CORBA

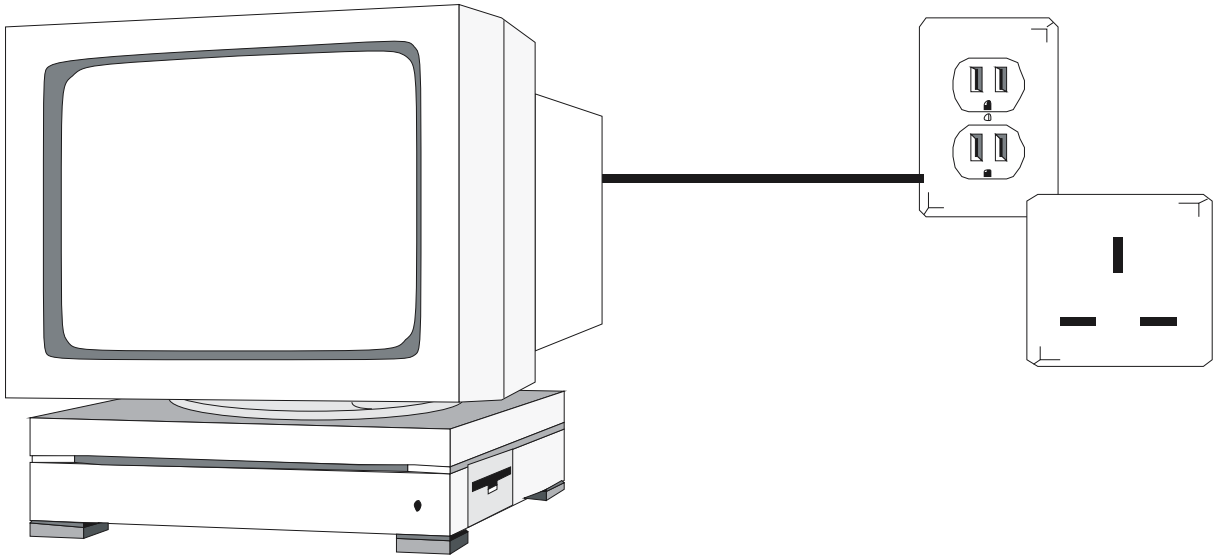
- Common Object Request Broker Architecture

CORBA

- Common ((Object Request) Broker) Architecture
- Standaryzacja: Object Management Group (http://www.omg.org)
- Obecnie wersja specyfikacji 3.0
- M. Henning, S. Vinoski: Advanced CORBA Programming with C++, Addison-Wesley 1999

Global Information Appliance

- Podłączenie urządzenia do sieci informatycznej równie łatwe, jak do sieci zasilającej



Obszar działania OMG

- Integracja aplikacji
  - przetwarzanie rozproszone
- Systemy przetwarzania informacji pochodzących z różnych źródeł
  - heterogeniczych
  - sieciowych
  - różnych producentów

Poprzednie podejścia

- Zbyt niskopoziomowe
  - dobre elementy składowe, ale nie dla poziomie interesującym dla projektanta aplikacji
- Brak standaryzacji na wyższym poziomie

Powstawanie standardów OMG

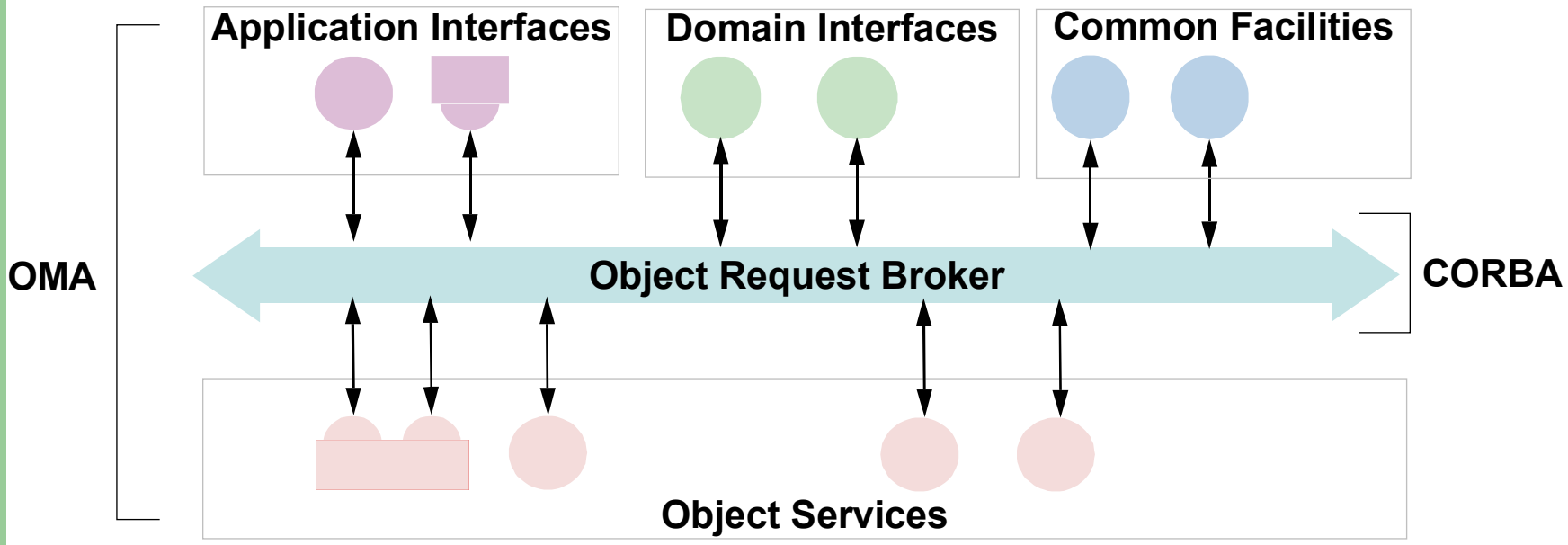
- OMG wybiera interfejsy
  - z konkurujących propozycji przemysłu
- OMG publikuje interfejsy
  - dostępne bezpłatnie dla każdego
- OMG kontroluje interfejsy
  - należą one do OMG, kontroluje ona ich rozwój
- OMG współpracuje z ciałami standaryzującymi
  - specyfikacje stają się standardami

OMG jest neutralna

- Nie tworzy ani nie sprzedaje implementacji
- Nie testuje implementacji
  - testy wykonuje X/Open

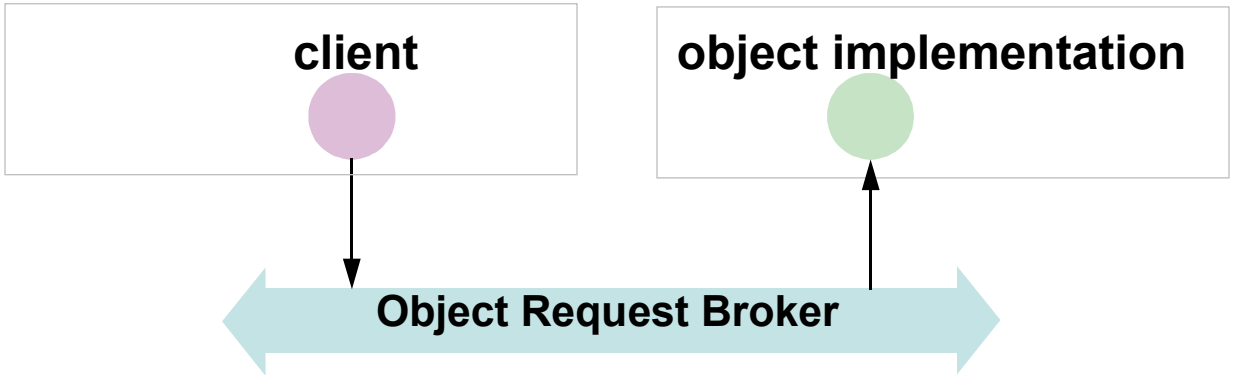
Object Management Architecture

- Object Request Broker + obiekty
  - Object services, Common Facilities, Domain Interfaces, Application Interfaces



Object Request Broker (ORB)

- Obiekty uzyskują dostęp do innych obiektów za pośrednictwem ORB
  - ORB lokalizuje implementacje obiektów i zajmuje się komunikacją z nimi
  - klient i serwer mogą być napisane w różnych językach i działać na różnych typach komputerów



Object Services (CORBA services)

- Podstawowe usługi na poziomie systemu
  - są obiektami
  - mają specyfikacje interfejsów
  - możliwe różne implementacje
- Przykłady
  - Transakcje
  - Współbieżność
  - Zdarzenia

Common Facilities (CORBA facilities)

- Obiekty współdzielone przez aplikacje
  - e-mail
  - drukowanie
- Umożliwiają współpracę produktów różnych producentów

Domain interfaces

- Standardowe obiekty dla różnych dziedzin zastosowań
  - przetwarzanie danych geologicznych
  - zarządzanie systemami

Application interfaces

- Specyficzne dla danej aplikacji
  - dostarczane przez niezależnych producentów oprogramowania
  - dostarczone przez użytkownika końcowego
- Nie specyfikowane przez OMG

Special Interest Groups

- Internet
- Realtime
- Electronic Commerce

CORBA jest przenośna

- Platformy i systemy operacyjne
- Języki programowania: C C++ SmallTalk Ada95 COBOL Java LISP PL/1 Python

CORBA – interfejs programisty

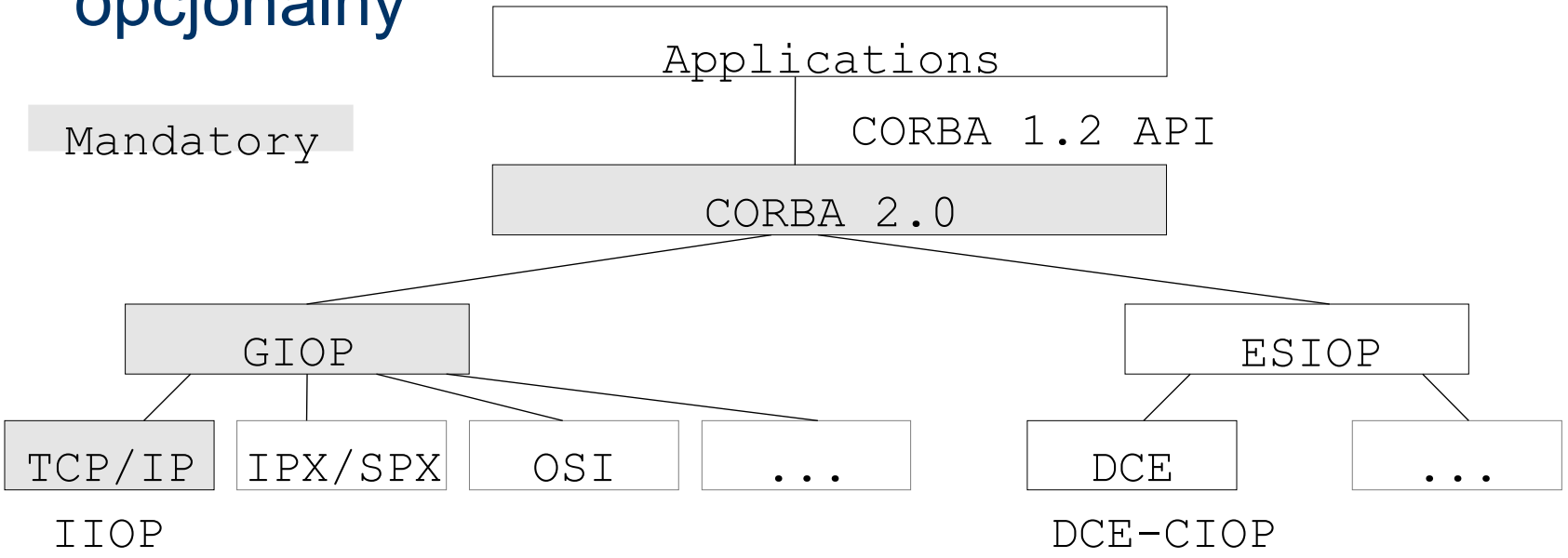
- CORBA jest zorientowana obiektowo
- Wymaga stosowania zasad programowania obiektowego
  - niekoniecznie w języku zorientowanym obiektowo (np. C)

Przenośność, różnorodność i zdolność do współpracy

- Nie wymaga tej samej platformy sprzętowej i tego samego języka programowania w całej aplikacji
  - klient w Javie na palmtopie
  - serwer w COBOLU na mainframe
- Umożliwia późniejszą zmianę platformy i języka programowania
- Wymaga współpracy pomiędzy implementacjami różnych producentów

Protokoły używane w standardzie CORBA

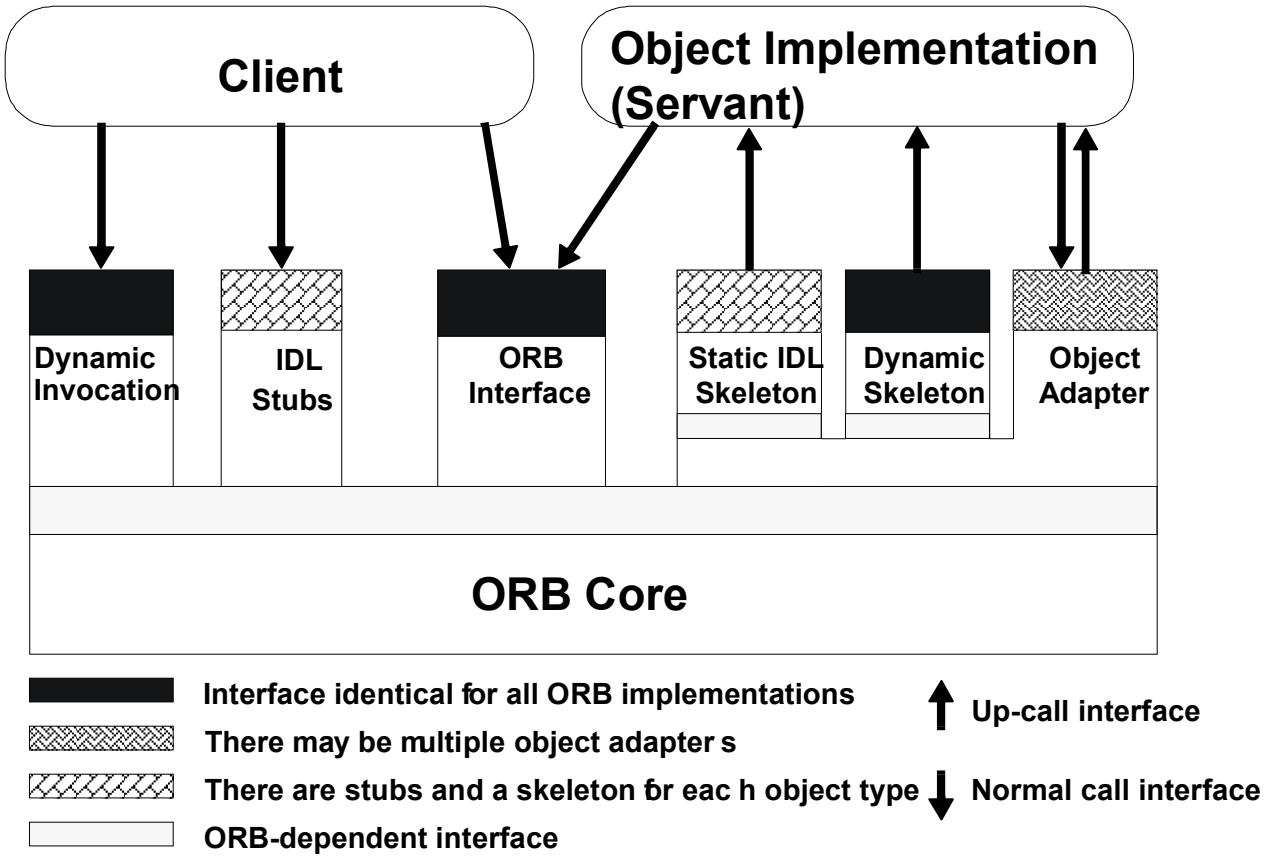
- IIOP (Internet Inter-ORB Protocol) jest obowiązkowy
- DCE-CIOP (Common Inter-ORB Protocol) jest opcjonalny



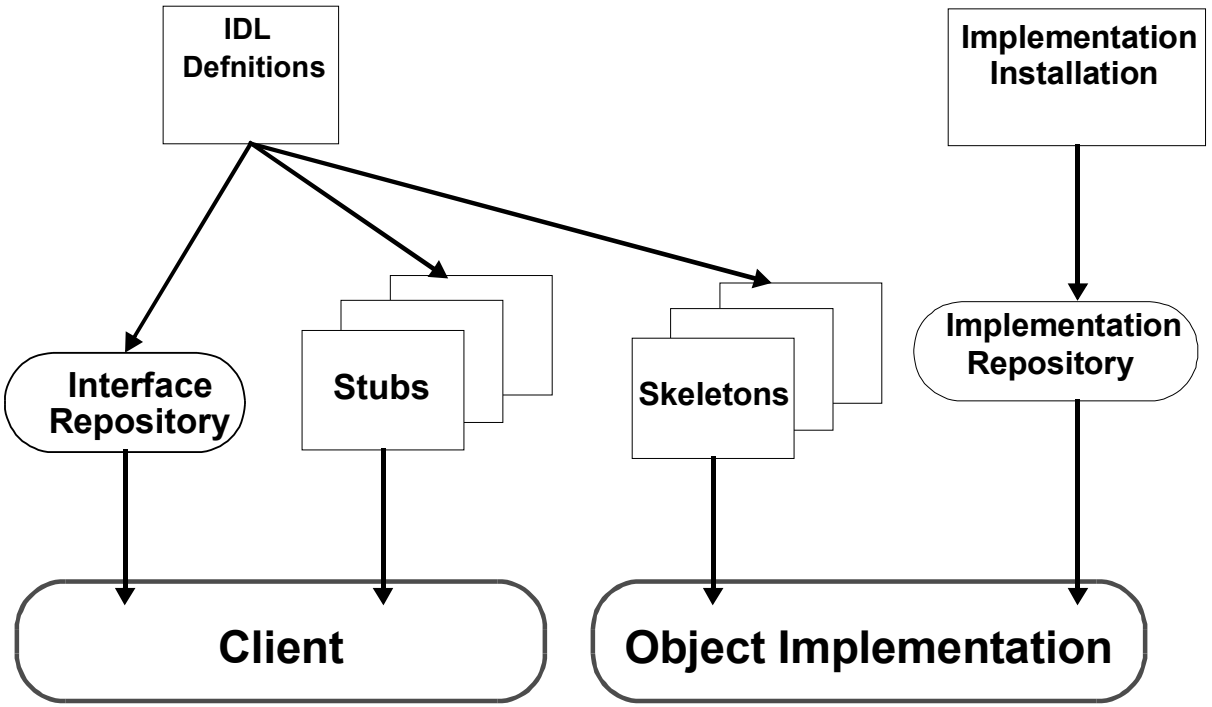
Założenia projektowe GIOP i IIOP

- Możliwie szeroka dostępność
- Prostota
- Skalowalność
- Niski koszt
- Ogólność
- Neutralność pod względem architektury

Przepływ sterowania przy wywołaniu operacji



Interface Repository i Implementation Repository



Typy wywołań

- Synchroniczne (synchronous)
  - blokujące do momentu otrzymania odpowiedzi
- Opóźnione synchroniczne (deferred synchronous)
  - wysyłamy zapytanie i sprawdzamy, czy przyszła odpowiedź
- Jednokierunkowe (oneway)
  - nie gwarantuje przesłania zapytania
- Asynchroniczne (asynchronous, CORBA 3.0)
  - callback lub polling

OMG Interface Definition Language

```
interface Employee {
    long number();
};

interface  EmployeeRegistry {
    Employee lookup(in long emp_number);
};
```

Dziedziczenie interfejsów

```
interface Printer {
    void print();
};

interface  ColorPrinter: Printer {
    enum ColorMode {BlackAndWhite, FullColor };
    void set_color(in ColorMode mode);
};
```

- Wszystkie interfejsy są niejawnie dziedziczone z interfejsu **`Object`**

Wywołanie i przesłanie operacji

- Statyczne wywołanie i przesłanie
  - Stubs i skeletons
- Dynamiczne wywołanie i przesłanie
  - Dynamic Invocation Interface
  - Dynamic Skeleton Interface

Object Adapters

- Tworzą object references, umożliwiając adresowanie obiektów
- Zapewniają ucieleśnienie (incarnation) obiektów przez serwer
- Odbierają zgłoszenia od ORB po stronie serwera i przesyłają do odpowiednich serwantów
- Portable Object Adapter

Wywołanie operacji przez ORB

- Lokalizuje obiekt docelowy
- Aktywuje serwer, jeżeli nie jest uruchomiony
- Przesyła argumenty operacji do serwera
- Aktywuje serwant, jeżeli jest to niezbędne
- Czeka na wykonanie operacji
- Zwraca do klienta parametry **`out`**, **`inout`** i wartość funkcji przy pomyślnym wykonaniu operacji
- Zwraca wyjątek przy błędnym wykonaniu operacji

Cechy wywołania operacji w systemie CORBA

- Niezależność od lokalizacji (location transparency)
- Niezależność od serwera (server transparency)
- Niezależność od języka
- Niezależność od implementacji
- Niezależność od architektury
- Niezależność od systemu operacyjnego
- Niezależność od protokołu
- Niezależność od warstwy transportowej

Semantyka object references

- Każda referencja identyfikuje dokładnie jeden obiekt
- Wiele referencji może odnosić się do tego samego obiektu
- Referencje mogą być puste (nie wskazywać żadnego obiektu)
- Referencje mogą wskazywać na nieistniejące obiekty (dangling references)
- Referencje są nieprzezroczyste dla klienta
- Referencje są mocno utypowione
- Referencje wspierają późne dowiązanie
- Referencje mogą być trwałe
- Referencje mogą być używane przez różne ORBy

Uzyskiwanie referencji

- Jako rezultat operacji
- Przez standardowe usługi, takie jak Naming Service albo Trading Service
- Przez zamianę na łańcuch tekstowy i zapis/odczyt z pliku
- Innymi metodami (e-mail, strona www)

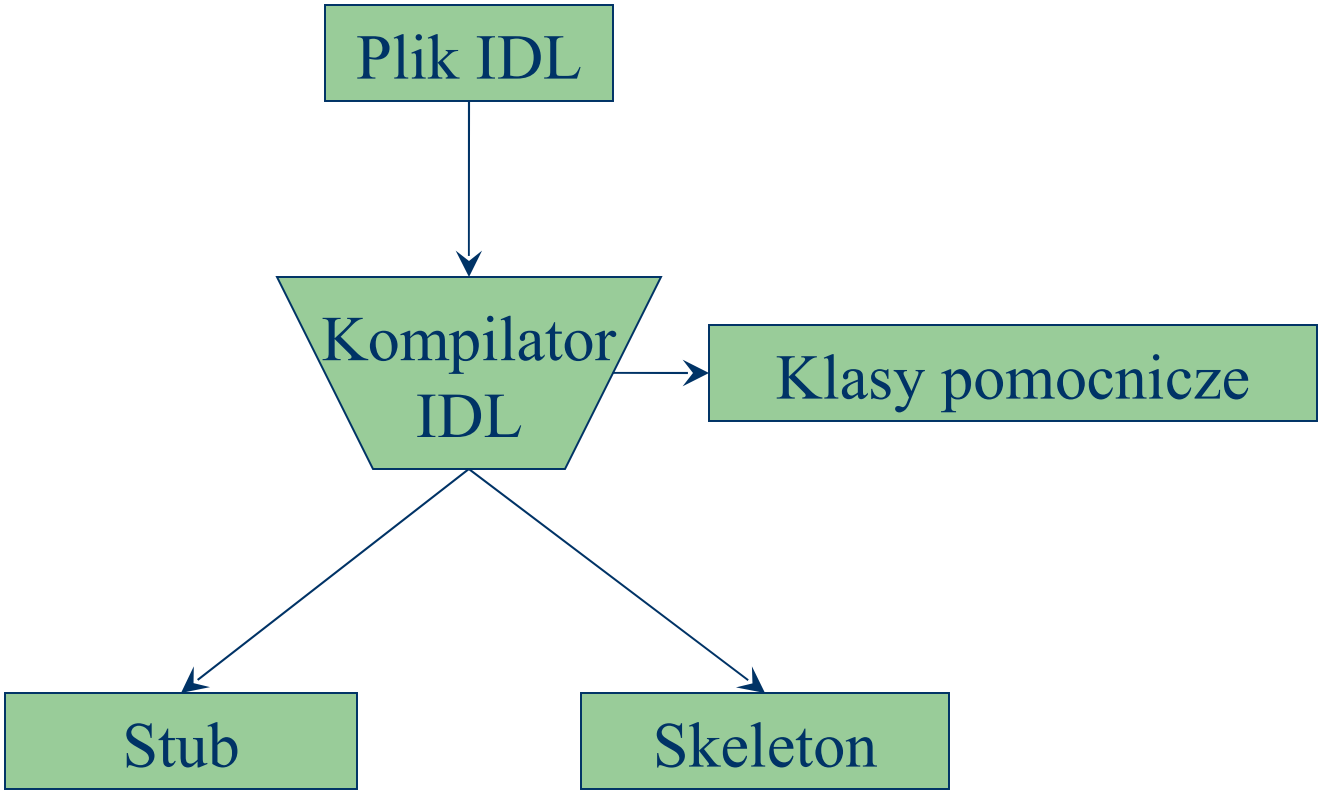
Interoperable Reference (IOR)

- Repository ID
  - standaryzowana
- Endpoint Info
  - standaryzowana
- Object Key
  - nie standaryzowana

Tworzenie aplikacji CORBA

- Określenie obiektów aplikacji i zdefiniowanie interfejsów w IDL
- Kompilacja definicji IDL do stubów i skeletonów
- Deklaracja i implementacja klas serwantów w C++, będących ucieleśnieniem obiektów
- Napisanie funkcji main() w serwerze
- Kompilacja i konsolidacja serwera
- Napisanie, kompilacja i konsolidacja klienta

Tworzenie aplikacji CORBA



Prosta aplikacja CORBA

```
struct TimeOfDay {
short   hour;  // 0 - 23
short   minute;// 0 - 59
short   second;// 0 - 59
};
interface Time {
    TimeOfDay get_gmt();
};
```

```
omniidl -bcxx time.idl
```

- time.hh
- timeSK.cc

```
//time.hh
class _impl_Time :
 public virtual omniServant
{
public:
 virtual ~_impl_Time();
 virtual TimeOfDay get_gmt() = 0;
};

class POA_Time :
 public virtual _impl_Time,
 public virtual
   PortableServer::ServantBase
{
public:
 virtual ~POA_Time();
 Time_ptr _this();
}
```

Serwer

```
//server.hh
#include "time.hh"

class Time_impl : public virtual
    POA_Time {
public:
     virtual TimeOfDay get_gmt()
    throw(CORBA::SystemException);
};
```

```
//server.cc
#include <time.h>
#include <iostream>
#include "server.hh"
using namespace std;

TimeOfDay Time_impl::get_gmt()
    throw(CORBA::SystemException)
{
    time_t time_now = time(0);
    struct tm * time_p =
    gmtime(&time_now);

    TimeOfDay tod;
    tod.hour = time_p->tm_hour;
    tod.minute = time_p->tm_min;
    tod.second = time_p->tm_sec;

    return tod;
}
```

Serwer (cd.)

```
//server.cc
int main(int argc, char * argv[]) {
try {
 // Initialize orb
 CORBA::ORB_var orb =
 CORBA::ORB_init(argc, argv);
// Get reference to Root POA.
 CORBA::Object_var obj = orb ->
 resolve_initial_references
  ("RootPOA");
 PortableServer::POA_var poa =
 PortableServer::POA::_narrow(obj);
 // Activate POA manager
 PortableServer::POAManager_var mgr
 = poa->the_POAManager();
 mgr->activate();
 // Create an object
 Time_impl time_servant;
 // Write its stringified
 //reference to stdout
 Time_var tm =
 time_servant._this();
 CORBA::String_var str =
 orb->object_to_string(tm);
 cout << str << endl;
 // Accept requests
 orb->run();
}
catch (const CORBA::Exception &)
{
 cerr << "Uncaught CORBA"
 "exception" << endl;
 return 1;
}
return 0; }
```

Kompilacja i uruchomienie

```
export OMNIHOME=/usr/local/omniORB-4.0.0

g++ -c -O2 -D__OMNIORB4__ -D_REENTRANT -I$OMNIHOME/include
-D__OSVERSION__=2 -D__linux__ -D__x86__ -o timeSK.o timeSK.cc

g++ -c -O2 -D__OMNIORB4__ -D_REENTRANT -I$OMNIHOME/include
-D__OSVERSION__=2 -D__linux__ -D__x86__ -o client.o client.cc

g++ -o server -L$OMNIHOME/lib timeSK.o client.o -lomniORB4 -lomnithread
-lpthread

./server
IOR:010000000d00000049444c3a54696d653a312e3000000000010000000000000064000
0000010102000e0000003231322e3234342e38362e343500cf800e000000fec16fa93d000
00346b000000000000000200000000000000080000001000000005454410100000001c00
000001000000010001000100000001000105090101000100000009010100
```

Klient

```
//client.cc
#include <iostream>
#include <iomanip>
#include "time.hh"
using namespace std;

int main(int argc, char * argv[]) {
try {
// Initialize orb
CORBA::ORB_var orb =
 CORBA::ORB_init(argc, argv);
// Check arguments
if (argc != 2) {cerr <<
"Usage: client IOR_string" << endl;
 throw 0; }
// Destringify argv[1]
CORBA::Object_var obj =
 orb->string_to_object(argv[1]);
 if (CORBA::is_nil(obj)) {
  cerr << "Nil Time reference" << endl;
  throw 0;}
```

```
// Narrow
Time_var tm = Time::_narrow(obj);
if (CORBA::is_nil(tm)) {
cerr << "Argument is not a Time "
"reference" << endl; throw 0; }
// Get time
TimeOfDay tod = tm->get_gmt();
cout << "Time in Greenwich is "
 << setw(2) << setfill('0') << tod.hour
 << ":" << setw(2) << setfill('0') <<
 tod.minute << ":" << setw(2) <<
 setfill('0') << tod.second << endl;
}
catch (const CORBA::Exception &) {
cerr << "Uncaught CORBA exception" <<
 endl;
return 1;
}
catch (...) {
 return 1;
}
 return 0;
}
```

Kompilacja i uruchomienie

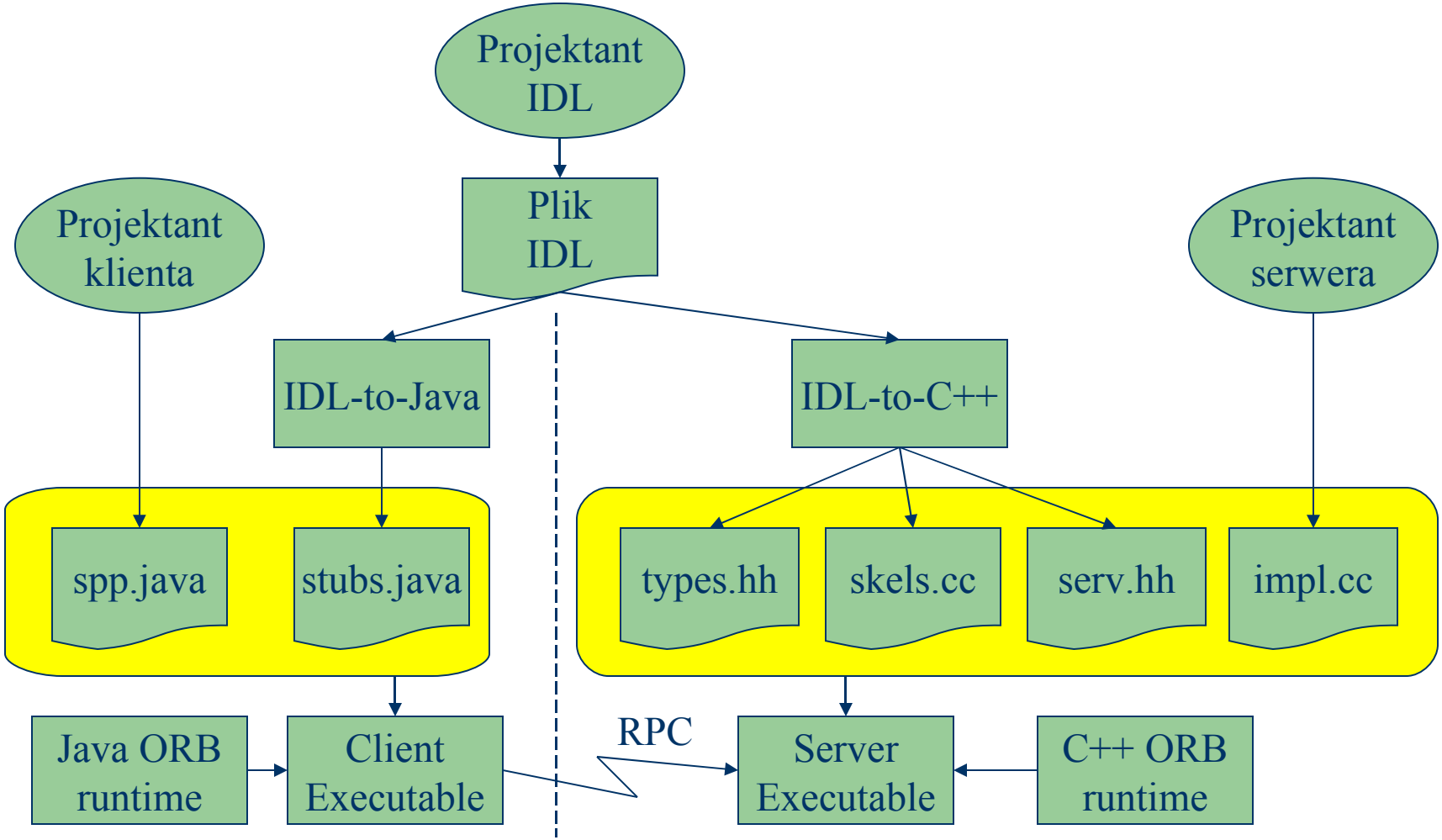
```
export OMNIHOME=/usr/local/omniORB-4.0.0

g++ -c -O2 -D__OMNIORB4__ -D_REENTRANT -I$OMNIHOME/include
-D__OSVERSION__=2 -D__linux__ -D__x86__ -o server.o server.cc

g++ -o server -L$OMNIHOME/lib timeSK.o server.o -lomniORB4 -lomnithread
-lpthread

$ ./server > ior &
[1] 4587
$ ./client `cat ior`
$ Time in Greenwich is 21:06:54
$ kill %1
[1]+ Terminated              ./server &
$
```

Klient i serwer w różnych językach



Pliki źródłowe IDL

- Rozszerzenie .idl
- Free-form
  - Tabulacje, odstępy, znaki końca linii nie mają znaczenia
- Preprocessing
  - Identyczny jak preprocesor C++
  - Np. #define, #include
- Identyfikatory muszą być zadeklarowane przed użyciem

Reguły leksykalne

- Komentarze
  - C-style /* */
  - C++-style //
- Słowa kluczowe
  - Pisane małymi literami
  - Wyjątki: Object, TRUE, FALSE
- Identyfikatory
  - Wymagane konsekwentne stosowanie dużych i małych liter
  - Nie może być dwóch identyfikatorów różniących się tylko wielkością liter

Podstawowe typy IDL

Typ	Zakres	Rozmiar
short	-2^{15} do $2^{15}-1$	≥16 bitów
long	-2^{31} do $2^{31}-1$	≥32 bitów
unsigned short	0 do $2^{16}-1$	≥16 bitów
unsigned long	0 do $2^{32}-1$	≥32 bitów
float	IEEE pojedyncza precyzja	≥32 bitów
double	IEEE podwójna precyzja	≥64 bitów
char	ISO Latin-1	≥8 bitów
string	ISO Latin-1 z wyj. ASCII NUL	zmienny
boolean	TRUE lub FALSE	nieokreślony
octet	0 do 255	≥8 bitów
any	Identyfikowany w czasie wykonania	zmienny

Typ `any`

- Typ **`any`**
  - typ uniwersalny
  - podobny do **`void*`** lub **`stdarg`**
  - bezpieczniejszy, bo samoopisujacy
  - utypowiony, trudniej o błędną interpretację

Typy definiowane przez użytkownika

- Typy nazwane - **`typedef`**
- Typy wyliczeniowe – **`enum`**
- Structury – **`struct`**
- Unie – **`union`**
- Tablice
- Sekwencje – **`sequence`**
- Typy rekursywne
- Stałe i literały

Typy nazwane

```
typedef short     YearType;
typedef short     TempType;
typedef TempType  TemperatureType;
```

- ostatnia deklaracja w złym stylu – niepotrzebna inna nazwa dla tego samego
- nie jest określone, czy **`TempType`** i **`TemperatureType`** to oddzielne typy,czy mogą być używane zamiennie – zależy od języka docelowego

Typy wyliczeniowe

```
enum Color { red, green, blue,
  black, mauve, orange };
```

- `typedef` nie jest potrzebny dla typów wyliczeniowych
- typ o długości min. 32 bitów
- nie są zdefiniowane konkretnych wartości dla poszczególnych etykiet
- zapewnia kolejność rosnącą od lewej do prawej

Typy wyliczeniowe (cd.)

- niedozwolone przypisanie konkretnych wartości poszczególnym etykietom – `enum Color { red = 0, green = 7 };` jest niepoprawne
- etykiety należą do otaczającej przestrzeni nazw, poniższy zapis jest niepoprawny:

```
enum InteriorColor {white,beige,grey};
enum ExteriorColor {yellow,beige,green};
```

- typ wyliczeniowy nie może być pusty

Struktury

- mają jedno lub więcej pól dowolnego typu, włączając typy zdefiniowane przez użytkownika

```
struct TimeOfDay {
   short hour;
   short minute;
   short second;
};
```

Struktury (cd.)

- definicja struktury to oddzielna przestrzeń nazw

```
struct Outer {
  struct FirstNested {
    long first;
    long second;
  } first;
  struct SecondNested {
    long first;
    long second;
  } second;
};
```

Struktury (cd.)

- prawie to samo, ale czytelniej:

```
struct FirstNested {
  long first;
  long second;
};
struct SecondNested {
  long first;
  long second;
};
struct Outer {
  FirstNested first;
  SecondNested second;
};
```

Unie

- posiadają pole dyskryminatora typu

```
union ColorCount switch (Color) {
  case red:
  case green:
  case blue:
    unsigned long num_in_stock;
  case black:
    float discount;
  default:
    string order_details;
};
```

Unie (cd.)

- jeżeli występuje opcja **`default`**, musi być możliwość jej wyboru

```
union U switch (boolean) {
   case FALSE:
      long count;
   case TRUE:
      string message;
   default: //illegal, cannot happen
      float cost;
};
```

Unie (cd.)

- przypadek szczególny: wartość opcjonalna

```
union AgeOpt switch (boolean) {
   case TRUE:
      unsigned short age;
};
```

Unie (cd.)

- symulacja przeciążenia funkcji (nie zalecana)

```
enum InfoKind { text, numeric, none };
union Info switch (InfoKind) {
  case text:
    string description;
  case numeric:
    long index;
};

interface Order {
  void set_details(in Info details);
};
```

Unie (cd.)

- zalecane rozwiązanie:

```
interface Order {
  void set_text_details(in string details);
  void set_details_index(in long details);
  void clear_details();
};
```

Tablice

- jedno- i wielowymiarowe

```
typedef Color ColorVector[10];
typedef string IDtable[10][20];
```

- **`typedef`** obowiązkowy:

```
Color ColorVector[10]; //invalid
```

- wszystkie wymiary obowiązkowe:

```
typedef string IDtable[][20]; //invalid
```

- przekazywanie indeksów między klientem i serwerem nieprzenośne: początkowy indeks zależny od języka programowania

Sekwencje

- wektory o zmiennej długości, z możliwością określenia maksymalnej długości:

```
typedef sequence<Color>     Colors;
typedef sequence<long, 100> Numbers;
```

- możliwe konstruowanie sekwencji sekwencji:

```
typedef sequence<Numbers> ListOfNumberVectors;
```

- typ elementu może być anonimowy (nie zalecane, ze względu na trudności z inicjalizacją):

```
typedef sequence<sequence<long,100> > LONV;
```

Sekwencje a tablice

- co kiedy zastosować:
  - lista zmiennej długości – sekwencja
  - lista o stałej długości – tablica
  - rekursywne struktury danych – sekwencja
  - macierz rzadka – sekwencja
- macierz rzadka:

```
typedef long Matrix[100][100];
interface MatrixProcessor {
  Matrix invert_matrix(in Matrix m);
};
```

Sekwencje a tablice (cd.)

```
struct NonZeroElement {
  unsigned short row;
  unsigned short col;
  long val;
};
typedef sequence<NonZeroElement> Matrix;

interface MatrixProcessor {
  Matrix invert_matrix(in Matrix m);
};
```

Typy rekursywne

- Rekursja poprzez struktury:

```
struct Node {
   long  value;
   sequence<Node> children;
};
```

Typy rekursywne (cd.)

- Rekursja poprzez unie:

```
enum OpType {OP_AND, OP_OR, OP_NOT, OP_BITAND, OP_BITOR,
             OP_BITXOR, OP_BITNOT};
enum NodeKind {LEAF_NODE, UNARY_NODE, BINARY_NODE};
union Node switch (NodeKind) {
case LEAF_NODE:
  long value;
case UNARY_NODE:
  struct UnaryOp {
    OpType             op;
    sequence<Node, 1> child;
  } u_op;
case BINARY_NODE:
  struct UnaryOp {
    OpType             op;
    sequence<Node, 2> children;
  } bin_op;
};
```

Typy rekursywne (cd.)

- Rekursja wyrażona wyłącznie za pomocą sekwencji:

```
//...
case BINARY_NODE:
  struct UnaryOp {
    OpType op;
    Node   children[2]; // Illegal recursion, not a sequence
  } bin_op;
//...
```

Deklaracje zapowiadające

- Pozwalają na uniknięcie typów anonimowych:

```
typedef sequence<Node> NodeSeq;
struct Node {
   long  value;
   NodeSeq children;
};
```

Deklaracje zapowiadające (cd.)

- Deklaracje zapowiadające mogą dotyczyć struktur i unii
- Dopóki brak definicji, typ jest *niekompletny*
- Typy niekompletne mogą występować tylko w deklaracjach sekwencji
- Sekwencja typów niekompletnych jest typem niekompletnym

Deklaracje zapowiadające (cd.)

```
struct Foo; // Introduces Foo type name,
// Foo is incomplete now
// ...
struct Foo {
// ...
}; // Foo is complete at this point
```

- W rekursywnych strukturach lub uniach niekompletne sekwencje składowe mogą odnosić się wyłącznie do właśnie definiowanych typów niekompletnych

```
struct Foo; // Forward declaration
typedef sequence<Foo> FooSeq;
struct Bar {
  long value;
  FooSeq chain; //Illegal, Foo is not an enclosing
                //struct or union
};
```

Wzajemnie rekursywne struktury

```
// Not legal IDL!
typedef something Adata;
typedef whatever  Bdata;
struct Astruct {
  Adata                  data;
  sequence<Bstruct, 1> nested; //illegal
};
struct Bstruct {
  Bdata                  data;
  sequence<Astruct, 1> nested;
};
```

Wzajemnie rekursywne struktury (cd.)

```
typedef something Adata;
typedef whatever  Bdata;
enum StructType { A_TYPE, B_TYPE };
union ABunion switch (StructType) {
case A_TYPE:
  struct Acontents {
    Adata                 data;
    sequence<ABunion,1> nested;
  } A_member;
 struct Bcontents {
    Bdata                 data;
    sequence<ABunion,1> nested;
  } B_member;
};
```

Stałe i literały

- tylko dla typów podstawowych, nie dla typów złożonych:

```
const float   PI = 3.1415926;
const char    NUL = '\0';
const string  LAST_WORDS = "My god, it's full of stars";
const octet   MSB_MASK = 0x80;

enum Color    { red, green, blue };
const Color   FAVOURITE_COLOR = green;

const boolean CONTRADICTION = FALSE; //bad idea
const long    ZERO = 0; //bad idea
```

Stałe i literały (cd.)

- inne nazwy (aliasy) typów podstawowych mogą być użyte do definiowania stałych:

```
typedef short TempType;
const TempType MAX_TEMP = 35;
```

- dostępne wszystkie typy literałów z C++

```
const string S1= ”Quote: \””;
const string S2= ”hello world”;
const string S3= ”hello” ” world”;
const string S4= ”\xA” ”B”; // two characters
const string<5> B5 = ”Hello”;
```

Wyrażenia stałe

- Operatory:
  - arytmetyczne: + - * / %
  - bitowe: | & ^ << >> ~
- Wyrażenia stało- lub zmiennoprzecinkowe, nie mieszane

```
const short MIN_TEMP = -10;
const short MAX_TEMP = 35;
const short AVG_TEMP = (MAX_TEMP + MIN_TEMP) / 2;

const float TWICE_PI = 3.14 * 2.0; // Can't use
                                   // 3.14 * 2 here
```

Wyrażenia stałe (cd.)

- Operacje bitowe
  - przesuwanie (unsigned) short o ponad 16 bitów lub (unsigned) long o ponad 32 bitów ma efekt niezdefiniowany
  - >> to przesunięcie logiczne

```
const long ALL_ONES = -1; //0xffffffff
const long LHW_MASK = ALL_ONES << 16; //0xffff0000
const long RHW_MASK = ALL_ONES >> 16; //0x0000ffff
```

Interfejsy i operacje

```
interface Haystack {
   exception NotFound {
        unsigned long num_straws_searched;
   };

   const unsigned long MAX_LENGHT = 10;      //Max len of a needle

   readonly attribute unsigned long num_straws;    //Stack size

   typedef long Needle;   //ID type for needles
   typedef string straw;          //ID type for straws

   void    add (in Straw s);                        //Grow Stack
   boolean remove(in Straw s);                      //Shrink Stack
   void  find(in Needle n) raises(NotFound) //Find Needle
};
```

Interfejsy i operacje (cd.)

- Reguły zasięgu takie same jak w C++

```
interface FeedShed {
   typedef sequence<Haystack> Stacklist;

   Stacklist feed_on_hand();//Return all stacks in shed

   void add(in Haystack s); //Add another haystack
   void eat(in Haystack s); //Cows need to be fed

   //look for a needle in all haystacks
   boolean find(in Haystack::Needle n)
                  raises(Haystack::Notfound);

   //Hide a needle (note that this is a oneway operation)
   oneway void hide(in Haystack s, in Haystack::Needle n);
};
```

- Nazwy interfejsów są typami
- Interfejsy można przekazywać jako parametry

Model komunikacyjny interfejsów

- Jak igły przedostają się z obory do stogów siana?
  - IDL tego nie wyjaśnia, możemy się tylko domyślać
  - może farmer?
- Operacje IDL i atrybuty to jedyne źródła definicji interakcji między obiektami
  - Jeżeli coś nie jest zapisane w IDL, dla CORBy nie istnieje
- Istnieje jakaś ukryta komunikacja między obiektami **`Haystack`** i **`FeedShed`**
- Może to utrudnić późniejsze modyfikacje systemu (rozdzielenie obiektów)

Atrybuty kierunkowe

- Użycie atrybutów kierunkowych
  - podnosi efektywność
  - określa odpowiedzialność za zarządzanie pamięcią
- Styl definicji
  - wartość zwracana jest wyróżniona

```
interface Primes {
  typedef unsigned long prime;
  prime  next_prime(in long n);
  void   next_prime2(in long n, out prime p);
  void next_prime3(inout long n);
};
```

Niedozwolone konstrukcje

- Przeciążanie
  - `prime  next_prime(in long n);`
  - `void   next_prime(in long n, out prime p);`
  - `void   next_prime(inout long n);`
- Typy anonimowe
  - `Sequence<long> get_longs();`
  - `void get_octets(out sequence<octet> s);`
- Operacje z atrybutem **`const`**
  - `SomeType read_value() const;`

Wyjątki definiowane przez użytkownika

```
exception Failed {};
exception RangeError {
  unsigned long supplied_val;
  unsigned long min_permitted_val;
  unsigned long max_permitted_val;
};
```

- Wyjątki nie mogą być zagnieżdżane i nie mogą być częścią składową innych typów

```
struct ErrorReport
{
  Object obj;
  RangeError exc; // Error
};
```

- Operacje używają słowa kluczowego **`raises`** aby wskazać zgłaszane wyjątki

```
interface Unreliable {
  void can_fail() raises(Failed);
  void can_also_fail() raises(Failed, RangeError);
};
```

- Brak dziedziczenia wyjątków

Projektowanie wyjątków

- Wyjątki tylko w wyjątkowych sytuacjach
- Zawierające użyteczną informację
- Zawierające dokładną informację
- Zawierające kompletną informację
- API wygodne dla użytkownika, nie dla implementatora
- Nie używać wartości zwracanych jako wskaźników błędów

Wyjątki systemowe

- Każda operacja może zwrócić wyjątek systemowy
- Nie podaje się tego jawnie

```
void op1() raises(BAD_PARAM);      //BAD
```

- Jest 39 wyjątków systemowych
- Wyjątki systemowe zawierają dwa pola
  - **`completion_status completed;`**
    - “Yes”, “No”, “Maybe”
  - **`unsigned long minor;`**
    - Dodatkowa informacja o błędzie

Operacje `oneway`

```
interface Events {
oneway void send(in EventData data);
};
```

- Zawodny przesył danych w jednym kierunku
  - typ zwracany **`void`**
  - brak parametrów **`out`** i **`inout`**
  - brak specyfikacji wyjątków
- Semantyka „best effort” i „at most once”
- Brak gwarancji nieblokowania i kolejności komunikatów
- CORBA Messaging zapewnia lepszą kontrolę nad tego rodzaju wywołaniami

Atrybuty

```
interface Thermostat {
  readonly attribute short temperature;
  attribute short nominal_temp;
};
```

- Równoważne:

```
interface Thermostat {
  short get_temperature();
  short get_nominal_temp();
  void set_nominal_temp(in short t);
};
```

- Brak możliwości zgłaszania wyjątków definiowanych przez użytkownika

Moduły

- Podobne do przestrzeni nazw w C++
  - Zapobiegają zaśmiecaniu globalnej przestrzeni nazw
  - Mogą być zagnieżdżane, zamykane i ponownie otwierane

```
module A {
    typedef short number;
    typedef string name;
};
module B {
    interface C {
        A::number age();
        A::name first_name();
        A::name last_name();
    };
};
module A { // modules can be re-opened
    interface C { // doesn't interfere with B::C
        number age(); // just need 'number', not 'A::number'
    };
};
```

Deklaracje zapowiadające

- Używane przy wzajemnie zależnych interfejsach:

```
interface Husband;
interface Wife {
  Husband get_spouse();
};
interface Husband {
  Wife get_spouse();
};
```

- Nie można tak deklarować interfejsu w innym module:

```
module Females {
  interface Males::Husband;
   //error
};
```

- Można to zrobić w następujący sposób:

```
module Females {
  interface Wife;
};
module Males {
  interface Husband {
    Females::Wife get_spouse();
  };
};
module Females {
  interface Wife {
    Males::Husband get_spouse();
  };
};
```

Dziedziczenie

```
interface Thermometer {
  typedef short TempType;
  readonly attribure TempType temperature;
};
interface Thermostat : Thermometer {
  void set_nominal_temp(in TempType t);
};
```

- Polimorfizm i reguły zasięgu jak w C++
- Niejawne dziedziczenie z interfejsu **`Object`**
- Dziedziczenie interfejsów, nie implementacji

```
interface Logger {
  long add(in Thermometer t, in unsigned short poll_interval);
  void remove(in long id);
};
```

Dziedziczenie – pusty interfejs

```
interface Vehicle {};
interface Car: Vehicle {
  void start();
  void stop();
};
interface Airplane: Vehicle {
  void take_off();
  void land();
};
interface Garage {
  void park(in Vehicle v);
  void make_ready(in Vehicle v);
};
```

Dziedziczenie – redefinicje

- Typy, stałe i wyjątki mogą być redefiniowane w klasie pochodnej

```
interface Thermometer {
  typedef long IDType;
  const UDType TID = 5;
  exception TempOutOfRange {};
};
interface Thermostat : Thermometer {
  typedef string IDType;
  const IDType TID = ”Thermostat”;
  exception TepmOutOfRange { long temp; };
};
```

Dziedziczenie – redefinicje (cd.)

- Atrybuty ani operacje nie mogą być redefiniowane w klasie pochodnej

```
interface Thermometer {
  attribute long temperature;
  void initialize();
};
interface Thermostat : Thermometer {
  attribute long temperature; //error
  void initialize(); //error
};
```

Dziedziczenie – redefinicje (cd.)

- Przeciążenie atrybutów i operacji jest również niedozwolone

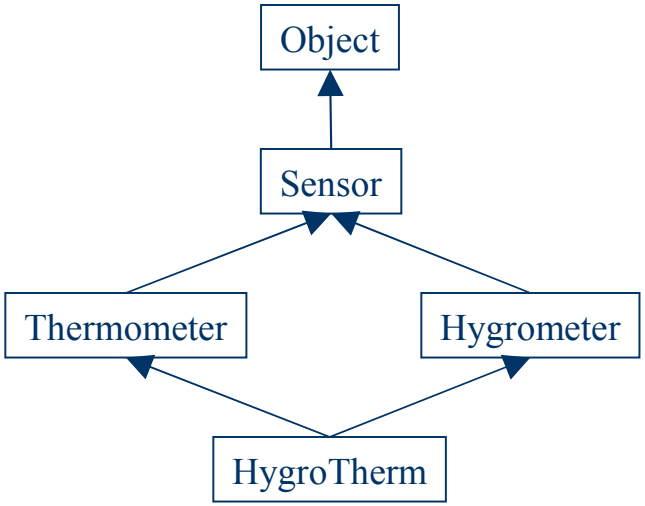
```
interface Thermometer {
  attribute string my_id;;
  string get_id();
  void set_id(in string s);
};
interface Thermostat : Thermometer {
  attribute double my_id; //error
  double get_id(); //error
  void set_id(in double d); //error
};
```

Dziedziczenie wielobazowe

```
interface Thermometer { /* ... */};
interface Hygrometer { /* ... */};
interface HygroTherm: Thermometer, Hygrometer { /* ... */};
```

- Można dziedziczyć więcej niż raz z jednego interfejsu bazowego:

```
interface Sensor { /* ... */};
interface Thermometer : Sensor{ /* ... */};
interface Hygrometer : Sensor{ /* ... */};
interface HygroTherm: Thermometer, Hygrometer { /* ... */};
```



Ograniczenia dziedziczenia wielobazowego

- Operacje i atrybuty mogą być odziedziczone tylko z jednego interfejsu podstawowego

```
interface Thermometer {
  attribute string model;
  void initialize();
};
interface Hygrometer {
  attribute string model;
  void initialize();
};
interface HygroTherm : Thermometer, Hygrometer {
 // ... ambiguity - which model and initialize()
};
```

Ograniczenia dziedziczenia wielobazowego (cd.)

- Sprzeczne definicje typów

```
interface Thermometer {
typedef string<16> ModelType;
};
interface Hygrometer {
  typedef string<32> ModelType;
};
interface HygroTherm : Thermometer, Hygrometer {
attribute ModelType model; //ambiguity – 16 or 32 chars?
};
```

- Jawna specyfikacja zasięgu

```
interface HygroTherm : Thermometer, Hygrometer {
attribute Thermometer::ModelType model; // OK, 16 chars
};
```

Nazwy i zasięg

- Identyfikatory unikatowe wewnątrz ich zasięgu
  - modułu, interfejsu, struktury, unii, wyjątku, definicji operacji
- W identyfikatorach rozróżniane wielkie i małe litery, ale nie mogą się powtarzać identyfikatory różniące się tylko wielkością liter

```
module CCS {
  typedef short TempType;
  typedef double temptype; //Error
};
```

- Nazwa w zasięgu nie może być taka sama, jak nazwa otaczającego zasięgu - poniższe konstrukcje są nielegalne:

```
module A {
    module A {
        /* . . . */
    };
};
```

```
interface SomeName {
  typedef long SomeName;
};
```

- Wyszukiwanie nazw przez przeszukiwanie kolejnych otaczających zasięgów – jak w C++

IDL Repository ID

```
module CCS {
  typedef short TempType;
  interface Thermometer {
    readonly attribute TempType temperature;
  };
  interface Thermostat : Thermometer {
    void set_nominal_temp(in TempType t);
  };
};

IDL:CCS:1.0
IDL:CCS/TempType:1.0
IDL:CCS/Thermometer:1.0
IDL:CCS/Thermometer/temperature:1.0
IDL:CCS/Thermostat:1.0
IDL:CCS/Thermostat/set_nominal_temp:1.0
```

#pragma prefix

```
#pragma prefix "acme.com"
module CCS {
  // ...
};

IDL:acme.com:/CCS:1.0
IDL:acme.com:/CCS/Temptype:1.0
IDL:acme.com:/CCS/Thermometer:1.0
IDL:acme.com:/CCS/Thermometer/temperature:1.0
IDL:acme.com:/CCS/Thermostat:1.0
IDL:acme.com:/CCS/Thermostat/set_nominal_temp:1.0
```

- Prefiks zmniejsza ryzyko kolizji nazw
- Prefiks nie wpływa na generowany kod

Ostatnie uzupełnienia IDL

- Szerokie znaki i łańcuchy (unicode)
  - `const wchar C =L'X';`
  - `const wstring GREETING = L'Hello'`
- 64-bitowe liczby całkowite
  - `long long`
  - `unsigned long long`
- Rozszerzony typ zmiennoprzecinkowy
  - long double
  - mantysa ≥64 bitów, cecha ≥15 bitów
- Stałoprzecinkowe liczby dziesiętne
  - `typedef fixed<9,2> AssetValue;`
  - `const fixed val1=3.14D;`