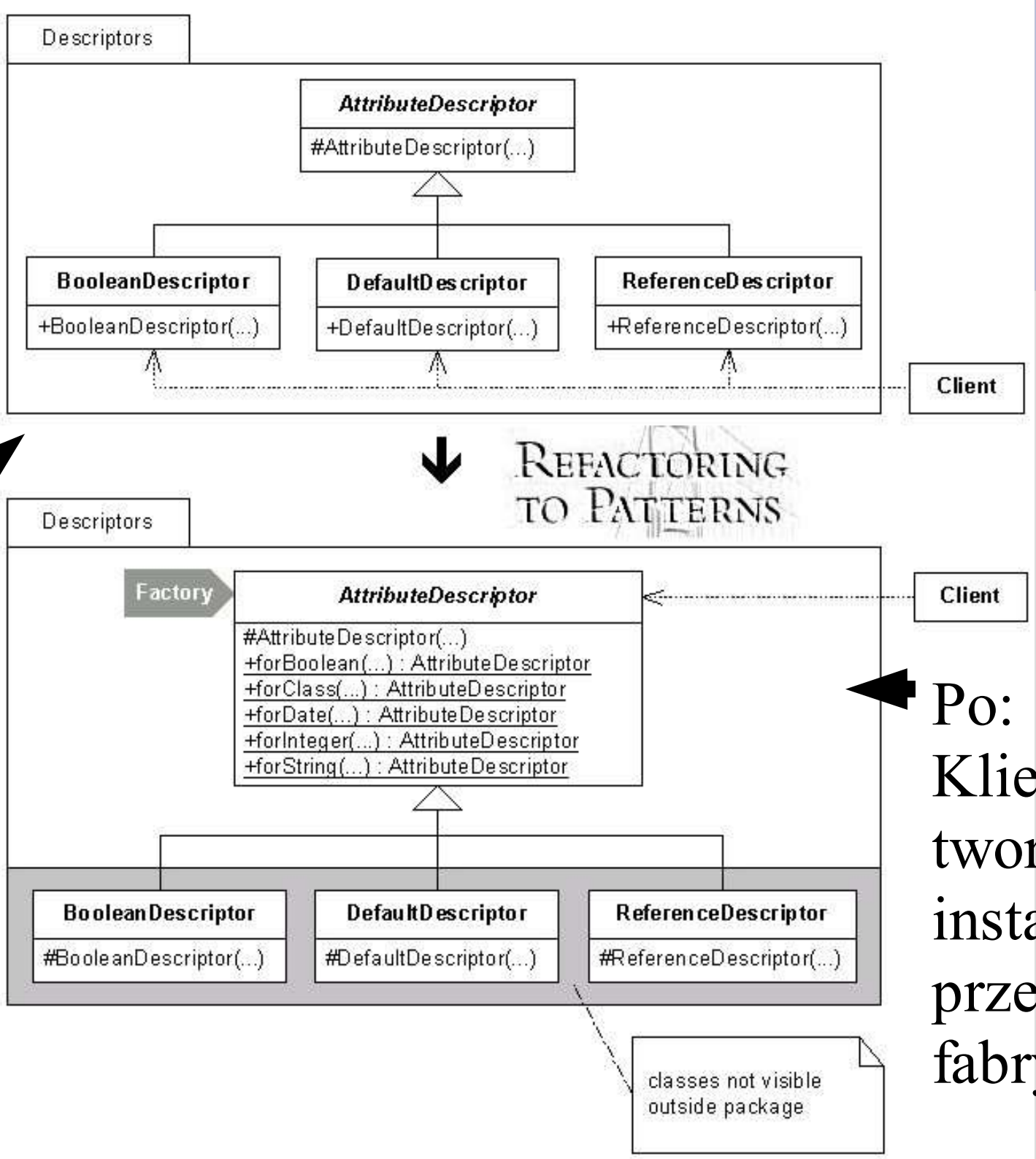
Wzorce a refaktoryzacja

- Refaktoryzacja – zmiana stuktury kodu bez zmiany zachowania
- Przykład:
  - zmiana nazwy zmiennej,
  - wyodrębnienie kodu do metody
- Niektóre proste refaktoryzacje są już zaimplementowane w Eclipse
- Popularne podejście to refaktoryzacja do wzorców projektowych

Przykład 2:

Encapsulate Classes with Factory

Przed: klient ma dostęp do wielu klas z pakietu, Wszystkie mają wspólny interfejs



Po: Klient tworzy instancje przez fabrykę

Java 2 Enterprise Edition (J2EE)

- Część technologii Java
- Platforma do tworzenia wielowarstwowych, rozproszonych aplikacji dla przedsiębiorstw
- Standard przemysłowy
- Zawiera wiele technologii
  - Enterprise Java Beans
  - Java DataBase Connectivity
  - Java Message Service
  - Java Servlet Pages
  - i wiele innych

Moja praca

- Wsparcie dla programisty J2EE piszącego w Eclipse
- Dwa podejścia
  - automatyczna generacja kodu dla wybranego wzorca projektowego
  - refaktoryzacja istniejącego kodu do wzorca projektowego

Dalsza część referatu

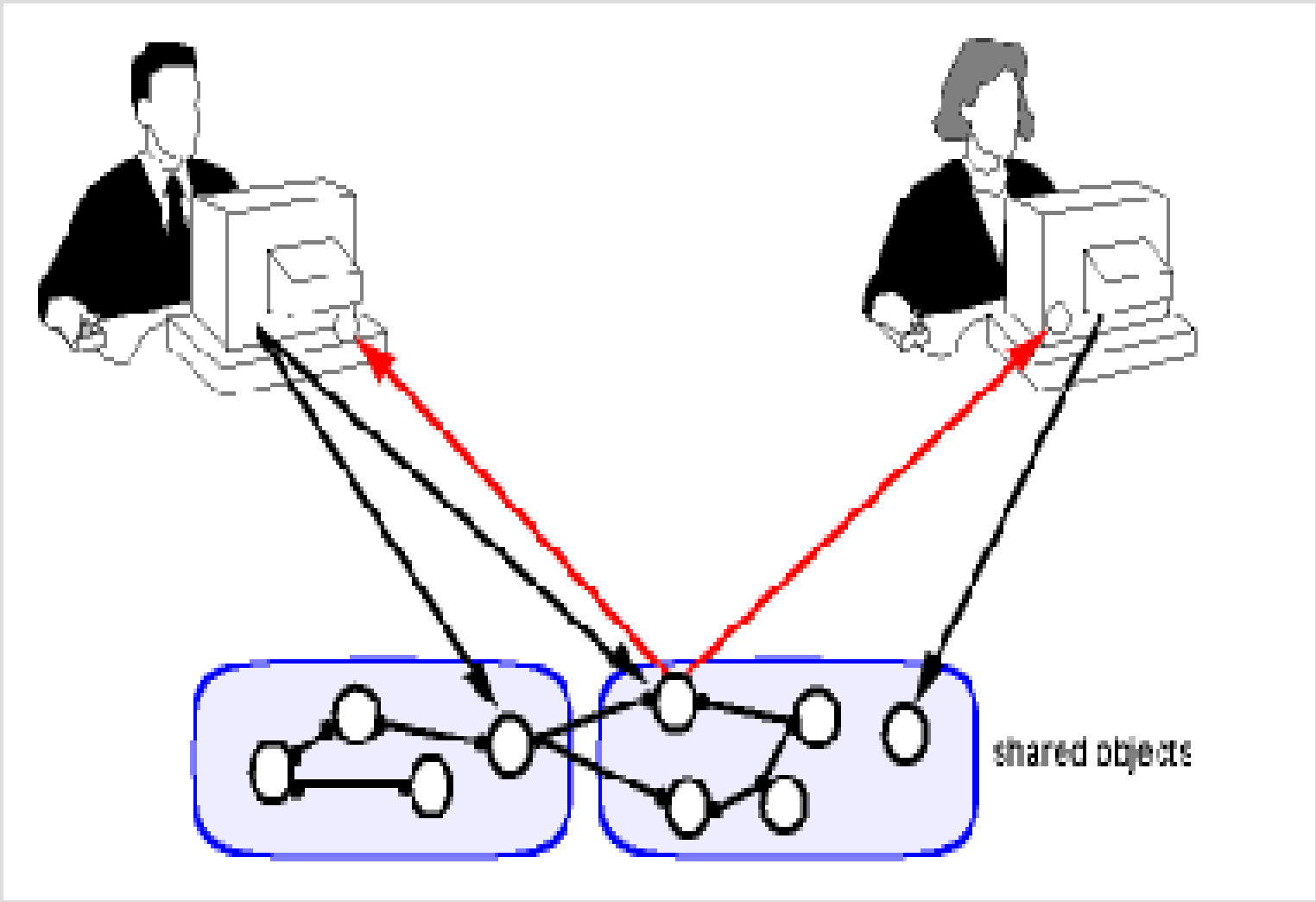
Protokoly w technologii obiektow rozproszonych:
CORBA, RMI/IIOP, DCOM, SOAP

Rozproszone obiekty

"Dawno, dawno temu... był sobie samotny, nieszczęśliwy obiekt, który bardzo się nudził. Chciał połączyć się z innymi sprytniejszymi i bardziej skomplikowanymi obiektami, które pomogłyby mu wyrwać się z izolacji i nudnego życia na przeciążonej maszynie klienckiej”

źródło: www.microsoft.com :)

Rozproszone obiekty

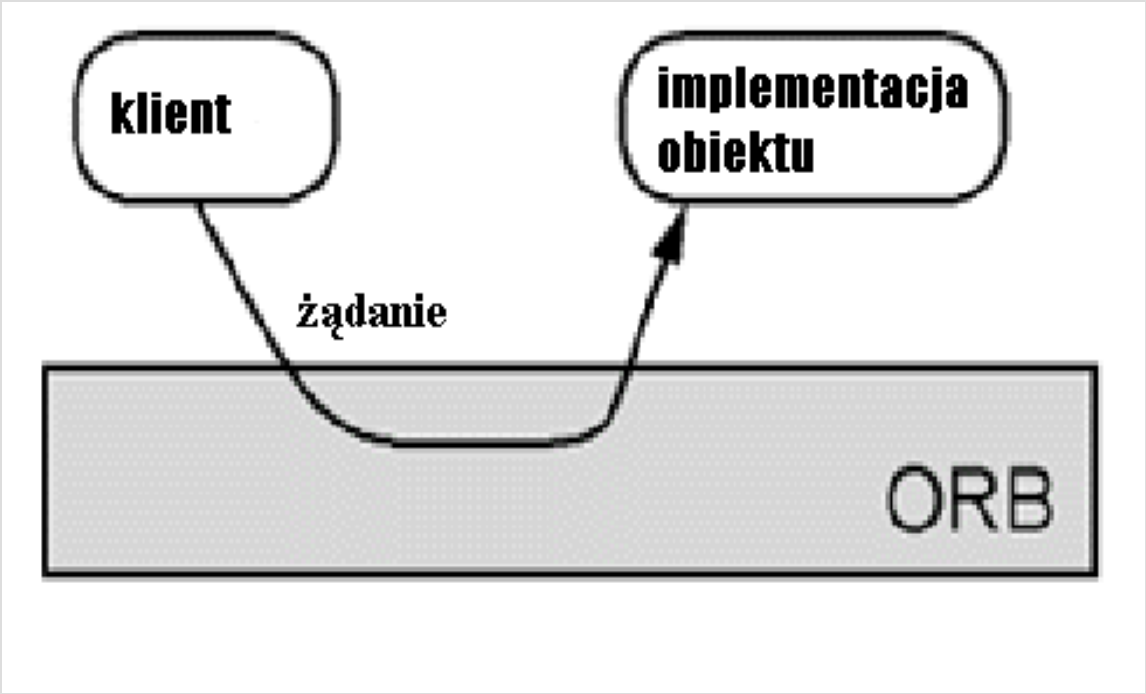


CORBA - Common Object Request Broker Architecture

CORBA - Standard zarządzany przez Object Management Group (OMG)

CORBA - Architektura

- Podstawowy paradygmat:
  - żądanie usługi od zdalnego obiektu
- Usługi zdalnego obiektu definiowane przez interfejsy



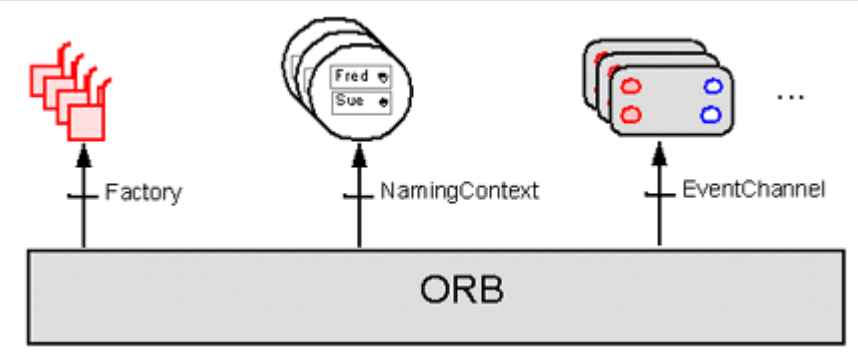
CORBA – Architektura – IDL

- Interfejsy zdefiniowane w IDL – Interface Definition Language
- Interfejs jest niezależny od języka programowania
- Stanowi tylko opis danych
- IDL umożliwia dziedziczenie interfejsów
- Obiekty zdalne są identyfikowane przez referencje, które mają typ interfejsu

CORBA – Architektura – ORB

- ORB – usługa sieciowa, która
  - lokalizuje zdalny obiekt
  - przekazuje mu żądanie
  - czeka na odpowiedź
  - przekazuje odpowiedź klientowi
- ORB zapewnia
  - przeźroczystość położenia zdalnego obiektu
  - niezależność od języka programowania

CORBA - Usługi



- Zarządzanie:
  - cyklem życia obiektu
  - nazwami obiektów
  - zdarzeniami
  - transakcjami
  - wyszukiwaniem obiektów
  - powiązaniami między obiektami

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

CORBA – Architektura – IDL

```
module StockObjects {

  struct Quote {
    string symbol;
    double price;
  };

  exception Unknown{};

  interface Stock {

    // Returns the current stock quote.
    Quote get_quote() raises(Unknown);

    // Sets the current stock quote.
    void set_quote(in Quote stock_quote);

    readonly attribute string description;
  };
};
```

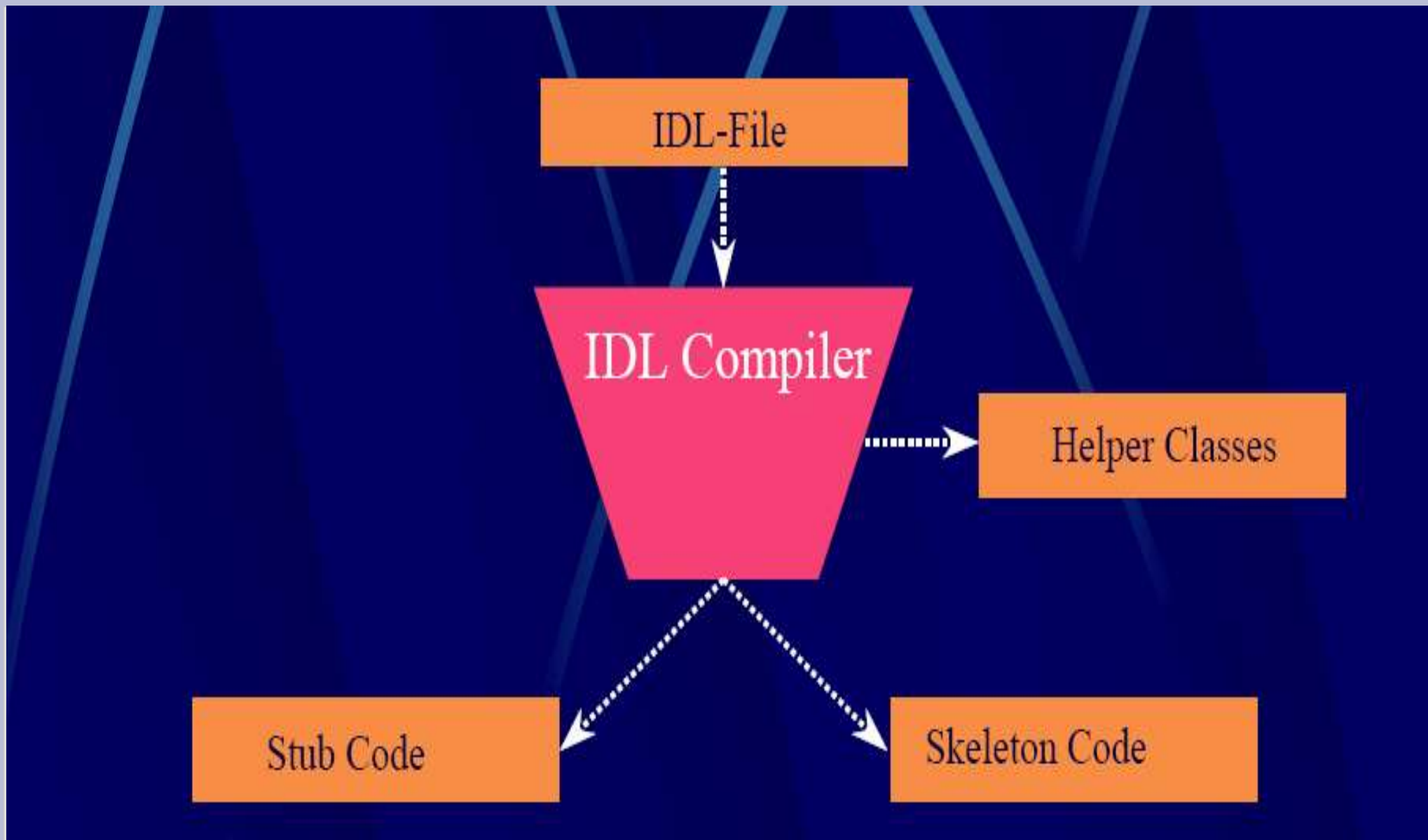
CORBA – Architektura
IDL – mapowanie na język programowania

IDL	**Java**	**C++**
module	package	namespace
interface	interface	abstract class
operation	method	member function
attribute	pair of methods	pair of functions
exception	exception	exception

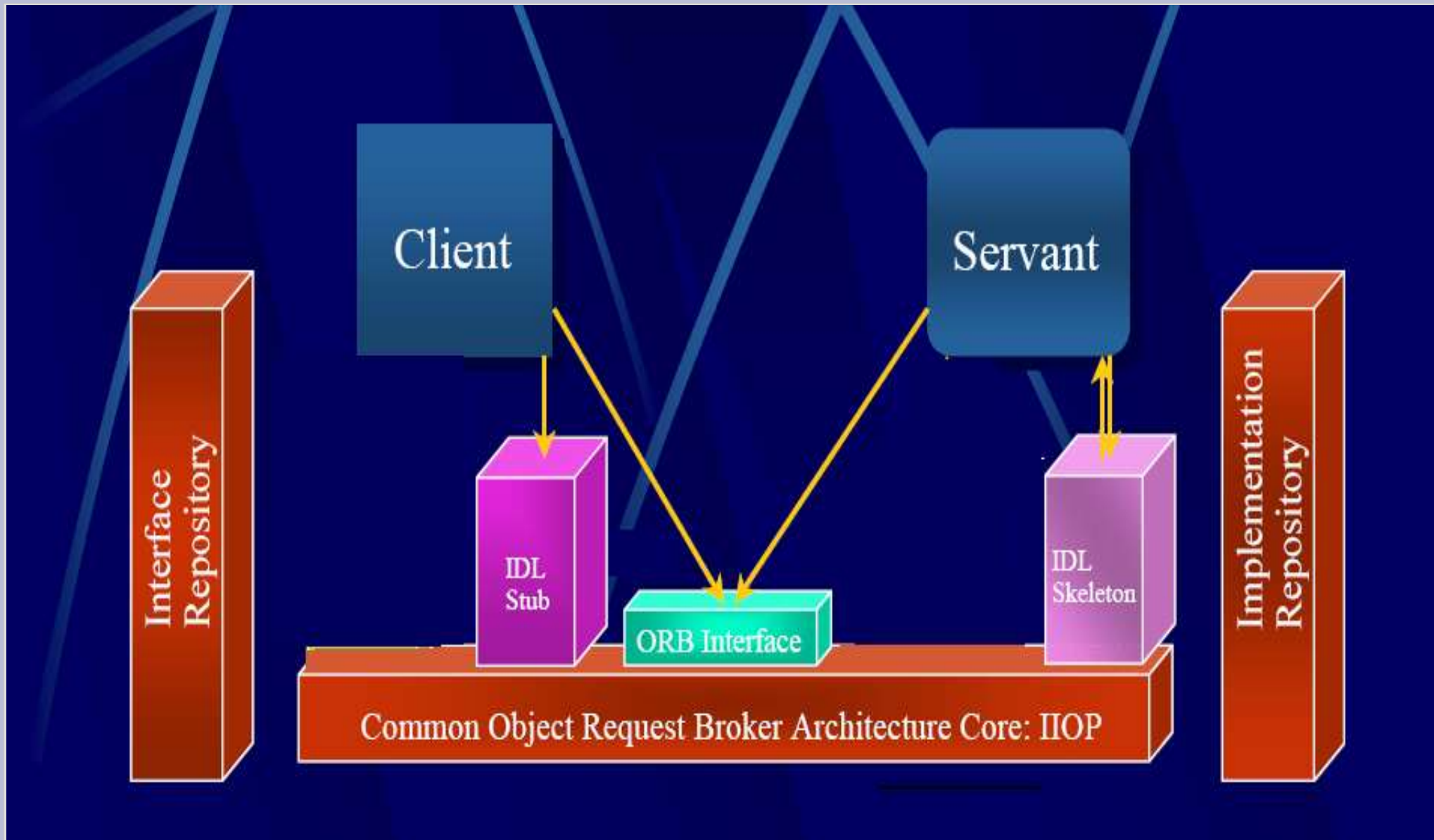
CORBA – Architektura pośrednik i szkielet (stub and skeleton)

- Kompilator IDL generuje dla danego języka programowania
  - stub
    - reprezentuje lokalnie obiekt zdalny
  - skeleton
    - punkt wejścia do systemu rozproszonego
    - konwertuje napływające dane, wywołuje metody i zwraca ponownie skonwertowane wyniki
- Implementacja obiektu zdalnego to zwykle obiekt dziedziczący po szkielecie zawierający implementacje metod z interfejsu IDL

CORBA – Architektura pośrednik i szkielet (stub and skeleton)



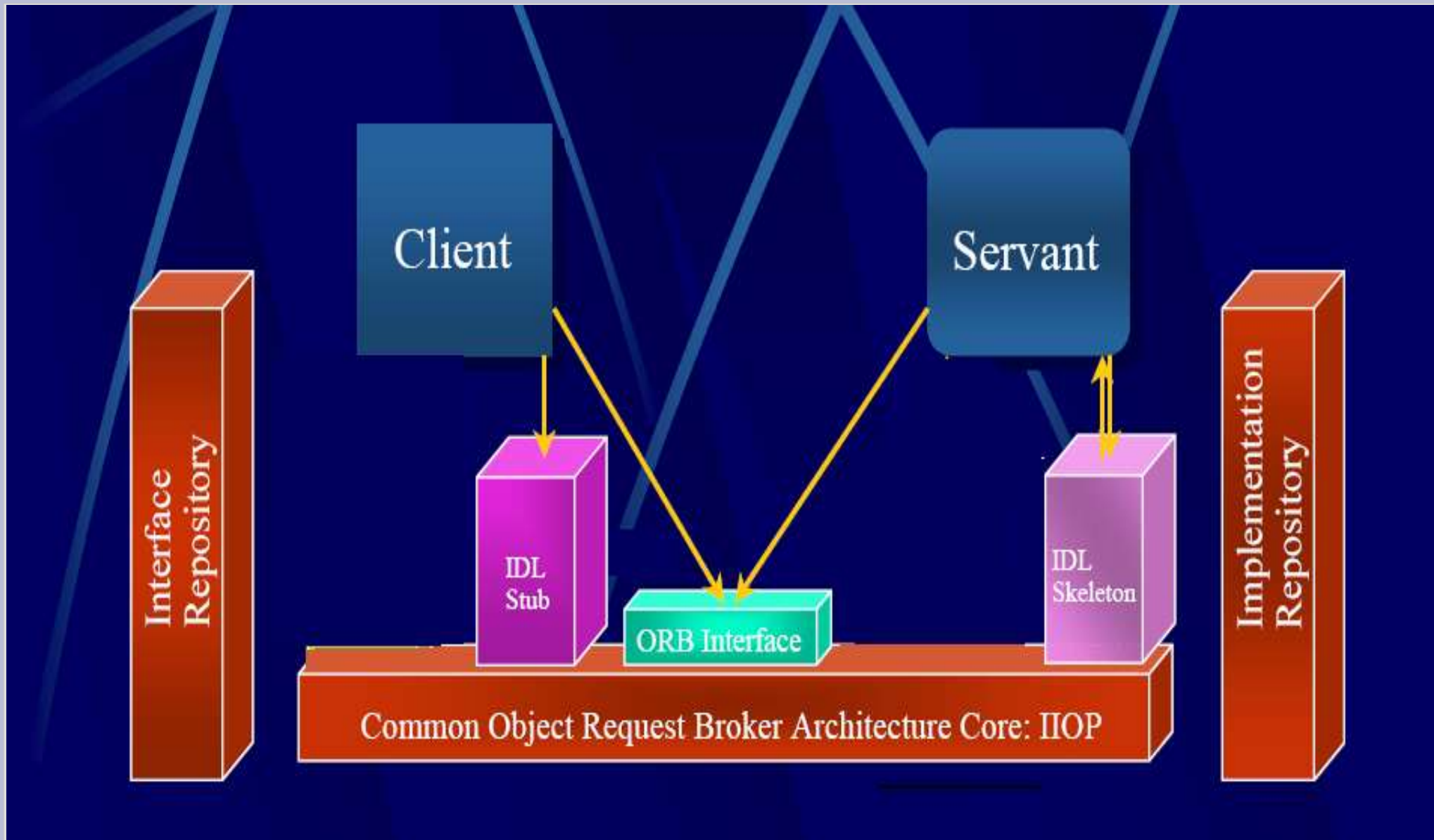
CORBA – Architektura pośrednik i szkielet (stub and skeleton)



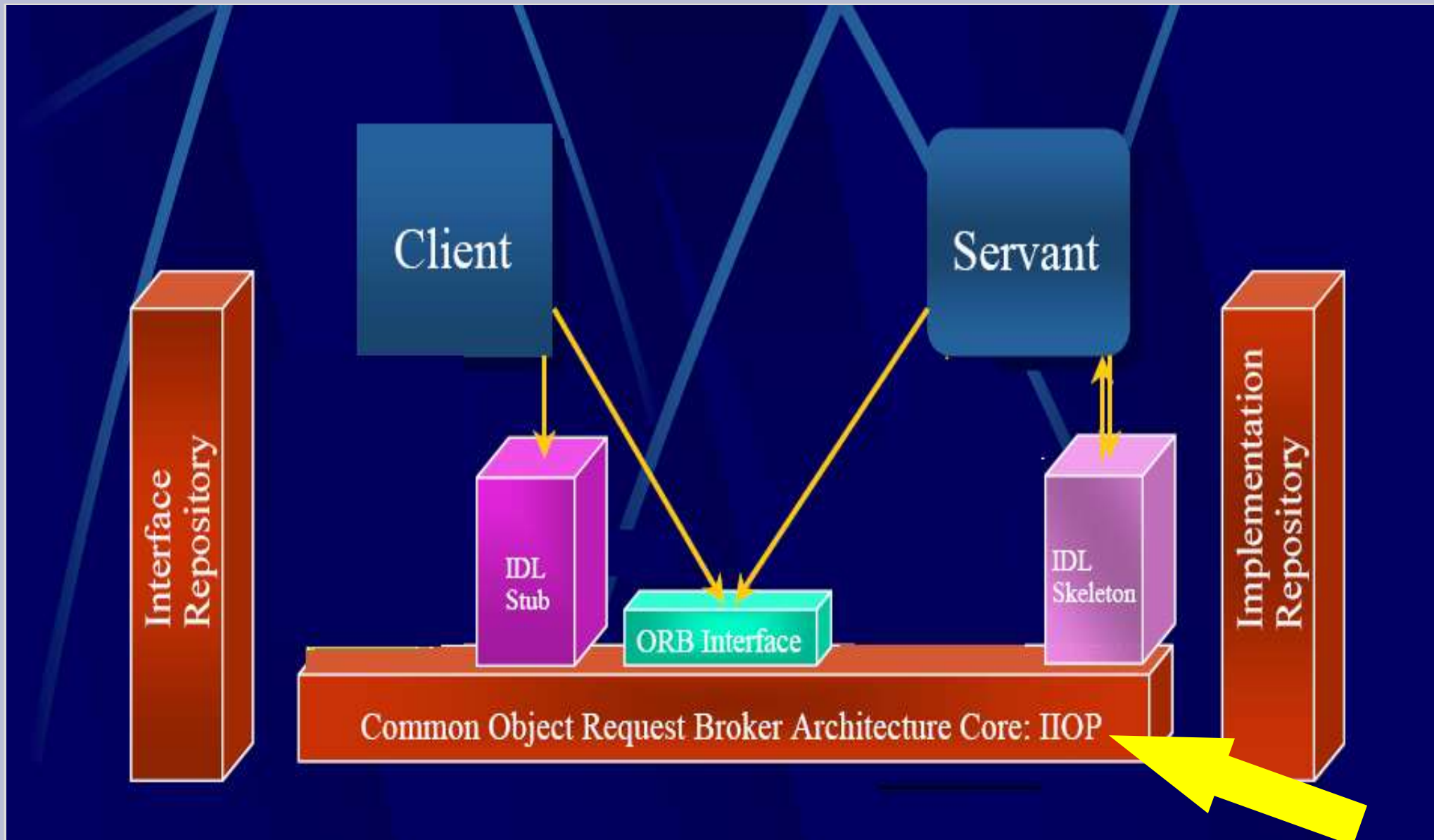
CORBA – tworzenie aplikacji

1. Specyfikacja interfejsu w IDL
2. Generacja pośrednika i szkieletu przez kompilator IDL
3. Implementacja klas obiektu zdalnego
4. Implementacja serwera do obsługi obiektu zdalnego
5. Kompilacja serwera i rejestracja go w ORB
6. Implementacja klienta
7. Kompilacja klienta i uruchomienie go

CORBA i IIOP


Client
Servant
Interface Repository
IDL Stub
ORB Interface
IDL Skeleton
Implementation Repository
Common Object Request Broker Architecture Core: IIOP

CORBA i IIOP


Client
Servant
Interface Repository
IDL Stub
ORB Interface
IDL Skeleton
Implementation Repository
Common Object Request Broker Architecture Core: IIOP

Internet Inter-ORB Protocol

- Założeniem CORBY jest współpraca aplikacji niezależne od ich położena w środowisku sieciowym
- W pierwszych wersjach CORBY były problemy ze współpracą między ORB'ami różnych producentów w różnych środowiskach
- Specyfikacja 2.0 CORBY w 1994, wprowadziła IIOP - Internet Inter-ORB Protocol

GIOP + TCP/IP = IIOP

- IIOP składa się z GIOP i warstwy transportowej
- GIOP - General Inter-ORB Protocol
  - specyfikuje Common Data Representation (CDR)
  - zawiera zbiór komunikatów dla wywołań CORBA
  - zawiera generalne założenia na temat warstwy transportowej
- GIOP może działać z dowolnym protokołem warstwy transportowej np. TCP/IP, Novell SPX, SNA
- Aby zapewnić współpracę, IIOP korzysta z TCP/IP

IOR

- CORBA 2.0 Interoperable Object Reference (IOR) – wspólny format referencji do obiektów wymienianych przez IIOP
- Format ten obejmuje adres internetowy serwera obsługującego obiekt i klucz służący od odnalezienia obiektu na tym serwerze

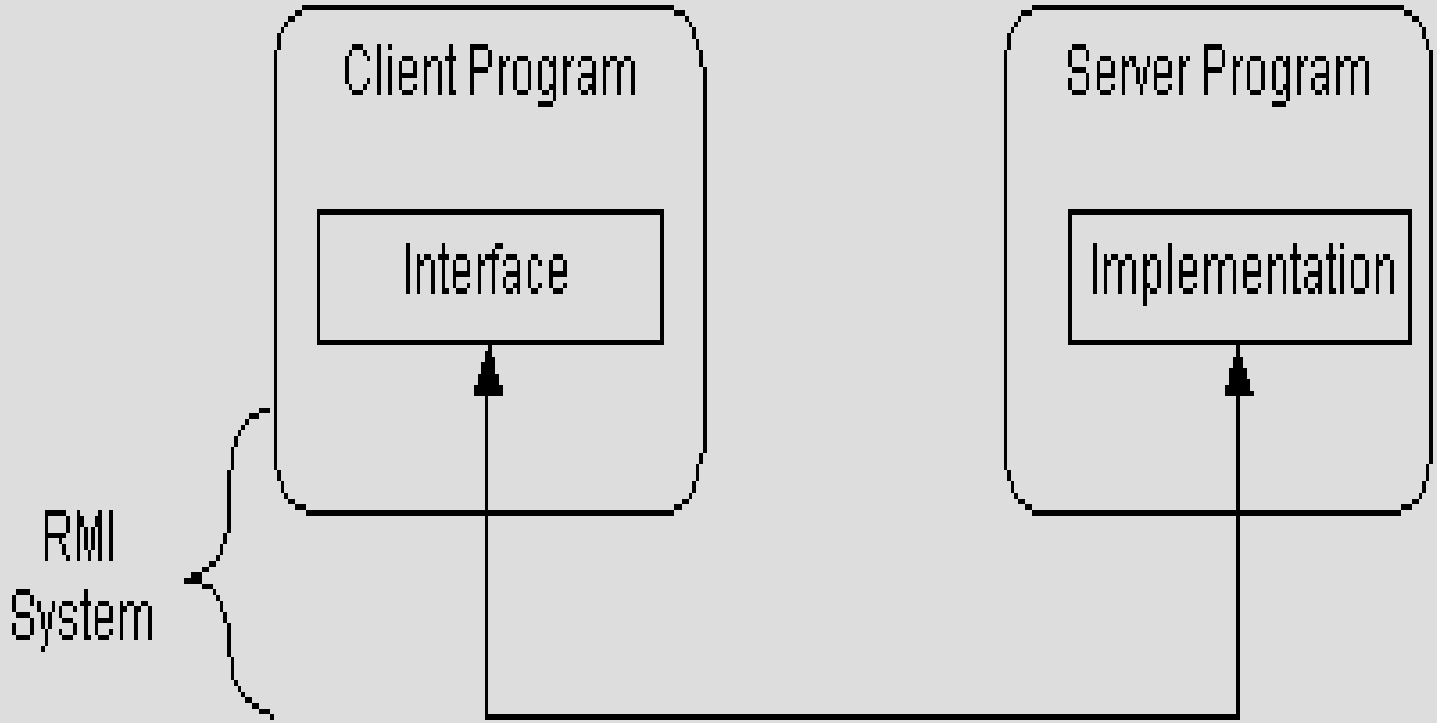
RMI/IIOP

RMI

- Remote Method Invocation
  - rozszerza model obiektów żyjących w jednej maszynie wirtualnej do modelu rozproszonego
  - programista może posługiwać się zdalnymi obiektami w taki sam sposób jak lokalnymi
  - architektura RMI definiuje zachowanie obiektów, zarządzanie wyjątkami, pamięcią, przekazywaniem parametrów i wyników metod

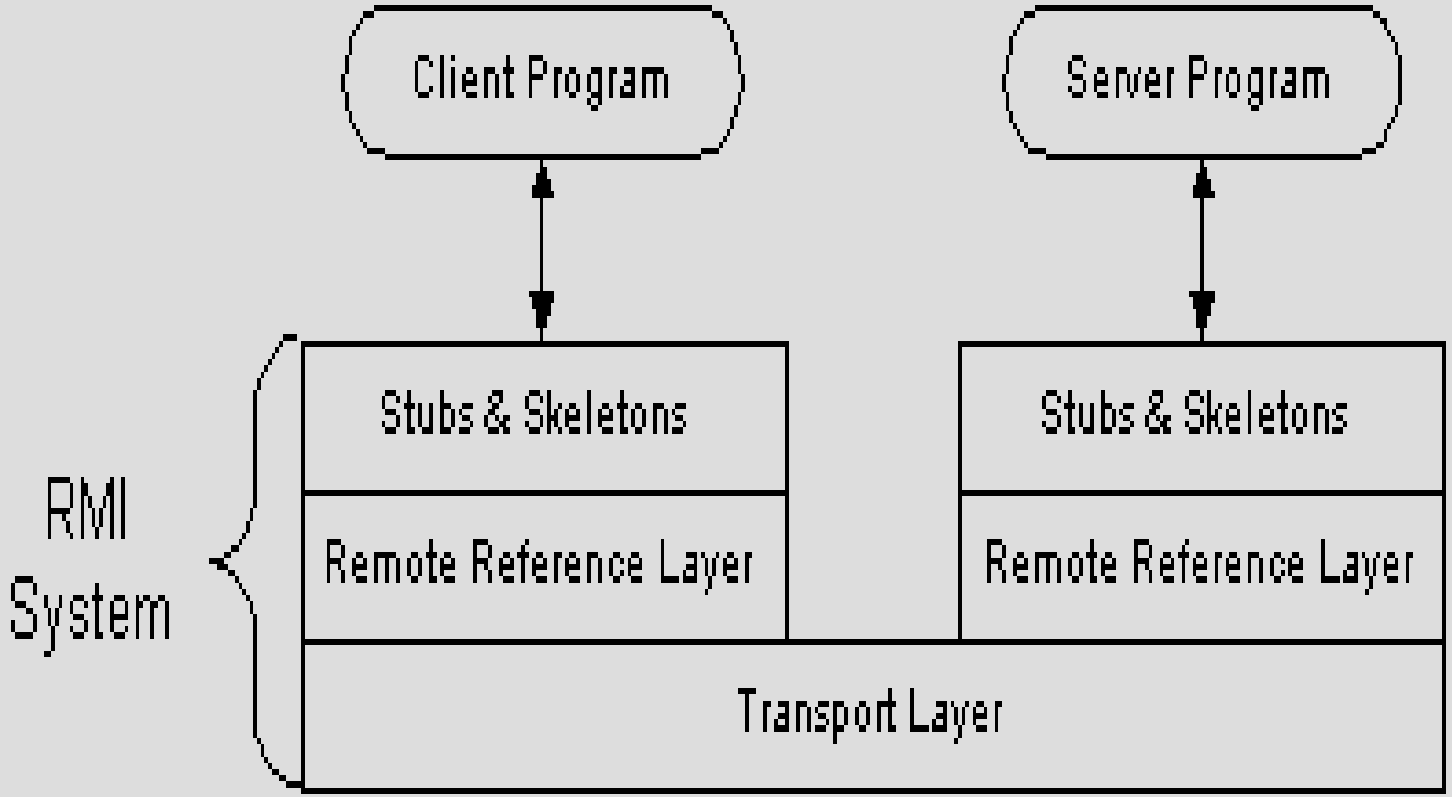
RMI

- Lokalny interfejs i zdalna implementacja



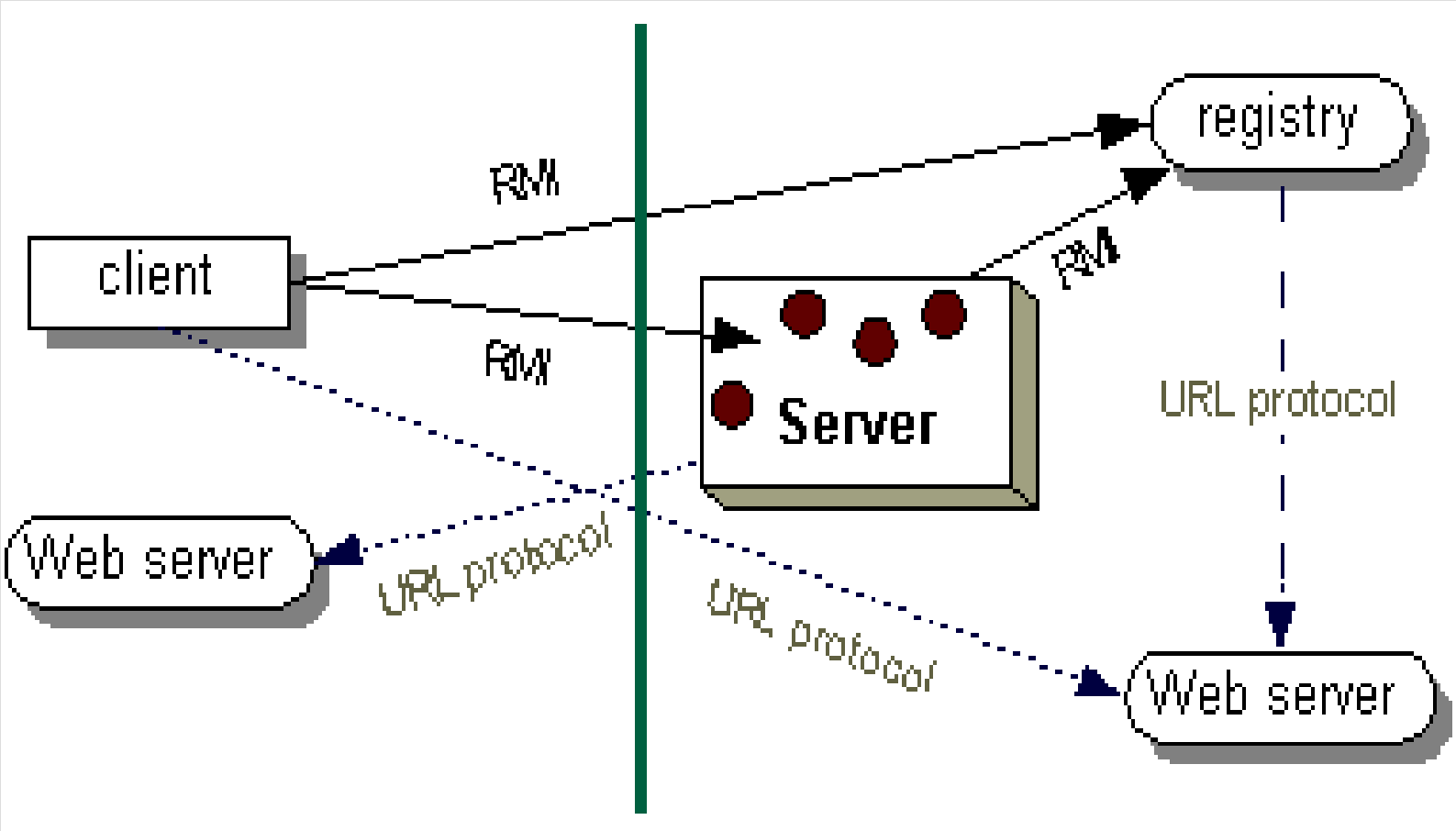
RMI

- Mechanizm zdalnych wywołań



RMI

- Mechanizm zdalnych wywołań



Programowanie RMI

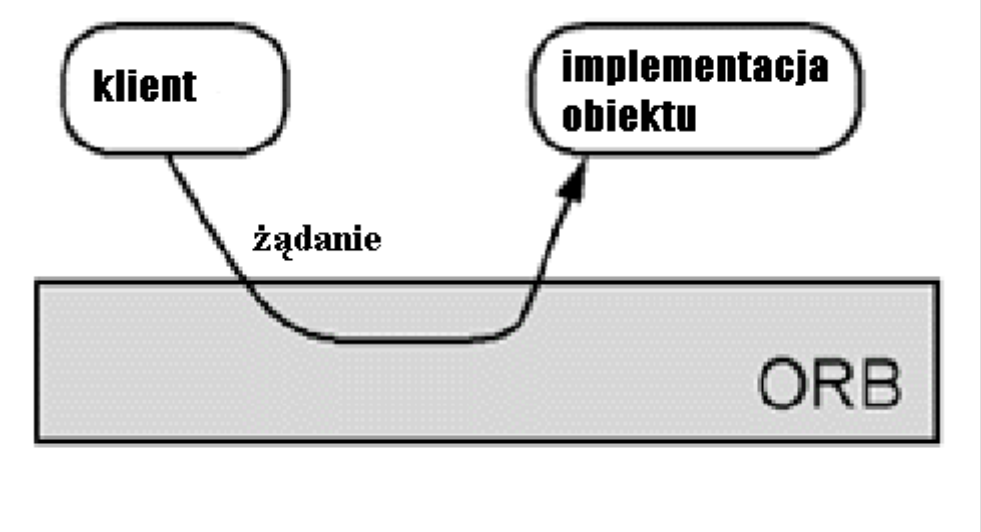
Obiekt lokalny	Obiekt zdalny
Zdefiniowany przez klasę Javy	Definiuje swoje zachowanie przez interfejs, który roszerza interfejs ***Remote***
Jest implementowany przez swoją klasę	Jego metody są wykonywane przez klasę Javy, która rozszerza zdalny interfejs
Nowa instancja jest tworzona operatorem ***new***	Nowa instancja jest tworzona na serwerze operatorem ***new.*** Klient nie może bezpośrednio stworzyć nowego zdalnego obiektu, chyba że używa Java 2 Remote Object Activation
Dostępny bezpośrednio przez referencję	Dostępny przez referencję, która wskazuje na pośrednika (stub), który implementuje zdalny interfejs
W lokalnej maszynie wirtualnej referencja wskazuje bezpośrednio na obiekt na stercie	Referencja wskazuje na pośrednika (stub) na lokalnej stercie. Pośrednik umie połączyć się ze zdalnym obiektem, który zawiera implementację zdalnych metod
Występują wyjątki wykonania (Runtime Exceptions) i wyjątki programu (Exceptions). Kompilator wymusza obsługę wyjątków programu	RMI wymusza obsługę zdalnych wyjątków (Remote Exceptions)

Protokoły w RMI

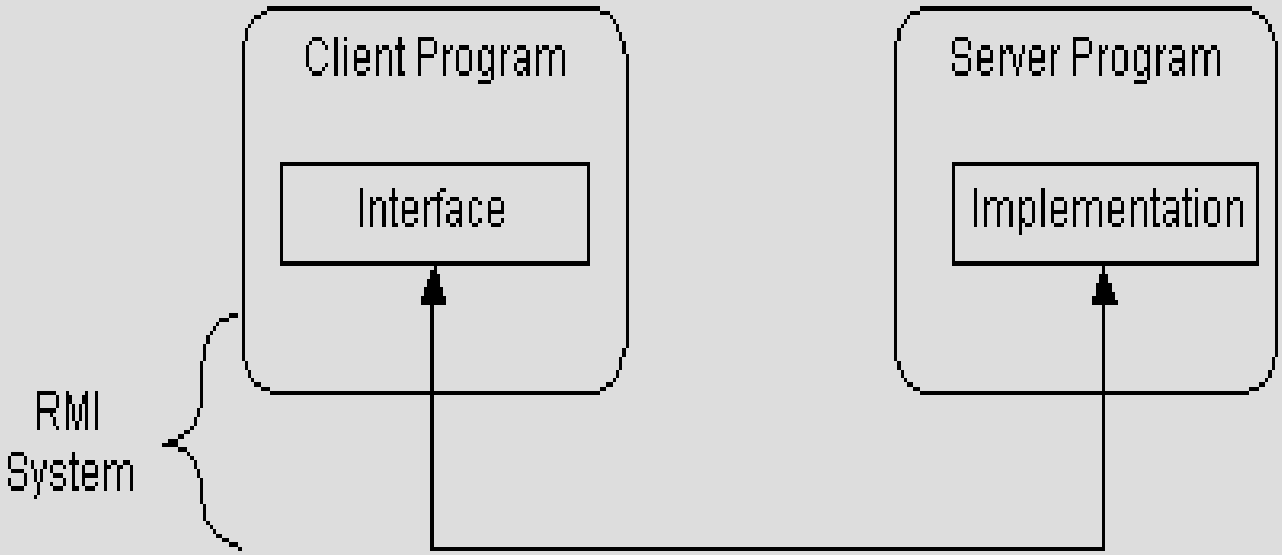
- Java Remote Method Protocol (JRMP)
  - oparty na strumieniach bajtów
  - korzysta z Java Object Serialization
- HTTP
  - żądania POST
  - przechodzi przez firewalle

RMI/IIOP

- CORBA:



- RMI:



RMI/IIOP

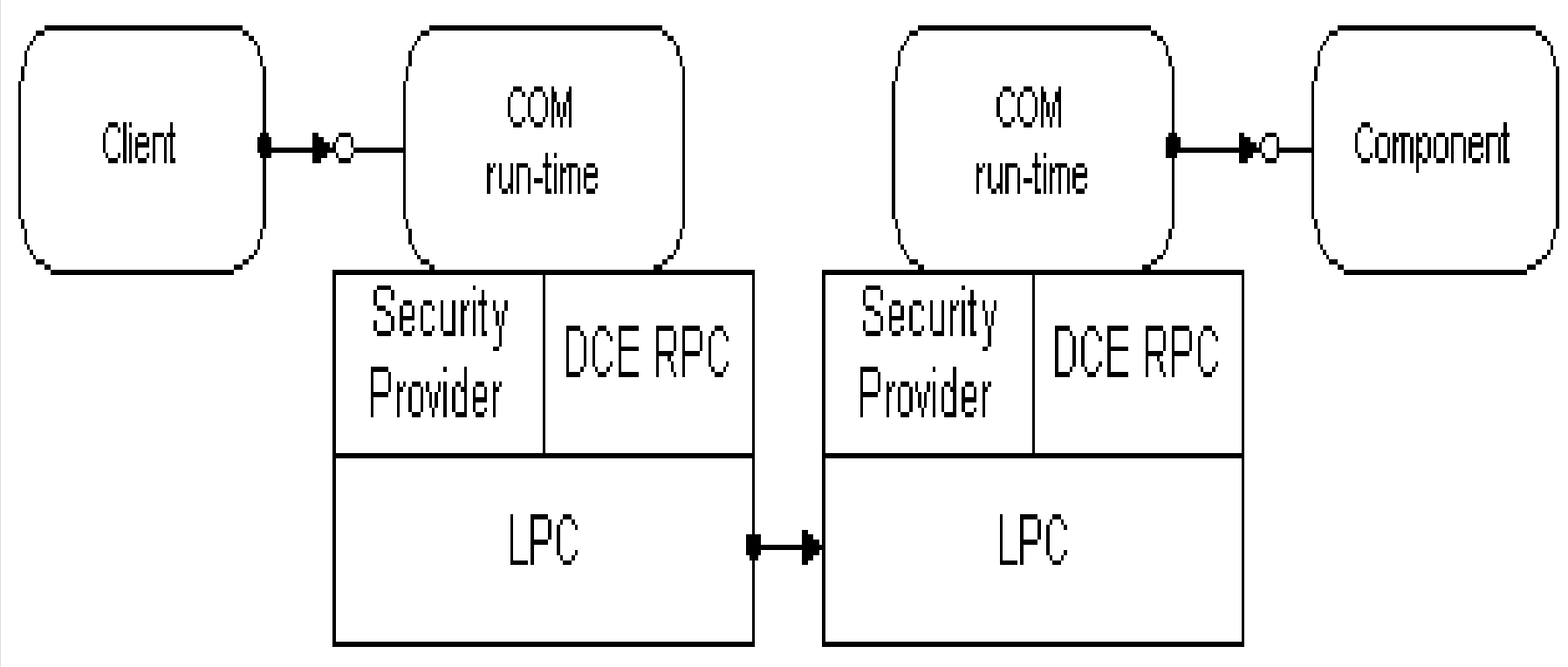
- RMI zamiast JRMP wykorzystuje IIOP
- W ten sposób można uzyskać dostęp do serwerów CORBA za pomocą API RMI
- Ciekawym zastosowaniem są Enterprise Java Beans (EJB), w których model zdalnego obiektu wykorzystuje RMI
- EJB można więc łatwo połączyć z innymi systemami przez CORBĘ, co daje ogromne możliwości integracji

Distributed Common Object Model technologia Microsoftu

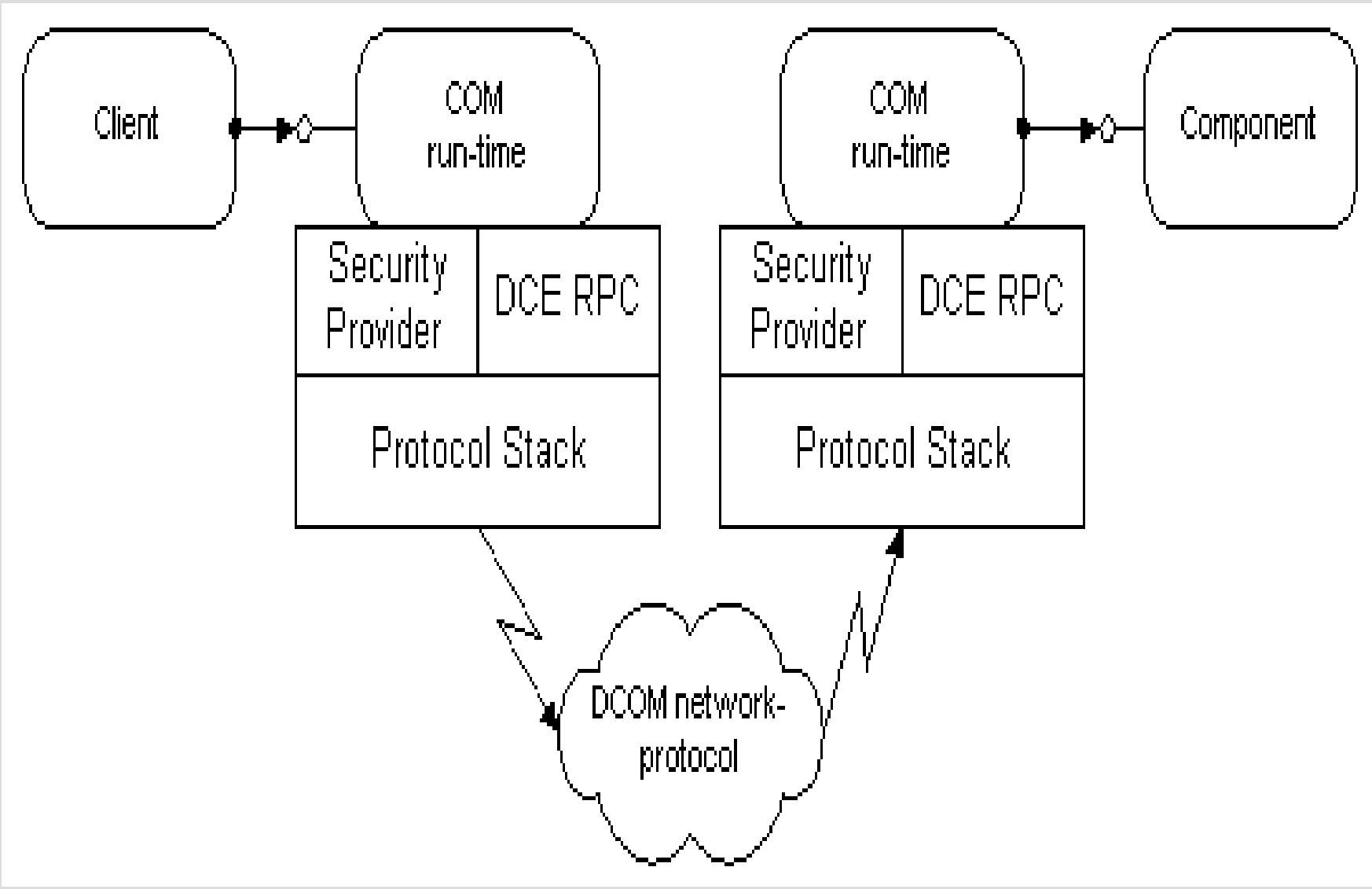
COM

- COM - Common Object Model
  - niezależny od platformy
  - rozproszony
  - system obiektowy
- Podstawa dla
  - OLE (Object Linking and Embedding)
  - ActiveX® (Internet-enabled components)
- Przeznaczony dla języków
  - Visual Basic
  - C++

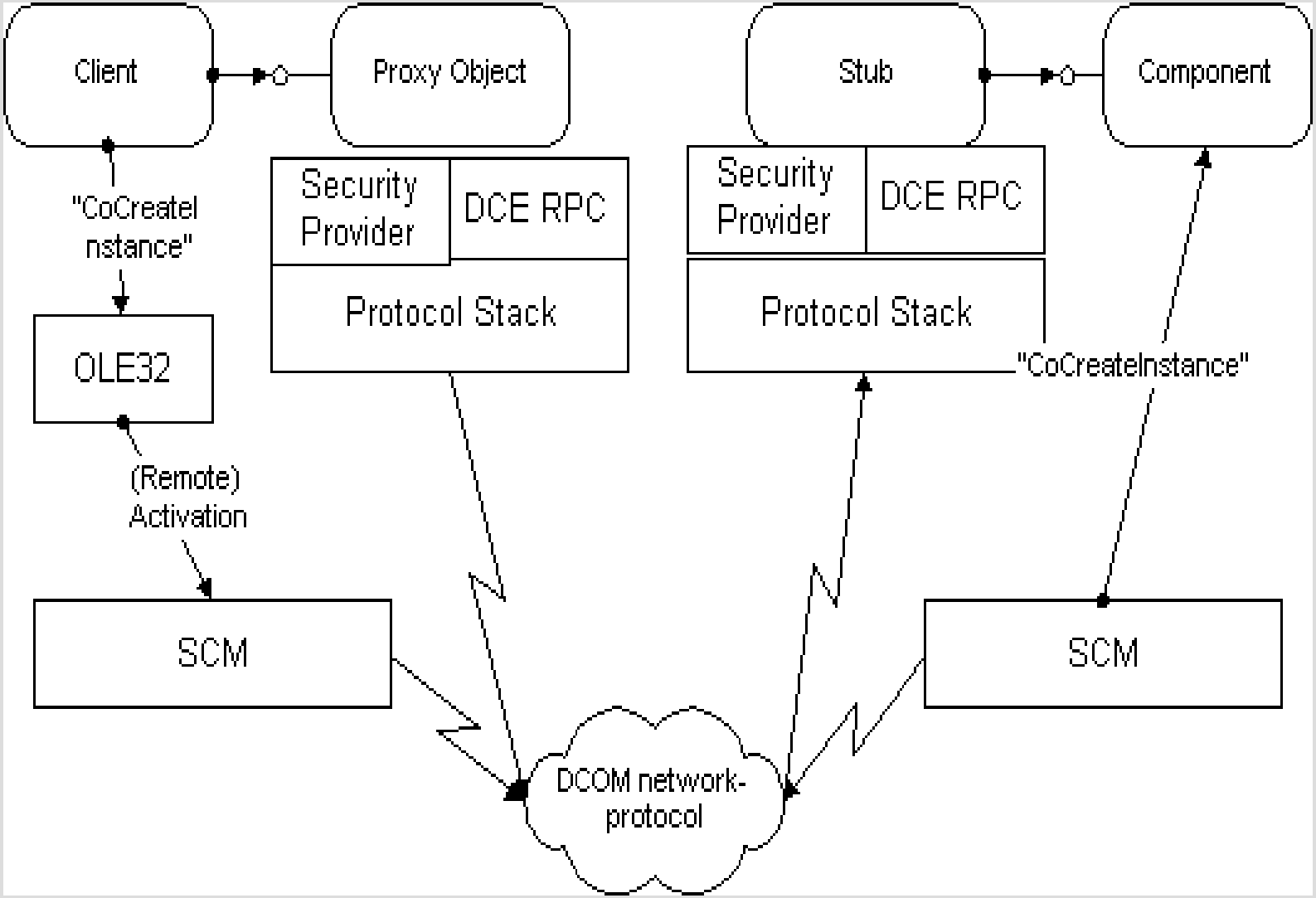
COM


Client
COM
run-time
COM
run-time
Component
Security
Provider
DCE RPC
Security
Provider
DCE RPC
LPC
LPC

DCOM


Client
COM
run-time
COM
run-time
Component
Security
Provider
DCE RPC
Security
Provider
DCE RPC
Protocol Stack
Protocol Stack
DCOM network-
protocol

DCOM - Architektura



DCOM - Architektura

- Identyczna jak RMI
- Pośredniki (proxy i stub)
- Opakowywanie danych przesyłanych przez sieć (marshalling)
- Wywołanie metody = wywołanie RPC między proxy i stub
- Rozszerzenie MS-RPC o Object RPC (ORPC)

SOAP – Simple Object Access Protocol

SOAP
Simple Object Access Protocol

- Definiuje format wysyłania wiadomości oparty na XML
- Niezależny od platformy i języka
- Prosty i rozszerzalny
- Komunikaty przechodzą przez firewalle
- Standard W3C
- Kluczowy element platformy .NET

SOAP – przykładowa wiadomość

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP – przykładowa wiadomość

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP – przykładowa wiadomość

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>

...
</soap:Header>

<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP – przykładowa wiadomość

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP – przykładowe żądanie

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-
   encoding">

<soap:Body>
   <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
      <m:Item>Apples</m:Item>
   </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

SOAP – przykładowa odpowiedź

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-
    encoding">

<soap:Body>
   <m:GetPriceResponse
    xmlns:m="http://www.w3schools.com/prices">
      <m:Price>1.90</m:Price>
   </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP – stosowane protokoły

- SOAP używa HTTP na TCP/IP
- Korzysta z żądań GET lub POST do przesyłania wiadomości

Literatura

- Specyfikacja CORBA
  - http://www.omg.org/
- RMI/IIOP
  - http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/
- EJB/CORBA
  - www.iona.com/whitepapers/CORBA-EJBInteropWPV02-00.pdf
- DCOM
  - http://www.microsoft.com/com/
- SOAP
  - http://www.w3.org/TR/soap/