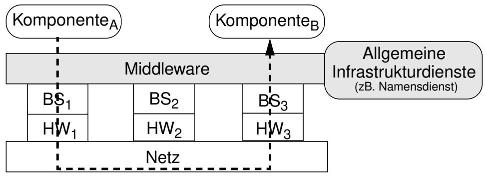
Einführung in CORBA

Kay Römer
Institut für Informationssysteme
ETH Zürich

Middleware

- Infrastruktur für verteilte Systeme
- Unterstützt Enwickler bei Behandlung der Probleme verteilter Systeme
- Erleichtert damit Entwicklung verteilter Anwendungen
- Logisch zwischen Betriebssystem und Anwendung



Middlewareplattform CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- Offene, frei verfügbare Spezifikation einer objektorientierten Middleware-Plattform
- Zentrale Komponente: Object Request Broker (ORB)
  - Verteilte Objekte als Berechnungseinheiten
  - Vermittlung von Methodenaufrufen auf Objekten
- Weitere Bestandteile:
  - System-Dienste
  - Horizontale und vertikale Anwender-Dienste

CORBA: Eigenschaften

- Objekte als Berechnungseinheiten:

  Methoden von Objekten können (entfernt) aufgerufen werden
- Ortstransparenz:

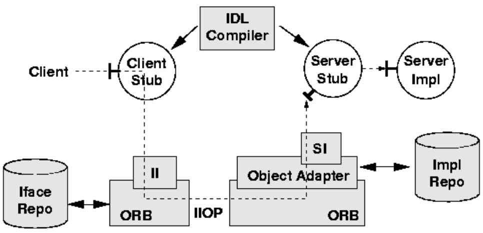
  Für den Aufrufer ist es egal, wo sich ein Objekt befindet
- Programmiersprachenunabhängigkeit:

  Komponenten einer verteilten Anwendung können in verschiedenen Programmiersprachen entwickelt werden
- Hardware-, Betriebssystemunabhängigkeit:

  CORBA kann prinzipiell auf jeder Hardware und unter jedem Betriebssystem verwendet werden
- Herstellerunabhängigkeit:

  Verschiedene CORBA-Implementationen sind interoperabel, wenn sie sich an die Spezifikation halten

CORBA: Architektur 1



- Object Request Broker (ORB):

  Weiterleitung von Methodenaufrufen zwischen Client und Server
- Object Adapter (OA):

  Anbindung von Objekt-Implementationen an den ORB (wir verwenden POA)

CORBA: Architektur 2

- Internet Inter ORB Protocol (IIOP)

  Kommunikationsprotokoll zwischen ORBs
- IDL-Compiler:

  Erzeugt Stubs aus einer IDL-Spezifikation
- Invocation Interface (II):

  Schnittstelle des ORB zum Absetzen von Methodenaufrufen
- Skeleton Interface (SI):

  Schnittstelle der Object Adapter zu den Server-Stubs
- Interface Repository (IR):

  Datenbank mit Typinformationen (für Typüberprüfung und Reflection)
- Implementation Repository (IMR):

  Datenbank mit Serverinformationen (für automatische Server-Aktivierung)

Interface Definition Language (IDL)

- Programmiersprachenunabhängigkeit erfodert Spezifikation der Schnittstellen der CORBA-Objekte in einer speziellen Sprache (IDL)
- Ähnlich C++, jedoch nur Typedeklarationen, keine Programmkonstrukte
- IDL-Konstrukte: **Basistypen** (e.g., long), **konstruierte Typen** (e.g., struct), **Template-Typen** (e.g., sequence), **Interfaces**, **Exceptions**, **Module**
- Language-Mapping definiert Abbildung von IDL in eine bestimmte Programmiersprache
- IDL-Compiler übersetzt IDL-Spezifikation in Stubs in der gewünschten Programmiersprache

IDL: Basistypen

- Ganzzahltypen: `short, long, long long`
- Vorzeichenlose Ganzzahltypen: `unsigned short, unsigned long, unsigned long long`
- Fliesskommatypen: `float, double, long double`
- Festkommatypen: `fixed<digits, scale>`
- Boolesche Variable: `boolean`
- Zeichen: `char, wchar`
- Byte: `octet`
- Werte beliebigen Typs: `any`

IDL: Template-Typen

- String: `string, wstring`
- Bounded String: `string<maxlen>, wstring<maxlen>`
- Sequence: `sequence<type>`
- Bounded Sequence: `sequence<type,maxlen>`
- Array: `type[size]`

IDL: Konstruierte Typen

- Exception: `exception Error { short code; };`
- Aufzählungen: `enum Color { Red, Blue };`
- Strukturen: `struct Point { float X; float Y; };`
- Union:

```
union PackedInt switch (short) {
case 2: short int2;
case 4: long int4;
default: long long int8;
};
```


IDL: Interfaces 1

- Deklaration der Schnittstelle eines CORBA-Objektes:

```
interface Konto {
  string konto_nummer();
  void einzahlen(in unsigned long betrag);
};
```

- Parameter werden mit einer Richtung versehen (`in`, `out` oder `inout`)
- Methoden können Exceptions werfen:

```
interface Konto {
  void abheben (in unsigned long betrag)
    raises (BankException);
};
```


IDL: Interfaces 2

- Vererbung (auch von mehreren Interfaces):

```
interface GiroKonto : Konto {
   attribute unsigned long kredit_limit;
};
```

- Attribute können abgefragt und gesetzt werden
- Überladen von Methoden (gleicher Methodenname mit verschiedenen Parametern) ist nicht möglich

IDL: Module

- Zusammengehörige Deklarationen können durch ein Modul zusammengefasst werden:

```
module Banking {
  exception BankException { ... };
  interface Konto { ... };
  ...
};
```

- Auf Elemente eines Moduls kann mittels des Scope-Operators zugegriffen werden:

```
Banking::Konto
```


Abbildung von IDL nach Java/C++

- Abbildung von IDL-Konstrukten auf Konstrukte in der entsprechenden Programmiersprache
- Teil der CORBA-Spezifikation (für C, C++, Java, Smalltalk, Ada, ...)
- Wird vom IDL-Compiler implementiert

Java/C++-Mapping: Basistypen

IDL	Java	C++
short	short	CORBA::Short
long	int	CORBA::Long
long long	long	CORBA::LongLong
unsigned short	short	CORBA::UShort
unsigned long	int	CORBA::ULong
unsigned long long	long	CORBA::ULongLong
float	float	CORBA::Float
double	double	CORBA::Double
long double	-	CORBA::LongDouble
char	char	CORBA::Char
wchar	char	CORBA::WChar
boolean	boolean	CORBA::Boolean
octet	byte	CORBA::Octet
any	CORBA.any	CORBA::Any
fixed	math.BigDecimal	CORBA::Fixed

Java/C++-Mapping: Template-Typen

IDL	Java	C++
string, string<X>	lang.String	char*
wstring, wstring<X>	lang.String	CORBA::WChar*
Array	Klasse mit Array	Array

- In C++ werden Helferfunktionen für die Behandlung von Strings bereitgestellt:

```
char *CORBA::string_dup (const char *s);
char *CORBA::string_alloc (CORBA::Long len);
void CORBA::string_free (char *s);
```

- Ausserdem Helferklassen `CORBA::String_var` und `CORBA::WString_var` zur "automatischen" Speicherverwaltung

Java/C++-Mapping: Konstruierte Typen

IDL	Java	C++
enum	Klasse mit Konstanten	enum
struct	Klasse mit Members	struct
exception	Klasse mit Members von `Exception` abgeleitet	Klasse von `CORBA::UserException` abgeleitet
union	Klasse mit Zugriffsmethoden	Klasse mit Zugriffsmethoden

Java-Mapping: Interfaces

```
interface Hello { void hello(); }; // IDL
```

- Java-Interface mit gleichem Namen:

```
public interface Hello { void hello(); };
```

- Basisklasse für die Implementierung:

```
public class HelloPOA { ... };
```

- Hilfsklasse mit einigen nützlichen Operationen:

```
public class HelloHelper {
  public static Hello narrow (Object o);  ... };
```

- Attribute werden auf zwei Methoden abgebildet, `attribute short x` in IDL wird zu:

```
void x (short wert);
short x ();
```

C++-Mapping: Interfaces

- Klasse mit gleichem Namen:

```
class Hello {
  void hello ();
  static Hello_ptr _narrow(CORBA::Object_ptr o);
  static Hello_ptr _duplicate(Hello_ptr o); };
```

- Basisklasse für die Implementierung:

```
class Hello_skel { ... };
```

- Klasse zur "automatischen" Speicherverwaltung:

```
class Hello_var { ... };
```

- Attribute werden auf zwei Methoden abgebildet, `attribute short x` in IDL wird zu:

```
void x (short wert);
short x () const;
```


Java/C++-Mapping: Module

- In Java: Abbildung auf `package`
- In C++: Abbildung auf `namespace`

ORB-API (Java)

- Umwandlung von Objekt-Adressen in Strings:

```
String ORB.object_to_string (CORBA.Object o);
CORBA.Object ORB.string_to_object (String s);
```

- Initialisierung:

```
CORBA.ORB ORB.init (String args[], Properties props);
```

- Event-Loop (nur im Server):

```
ORB.run ();
ORB.shutdown (boolean wait);
```


POA-API (Java)

- Initialisierung:

```
CORBA.Object obj = orb.resolve_initial_references("Root-
POA");
POA poa = POAHelper.narrow (obj);
```

- Aktivierung:

```
poa.the_POAManager().activate();
```

- Anmelden eines Objektes (vom Typ X):

```
CORBA.Object obj = poa.servant_to_reference (x_impl);
X x = XHelper.narrow (obj);
```

- Abmelden eines Objektes (vom Typ X):

```
poa.deactivate_object (poa.reference_to_id (x));
```


ORB-API (C++)

- Umwandlung von Objekt-Adressen in Strings:

```
char *ORB::object_to_string (CORBA::Object_ptr o);
CORBA::Object_ptr CORBA::string_to_object (char *s);
```

- Initialisierung:

```
CORBA::ORB_ptr CORBA::ORB_init (int &argc, char *argv[]);
```


Beispiel: IDL

```
// Datei: hello.idl

module H {
  interface Hello {
    void hello (in string msg);
  };
};
```

Beispiel: Objekt-Implementierung (Java)

```
// Datei: HelloImpl.java

import H.*;
import java.io.*;

public class HelloImpl extends HelloPOA {
  public void hello (String msg)
  {
    System.out.println (msg);
  }
};
```


Beispiel: Server (Java)

```
// Datei: Server.java
import H.*;
import java.io.*;
import java.util.*;

import org.omg.CORBA.*;
import org.omg.PortableServer.*;

public class Server
{
  public static void main (String args[])
      throws Exception
  {
    // ORB Initialisierung
    ORB orb = ORB.init (args, null);

    // POA Initialisierung und Aktivierung
    POA poa = POAHelper.narrow(
        orb.resolve_initial_references("RootPOA"));
    poa.the_POAManager().activate();
```



```
    // Objekt erzeugen, beim POA anmelden
    // und Objektreferenz erzeugen
    HelloImpl h = new HelloImpl();
    Hello ref = HelloHelper.narrow (
        poa.servant_to_reference (h));

    // Objektreferenz in Datei schreiben
    BufferedWriter out =
      new BufferedWriter (new FileWriter ("hello.ref"));
    out.write (orb.object_to_string (ref));
    out.newLine();
    out.close();

    // Event-loop des ORB
    orb.run();

    // Objekt wieder abmelden
    poa.deactivate_object (poa.reference_to_id (ref));
  }
}
```


Beispiel: Client (C++)

```
// Datei: Client.cc

#include <hello.h>
#include <iostream.h>
#include <fstream.h>

int main (int argc, char *argv[]) {
  CORBA::ORB_var orb = CORBA::ORB_init (argc, argv);

  ifstream istr ("hello.ref");
  char ref[1000];
  istr >> ref;

  CORBA::Object_var o = orb->string_to_object (ref);
  H::Hello_var h = H::Hello::_narrow (o);
  h->hello ("hello world!");

  return 0;
}
```


Beispiel: Übersetzen

- Java-Code übersetzen:

```
idlj -fall hello.idl
javac Server.java
```

- C++-Code übersetzen:

```
idl hello.idl
mico-c++ -c hello.cc
mico-c++ -c Client.cc
mico-ld -o Client hello.o Client.o -lmico2.3.11
```

- Ausführen:

```
java Server
./Client
```