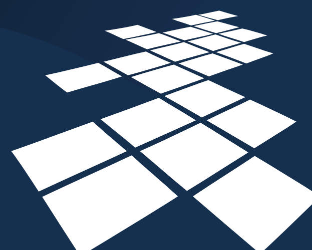
CORBA

wykład prowadzi
Mikołaj Morzy



UCZELNIA
ONLINE

Zaawansowane aplikacje internetowe

Plan wykładu

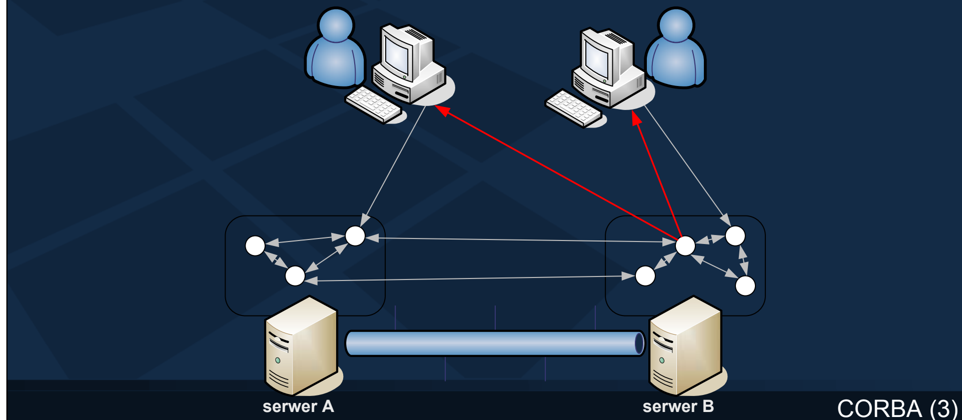
- Wprowadzenie
- Architektura CORBA
- IDL – język definicji interfejsów
- ORB – Object Request Broker
- Usługi i POA
- Aplikacje CORBA
  - Tworzenie serwera
  - Tworzenie klienta

CORBA (2)

Celem wykładu jest przedstawienie architektury CORBA, standardu tworzenia rozproszonych aplikacji obiektowych. Wykład ma na celu wprowadzenie słuchacza w zagadnienia związane z tworzeniem rozproszonych, niezależnych od platformy aplikacji internetowych oraz przedstawienie niezbędnych podstaw architektury CORBA. Pełna prezentacja możliwości architektury CORBA zdecydowanie wykracza poza ramy tego wykładu. W trakcie wykładu zostaną przedstawione podstawowe komponenty architektury: język definicji interfejsów IDL, broker ORB, sposób zarządzania obiektami, oraz dodatkowe usługi i adaptery. Wykład zakończy krótka prezentacja tworzenia aplikacji CORBA w języku Java.

Aplikacje rozproszone

- Rozproszenie danych
- Rozproszenie obliczeń
- Rozproszenie użytkowników



Wiele aplikacji jest w sposób naturalny rozproszonych. Rozproszenie aplikacji może być spowodowane przez następujące czynniki:

•dane, na których działa aplikacja, mogą być przechowywane na wielu komputerach, dostęp do danych może się odbywać tylko zdalnie, systemy przechowujące dane mogą być heterogeniczne,

•obliczenia wykonywane przez aplikację mogą być rozproszone w celu lepszego wykorzystania mocy obliczeniowej dostępnych komputerów lub w celu wykorzystania jakichś specyficznych cech wybranych komputerów,

•użytkownicy aplikacji mogą być fizycznie rozproszeni.

Rysunek zamieszczony na slajdzie ilustruje scenariusz typowy dla aplikacji rozproszonej. Użytkownicy wykonują swoje fragmenty aplikacji rozproszonej wykorzystując do tego celu lokalne komputery. Fragmenty aplikacji komunikują się ze sobą za pomocą obiektów współdzielonych, wykonujących się najczęściej na wielu serwerach. Obiekty wykonujące się w ramach jednego serwera mogą być ze sobą mocno powiązane, ale mogą także wiązać się z obiektami wykonującymi się zdalnie na innym serwerze.

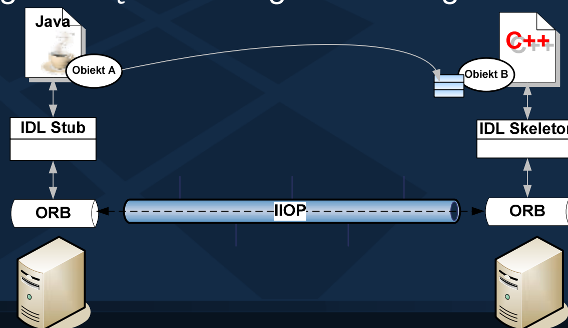
Cechy aplikacji rozproszonych

	System lokalny	System rozproszony
Komunikacja	szybka	wolna
Awarie	obiekty ulegają awarii wszystkie na raz	obiekty ulegają awarii niezależnie, możliwy podział sieci
Współbieżny dostęp	wielowątkowo	tak
Bezpieczeństwo	wysokie	niskie

W przypadku aplikacji rozproszonych napotykamy wiele problemów, które nie występują w przypadku aplikacji lokalnych, gdzie logika aplikacji i obiekty aplikacji wykonują się w tym samym lokalnym środowisku. Podstawowe i najważniejsze, z punktu tworzenia aplikacji, różnice między systemami lokalnymi i rozproszonymi są przedstawione w powyższej tabeli. Komunikacja między obiektami lokalnymi, wykonującymi się w ramach jednego procesu, jest o rzędy wielkości szybsza niż komunikacja między obiektami wykonującymi się na różnych komputerach. Jeśli dwa obiekty aplikacji są ze sobą ściśle związane, powinny się wykonywać lokalnie. Awarie systemów lokalnych mają najczęściej charakter atomowy, przerwanie procesu aplikacji pociąga za sobą awarię wszystkich obiektów lokalnych. W przypadku systemu rozproszonego każdy obiekt może ulec awarii niezależnie, stąd aplikacja musi obsługiwać też sytuację, gdy dany obiekt nie jest dostępny. Dodatkowo, komunikacja między obiektami może się nie powieść z powodu przerwania łączności sieciowej. Logika aplikacji rozproszonej powinna przewidywać i poprawnie obsługiwać takie sytuacje. Dostęp do obiektów lokalnych jest najczęściej sekwencyjny. Współbieżność uzyskuje się najczęściej poprzez wielowątkowe wykonanie aplikacji. Współbieżny dostęp do obiektów narzuca konieczność synchronizacji dostępu do obiektów. W systemie rozproszonym każdy obiekt może być odczytywany i modyfikowany współbieżnie przez wiele procesów, stąd zawsze konieczne jest uwzględnienie mechanizmów synchronizacji w architekturze aplikacji. Wreszcie lokalne wykonanie obiektów nie pociąga za sobą dodatkowych problemów związanych z bezpieczeństwem. W systemie rozproszonym identyfikacja i uwierzytelnianie obiektów, usług i metod są niezbędne w praktycznie każdej aplikacji.

Co to jest CORBA

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  - architektura obiektowych systemów rozproszonych umożliwiająca współpracę między heterogenicznymi, rozproszonymi kolekcjami obiektów
- Paradygmat: żądanie usługi od zdalnego obiektu



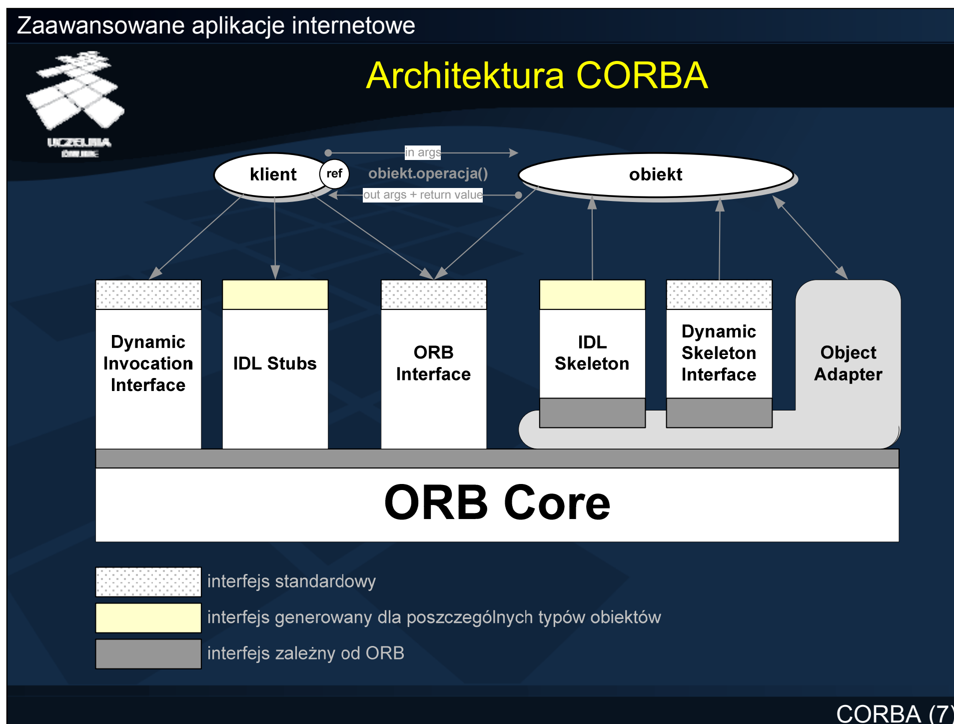
CORBA (ang. Common Object Request Broker) to standardowa architektura obiektowych systemów rozproszonych, która umożliwia współpracę i łączenie między heterogenicznymi i rozproszonymi kolekcjami obiektów. Cała architektura CORBA jest zbudowana zgodnie z paradygmatem żądania wykonania usługi przez zdalny obiekt. Rysunek na slajdzie przedstawia schematycznie ideę architektury CORBA. Obiekt aplikacji klienckiej, napisanej w języku Java, zgłasza żądanie wykonania usługi (czyli wywołania metody) przez zdalny obiekt w aplikacji stanowiącej serwer (czyli dostawcę usług). Aplikacja po stronie serwera jest napisana w języku C++. Oba programy wykorzystują specjalny interfejs IDL do stworzenia obiektów pomocniczych, które tłumaczą obiekty aplikacyjne na postać ujednoliconą. Obiekty ujednolicone to obiekty CORBA, które mogą być swobodnie wymieniane między poszczególnymi komponentami architektury. Komunikacją między elementami aplikacji rozproszonej zajmują się brokerzy obiektów, ORB (ang. Object Request Broker), komunikujący się ze sobą poprzez protokół IIOP.

OMG i historia standardu

- Object Management Group
  - Konsorcjum 800 firm informatycznych
  - Standardy: OMA, CORBA, UML, XMI, MDA
- Historia standardu CORBA
  - CORBA 1.0, grudzień 1990
  - CORBA 1.1, 1.2 (wprowadzenie IDL), grudzień 1993
  - CORBA 2.0 (wprowadzenie IIOP), grudzień 1994
  - CORBA 2.1, 2.2, …, 2.6, grudzień 2001
  - CORBA 3.0, lipiec 2003

Standard CORBA został opracowany przez OMG (ang. Object Management Group), konsorcjum zrzeszające około 800 firm z branży informatycznej. Zostało ono powołane w 1989 roku przez wiodące na rynku firmy i od tego czasu zajmuje się standaryzacją wielu produktów i technologii związanych z przetwarzaniem rozproszonym. Podstawowym dziełem OMG jest standard OMA (ang. Object Management Architecture), który stanowi połączenie wielu różnych technologii internetowych. Inne standardy opracowane i pielęgnowane przez OMG to CORBA, UML (ang. Unified Modeling Language), XMI (ang. XML Metadata Interchange) oraz MDA (ang. Model Driven Architecture). CORBA jest zatem wynikiem szerokiego kompromisu między wieloma producentami, co niestety często rzutuje na przenaszalność i uniwersalność niektórych modułów.

Jak na produkt informatyczny, CORBA ma za sobą wieloletnią tradycję. Pierwsza wersja standardu CORBA pojawiła się w roku 1990. Wprowadzenie języka IDL (wersja 1.1, 1991 rok) i ustabilizowanie wielu aspektów architektury w wersji 1.2 (1993) zdobyło architekturze CORBA wielu zwolenników. Kamieniem milowym rozwoju standardu była wersja 2.0, wprowadzająca protokół IIOP i opublikowana w roku 1994. Kolejne dziesięć lat to następne wersje standardu, aż do wersji 2.6, i liczne poprawki i rozszerzenia, przede wszystkim o wiązania dla coraz to nowych języków programowania. Od lipca 2003 obowiązuje wersja 3.0 standardu.



Slajd przedstawia schemat architektury CORBA. Klient wykorzystuje kompilator IDL, który na podstawie definicji interfejsu generuje kod pieńka (ang. stub). Pieniek jest obiektem automatycznie generowanym po stronie klienta i odpowiedzialnym za uszykowanie (ang. marshal) zlecenia. Szeregowanie zlecenia polega na zamianę wywołania metody lokalnej, którego użył klient, na postać dogodną do transmisji sieciowej i wywołanie metody na właściwym obiekcie zdalnym. Po stronie serwera znajduje się automatycznie wygenerowany szkielet (ang. skeleton), który przeprowadza operację odwrotną do pieńka. Szkielet transformuje żądanie klienta (ang. unmarshal) do postaci zrozumiałej dla obiektu i wykonuje żądaną operację. Wynik operacji, wraz z wartościami parametrów wyjściowych, jest przesyłany z powrotem do klienta. Zarówno pieniek, jak i szkielet, są automatycznie generowane przez kompilator IDL na podstawie plików IDL zawierających niezależny od języka implementacji opis usługi. Komunikacja między klientem i obiektem odbywa się poprzez brokera obiektów. Interfejs dostępu do warstwy ORB jest standardowy dla wszystkich implementacji architektury CORBA. Po stronie serwera występuje także adapter obiektów (ang. Object Adapter). Jest to obiekt odpowiedzialny przede wszystkim za adaptowanie interfejsu obiektu świadczącego usługę (ang. servant) do interfejsu klienta składającego żądanie.

Większość wywołań metod w architekturze CORBA ma charakter statyczny. Kompilator IDL generuje kod pieńka i szkieletu na podstawie statycznej, znanej w momencie kompilacji specyfikacji. Czasem jednak zachodzi konieczność wywołania dynamicznego metody. CORBA oferuje taką możliwość za pomocą mechanizmu dynamicznej inwokacji interfejsu (po stronie klienta) i dynamicznego budowania szkieletu interfejsu (po stronie serwera). W przypadku stosowania tej metody zamiast statycznych pieńków i szkieletów klient komunikujący się z obiektem zdalnym wykorzystuje fragmenty dynamicznie wygenerowanego kodu, które funkcjonują tak samo jak pieńki.

Zaawansowane aplikacje internetowe

Cechy architektury CORBA

- Model pojęciowy OMA
- Język IDL (abstrakcyjny język obiektowy)
- Wiązania do wielu języków programowania
- Automatyczne generowanie kodu pieńków i szkieletów
- Transparentność lokalizacji obiektów i usług
- Wspólne usługi i udogodnienia CORBA
- Niezależność od sprzedawców

CORBA (8)

Architektura CORBA stanowi implementację pojęciowego modelu architektury zarządzania obiektami (ang. OMA, Object Management Architecture), który zostanie opisany w dalszej części wykładu. CORBA dostarcza języka definicji interfejsów (ang. IDL, Interface Definition Language), który może być postrzegany jako abstrakcyjny język obiektowy. Dzięki językowi IDL obiekty zaimplementowane na różnych platformach i przy użyciu różnych języków programowania mogą komunikować się ze sobą swobodnie. Aktualnie CORBA oferuje wiązania do większości popularnych języków programowania, takich jak C, C++, Ada, Lisp, Java, SmallTalk, Python, Perl, Visual Basic i Tcl. Bardzo ważną cechą architektury, wynikającą wprost z wykorzystywania abstrakcyjnego języka IDL, jest możliwość automatycznej generacji kodu obiektów fasadowych: pieńków i szkieletów. Przyspiesza i ułatwia to tworzenie oraz testowanie aplikacji. CORBA oferuje pełną transparentność lokalizacji obiektów i usług. Klient zgłaszający żądanie wykonania operacji na obiekcie wykorzystuje do tego celu pieniek i wywołuje operację tak, jak gdyby była to metoda lokalna. Klient nie musi obsługiwać żadnych aspektów komunikacji sieciowej i zdalnego wołania metod. Z punktu widzenia klienta każdy wykorzystywany obiekt znajduje się (oczywiście wirtualnie) w przestrzeni adresowej klienta. Dodatkowo, CORBA oferuje także całe bogactwo tzw. wspólnych usług obiektowych (ang. CORBAServices), obejmujących zarządzanie nazwami, cyklem życia, trwałością, współbieżnością, transakcjami, dostępem zewnętrznym, zapytaniami, związkami, własnościami, zdarzeniami, i wieloma innymi aspektami przetwarzania obiektowego. Obok wspólnych usług CORBA oferuje też tzw. wspólne udogodnienia (ang. CORBAFacilities), przeznaczone dla konkretnych dziedzin zastosowań: rachunkowości, finansów, medycyny, i wielu innych. Wreszcie, teoretycznie CORBA oferuje niezależność od implementacji sprzedawców. Poszczególni brokerzy ORB powinni móc swobodnie się ze sobą porozumiewać. Praktyka jednak weryfikuje czasem te nadmiernie optymistyczne założenia.

Zaawansowane aplikacje internetowe

CORBA a RPC, Web Services

- Podobieństwa
  - Zdalne wywoływanie metod
  - Niezależność od języka implementacji
- Różnice
  - Abstrakcyjny język obiektowy (IDL)
  - Pełna definicja protokołów i zachowania
  - Wiązania do wielu języków programowania
  - Udogodnienia do zarządzania obiektami
  - Standardowe usługi
  - Duży narzut na zasoby (szybkość, pamięć)

CORBA (9)

Często spotykanym nieporozumieniem jest traktowanie architektury CORBA jako odpowiednika protokołu zdalnego wołania procedur RPC (ang. Remote Procedure Call) lub technologii usług sieciowych (ang. WS, Web Services). Podobieństwa sprowadzają się do tego, że wszystkie te technologie służą do zdalnego wywoływania metod i są niezależne od języka implementacji. Jednak bliższe przyjrzenie się porównywanym technologiom ujawnia wiele różnic. CORBA oferuje pełny abstrakcyjny język obiektowy IDL, którego brak w RPC i WS. Specyfikacja OMG w pełni i bardzo szczegółowo opisuje wszystkie elementy architektury CORBA (protokoły, mechanizmy wewnętrzne, zachowanie), dzięki czemu implementacje CORBY są dużo bardziej spójne niż w implementacje protokołów RPC. CORBA oferuje wiązania do większości języków programowania. CORBA jest też znacznie obszerniejszą technologią niż RPC lub WS. Obejmuje nie tylko zdalne wołanie metod, ale oferuje kompleksowe mechanizmy zarządzania obiektami, ich życiem, trwałością, interakcjami, itp. Wiele typowych, często powtarzających się problemów może być łatwo rozwiązanych za pomocą generycznych i standardowych usług oferowanych przez architekturę CORBA. Trzeba także wspomnieć, że CORBA nakłada na klientów i usługodawców dużo większy narzut na wykorzystywane zasoby (szybkość działania, pamięć operacyjna) niż protokoły RPC lub usługi sieciowe.

Zaawansowane aplikacje internetowe

Object Management Architecture

- Obiekt
  - model encji lub pojęcia
  - niezmienna tożsamość obiektów
- Operacja
  - zachowanie deklarowane przez sygnaturę
- Interfejs
  - nazwany zbiór operacji
  - może być dziedziczony i zamienialny
- Typ prosty

CORBA (10)

CORBA jest przykładem implementacji architektury zarządzania obiektami (ang. OMA, Object Management Architecture). Na architekturę zarządzania obiektami składają się model obiektów (opis obiektów w systemie rozproszonym) oraz model referencji (opis interakcji między rozproszonymi obiektami). Podstawowe pojęcia związane z architekturą OMA to obiekt, operacja, interfejs i typ prosty.

Obiektem nazywamy model encji lub pojęcia występującego w modelowanym świecie. Każdy obiekt posiada własną, unikalną i niezmienną tożsamość związaną z obiektem przez cały cykl życia obiektu. Operacja to zachowanie obiektu opisane za pomocą sygnatury. Na sygnaturę składają się: nazwa operacji, zbiór parametrów, tryb przekazywania parametrów (wejściowy, wyjściowy, wejściowo-wyjściowy) oraz typ zwrotny operacji. Nazwany zbiór operacji tworzy interfejs. Interfejs może być dziedziczony oraz zamienialny, jeśli alternatywny interfejs oferuje wymagane operacje i nie powoduje błędu interakcji z klientem inicjującym operację. Dziedziczenie interfejsów jest wielobazowe. Wreszcie typy proste to typy bazowe, nie będące typami obiektowymi. Konkretny zbiór obsługiwanych typów prostych nie jest definiowany w OMA, lecz delegowany do implementacji, np. CORBA.

IDL – wprowadzenie

- Cechy IDL
  - Deklaratywny język definicji interfejsów
  - Niezależność od języka implementacji
  - Składnia podobna do C++
  - Brak odniesień do implementacji operacji
- Pojęcia
  - **Moduł**: zbiór interfejsów (pakiet, przestrzeń nazw)
  - **Interfejs**: specyfikacja cech i operacji obiektu (klasa)
  - **Atrybut**: cecha obiektu (składowa)
  - **Operacja**: opis zachowania się obiektu
  - **Wyjątek**: opis nietypowej sytuacji

Język definicji interfejsów (ang. Interface Definition Language) to deklaratywny język specyfikacji obiektów kooperujących ze sobą za pomocą architektury CORBA. IDL jest całkowicie niezależny od języka implementacji konkretnego obiektu i nie zawiera żadnych odniesień do implementacji operacji zdefiniowanych w obiekcie. Składnia IDL jest wzorowana na języku C++. IDL można postrzegać jako lingua franca definicji klas i pakietów w heterogenicznych środowiskach przetwarzania obiektowego. IDL służy przede wszystkim do automatycznego generowania pieńków (po stronie klienta) i szkieletów (po stronie usługodawcy).

Podstawowe pojęcia występujące w IDL są następujące:

•Moduł: zbiór interfejsów zgrupowanych pod jedną nazwą i powiązanych logicznie lub biznesowo. Odpowiada pakietom (Java) lub przestrzeniom nazw (C++). Wyznacza zakres unikalności nazw obiektów, klas, wyjątków, itp.

•Interfejs: zbiór cech i operacji dostępnych w odniesieniu do danego obiektu. W terminologii obiektowych języków programowania odpowiada pojęciu klasy lub interfejsu. Interfejs może być zdefiniowany z wykorzystaniem dziedziczenia lub dziedziczenia wielobazowego.

•Atrybut: cecha zdefiniowana w interfejsie, odpowiada składowej klasy lub składowej interfejsu. Każdy atrybut posiada automatycznie wygenerowane metody dostępowe getter i setter. Atrybut może być tylko do odczytu.

•Operacja: opis zachowania obiektu zadeklarowany w postaci sygnatury, dla każdego parametru w sygnaturze można podać tryb przekazania parametru. Możliwe jest też zdefiniowanie trybu wykonania operacji. Domyślnie operacje wykonują się synchronicznie, ale mogą się też wykonywać asynchronicznie z potwierdzeniem lub asynchronicznie bez potwierdzenia (tryb "fire and forget")

•Wyjątek: służą do sygnalizacji i obsługi nietypowych sytuacji, dzielą się na wyjątki systemowe (standardowe dla architektury CORBA) oraz wyjątki użytkownika (definiowane w IDL, mogą zawierać własne atrybuty)

Zaawansowane aplikacje internetowe

IDL – przykład

messageBox.idl

```
module MessageModule {                                   (1)
   typedef sequence<string> MessageSeq;
(2) interface MessageBox {
        exception boxFull {};                            (3)
   (4)  attribute string reply;
        string leaveMessage(in string msg) raises (boxFull);   (5)
        MessageSeq getMessages();
   };
};
```

CORBA (12)

Przykład przedstawiony na tym i na kolejnych slajdach pokazuje sposób implementacji prostej aplikacji modelującej zachowanie się telefonicznej automatycznej sekretarki. Sekretarka posiada zapisaną domyślną odpowiedź udzielaną klientom. Klient może połączyć się z sekretarką i nagrać nową wiadomość lub odsłuchać wszystkie nagrane wiadomości. Sekretarka ma określoną pojemność komunikatów, próba pozostawienia komunikatu prowadząca do przepełnienia skrzynki powoduje zgłoszenie wyjątku. Slajd przedstawia definicję interfejsu MessageBox implementującego funkcjonalność automatycznej sekretarki. Plik messageBox.idl definiuje (1) moduł MessageModule zawierający wykorzystywane interfejsy i wyjątki. Głównym elementem umieszczonym w module jest interfejs MessageBox (2). W skład interfejsu MessageBox wchodzi wyjątek o nazwie boxFull (3). Interfejs MessageBox posiada atrybut reply (4) reprezentującą domyślną odpowiedź. Interfejs posiada też dwie operacje (5): pozostawienie nowej wiadomości (leaveMessage(in string) raises (boxFull)) oraz odsłuchanie wiadomości (getMessages()). Na podstawie tak zdefiniowanego pliku kompilator IDL jest w stanie wygenerować pieniek (obiekt klienta żądającego wykonania usług z interfejsu) oraz szkielet (obiekt reprezentujący usługodawcę, czyli automatyczną sekretarkę).

IDL – typy proste i złożone

- Wbudowane typy proste
  - [unsigned] `short`, `long`, `long long`
  - `float`, `double`, `long double`
  - `char`, `wchar`, `string`, `wstring`, `string<n>`
  - `boolean`, `octet`, `enum`, `any`
- Wbudowane typy złożone
  - `struct`, `union`, `sequence`, `array`
- Stałe
  - `const`
- Redefinicja typu: `typedef`

Do definiowania typów atrybutów można wykorzystywać w IDL typy proste i złożone. Wbudowane typy proste obejmują:

•Typy całkowitoliczbowe ze znakiem i bez znaku: [unsigned] short, long, long long

•Typy zmiennoprzecinkowe: float, double, long double

•Typy znakowe: char (ISO Latin-1), wchar (Unicode), string (ISO Latin-1), wstring (Unicode), string<n> (łańcuch o długości co najwyżej n znaków)

•Typ logiczny: boolean

•Typ binarny: octet (8 bitów nieinterpretowanych danych)

•Typ wyliczeniowy: enum

•Typ dowolny: any (dowolny typ CORBA)

Wbudowane typy złożone obejmują:

•Typy rekordowe: struct, union (oba wzorowane na C++)

•Typy wzorcowe: sequence (liniowy kontener o ograniczonej lub nieograniczonej pojemności), array (tablica)

Stałe wprowadza się za pomocą słowa kluczowego const. Istniejący typ może być redefiniowany lub zawężony za pomocą słowa kluczowego typedef. Na poprzednim slajdzie znalazł się wpis " typedef sequence<string> MessageSeq", jest to definicja typu MessageSeq jako nieograniczonej kolekcji liniowej łańcuchów znaków.

IDL – odwzorowania językowe

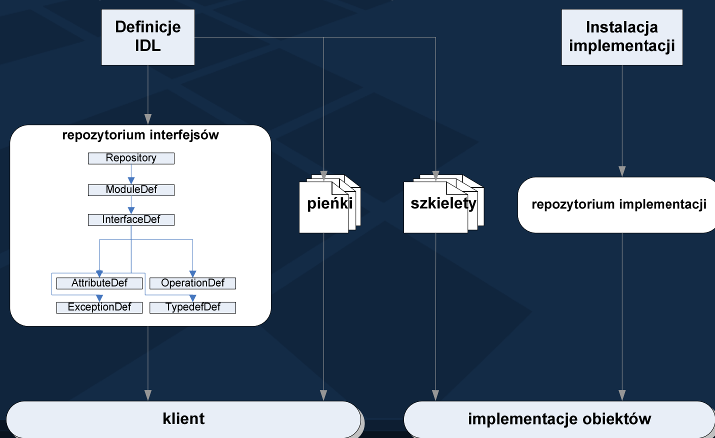
OMG IDL	Java	C++
short, long, double	short, int, long	short, long, double
any	org.omg.CORBA.Any	any class
string, wstring	java.lang.String	char*, wchar_t*
sequence	array	class
typedef	helper class	typedef
interface	class	class
exception	class	class
module	package	namespace

Slajd przedstawia odwzorowanie wybranych elementów IDL na konstrukty języków Java i C++. Przykładowo, atrybut zadeklarowany jako string zostanie zaimplementowany w Javie jako java.lang.String, a w C++ jako char*. Ponieważ IDL jest wzorowany na C++, widać dużą zbieżność nazw konstruktów. W przypadku Javy niektóre translacje stają się kłopotliwe. Na przykład konstrukt IDL typedef jest zaczerpnięty wprost z C++, język Java nie posiada takiego konstruktu i w odwzorowaniu typ zdefiniowany jako typedef w IDL zostanie w Javie zaimplementowany w postaci klasy pomocniczej.

Zaawansowane aplikacje internetowe

Repozytoria

- Repozytorium interfejsów
- Repozytorium implementacji



CORBA (15)

W przypadku statycznego wykorzystywania specyfikacji IDL dostęp do repozytorium nie jest konieczny, ponieważ zarówno klient, jak i usługodawca, posiadają pełnię wiedzy o obiektach i ich usługach przechowywaną w pieńkach i szkieletach. Jednak nie zawsze dostęp statyczny jest możliwy. W przypadku dynamicznych wywołań usług konieczna staje się możliwość dostępu do typów zdefiniowanych w IDL w trybie runtime. Taka sytuacja występuje np. w środowiskach, w których często następują zmiany w strukturze przetwarzanych obiektów. Dynamiczny dostęp do interfejsów, operacji i atrybutów zdefiniowanych w IDL odbywa się poprzez repozytorium interfejsów. Jest to hierarchiczna struktura zawierająca drzewo powiązanych ze sobą obiektów reprezentujących moduły, interfejsy, atrybuty, wyjątki, operacje, i stałe. Po stronie usługodawcy występuje analogiczna struktura, zwana repozytorium implementacji. Przechowywane są w nim informacje o implementacjach wszystkich dostępnych obiektów, alokacji zasobów, kontroli administracyjnej, prawach dostępu, itp. Repozytorium implementacji jest podstawowym narzędziem, za pomocą którego broker ORB potrafi lokalizować i uaktywniać obiekty.

Zaawansowane aplikacje internetowe

Object Request Broker – wprowadzenie

- ORB – centralna szyna komunikacji
- Najważniejsza cecha ORB – przezroczystość
  - Lokalizacja obiektu
  - Implementacja obiektu
  - Stan działania obiektu
  - Komunikacja między obiektami
- Zarządzanie referencjami do obiektów
  - Nieczytelne, niemodyfikowalne, nieusuwalne
  - Pozyskiwanie referencji
    - Tworzenie obiektu
    - Usługi katalogowe
    - Serializacja referencji do pliku lub bazy danych

CORBA (16)

Moduł brokera obiektów (ang. ORB, Object Request Broker) to najważniejsza, centralna część architektury CORBA. ORB stanowi główną szynę danych, za pomocą której komunikują się ze sobą obiekty. ORB przesyłają referencje do obiektów (odwzorowując referencje na szkielety), przekazują struktury danych, wywołują metody na rzecz lokalnych usługodawców i zdalnych obiektów. ORB oferuje całkowitą transparentność w następujących dziedzinach:

•Lokalizacja obiektu: klient wywołując metodę na obiekcie nie widzi żadnej różnicy między obiektami lokalnymi i zdalnymi, fizyczna lokalizacja usługodawcy jest całkowicie nieistotna

•Implementacja obiektu: wszystkie szczegóły implementacyjne usługodawcy (np. język programowania, w jakim napisany jest usługodawca) są całkowicie niewidoczne dla klienta

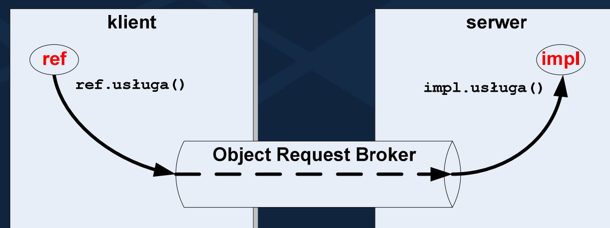
•Stan działania obiektu: z punktu widzenia klienta każdy usługodawca jest zawsze gotowy do przyjmowania i obsługiwania żądań, podczas gdy w rzeczywistości obiekt usługodawcy może wymagać uaktywnienia lub może aktualnie obsługiwać inne żądanie

•Komunikacja między obiektami: fizyczny sposób wywołania metody usługodawcy (komunikacja po protokole TCP/IP, pamięć współdzielona) jest nieistotny z punktu widzenia klienta.

Żądanie wykonania usługi jest przekazywane przez klienta usługodawcy za pomocą referencji na obiekt usługodawcy. Referencje są tworzone automatycznie w momencie tworzenia każdego obiektu. Referencje są nieczytelne i zrozumiałe tylko dla ORB, dodatkowo referencja do obiektu jest nieusuwalna i niezmienna w trakcie całego cyklu życia obiektu. Referencje można pozyskać w momencie tworzenia obiektu (tworzenie obiektu jest obowiązkiem aplikacji a nie architektury CORBA), za pomocą usług katalogowych (np. usługi nazw, usługi zostaną opisane w dalszej części wykładu), lub odczytane z pliku bądź z bazy danych. W ostatnim przypadku postać tekstowa referencji jest wynikiem operacji serializacji referencji.

ORB - usługodawcy

- Usługodawca: konkretna implementacja operacji obiektu CORBA wyspecyfikowanych w deskryptorze IDL
  - Wiele referencji odwzorowywanych na usługodawcę
  - Jedna referencja odwzorowywana na wielu usługodawców
  - Obiekt może nie mieć aktywnych usługodawców
  - Obiekt może wcale nie mieć usługodawców



Usługodawcą (ang. servant) nazywamy dowolny obiekt (lub inny element) przygotowany w konkretnym języku programowania i implementujący wszystkie operacje obiektu CORBA wyspecyfikowane w deskryptorze IDL obiektu. Każdy obiekt jest dostępny z poziomu klientów poprzez referencje. Odwzorowaniem referencji na usługodawcę zajmuje się moduł ORB. Wiele różnych referencji może być odwzorowywanych na tego samego usługodawcę, ale każda referencja może być też odwzorowywana na wielu różnych usługodawców podczas swego działania. Mogą także istnieć obiekty, które nie posiadają aktualnie żadnych aktywnych usługodawców (aktywizacja usługodawców jest obowiązkiem ORB), lub nie posiadają wcale żadnych usługodawców.

Zaawansowane aplikacje internetowe

ORB – lista dostępnych implementacji

- IIOP.NET, http://iiop-net.sourceforge.net/
- JacORB, http://www.jacorb.org/
- Java IDL, http://java.sun.com/products/jdk/idl/
- MICO is CORBA, http://www.mico.org/
- omniORB, http://omniorb.sourceforge.net/
- OpenORB, http://openorb.sourceforge.net/
- TheAceORB, http://www.cs.wustl.edu/~schmidt/TAO.html
- VisiBroker, http://www.borland.com/us/products/visibroker

CORBA (18)

Powyższy slajd przedstawia, niekompletną rzecz jasna, listę dostępnych implementacji ORB dla różnych języków programowania. W większości przypadków głównymi językami programowania są C++ i Java. Poszczególne implementacje różnią się konfiguracją, zakresem zaimplementowanych usług i stopniem zgodności ze standardem CORBA. Do celów dydaktycznych najlepiej się posłużyć implementacją Java IDL firmy Sun, ponieważ stanowi ona część platformy Java 2 Standard Edition (J2SE 1.4.2).

Usługi CORBA

– Life Cycle Service
– Persistence Service
– Naming Service
– Event Service
– Concurrency Control Service
– Transaction Service
– Relationship Service
– Query Service
– Properties Service
– Time Service

Istotnym elementem architektury CORBA jest definicja zbioru usług wspomagających integrację i współpracę rozproszonych obiektów. Te usługi, dla których implementacje architektury CORBA dostarczają gotowych usługodawców, nazywane są usługami CORBA (ang. CORBAServices), lub usługami obiektowymi (ang. Object Services). Są to gotowe, standardowe obiekty CORBA z własnymi definicjami interfejsów w IDL. Najważniejsze i najbardziej użyteczne usługi CORBA to:

•Life Cycle Service: definiuje sposób tworzenia, kopiowania, przenoszenia i niszczenia obiektów

•Persistence Service: definiuje wygodny interfejs zapewniania obiektom trwałości w obiektowych i relacyjnych bazach danych oraz w plikach dyskowych

•Naming Service: pozwala odwoływać się do obiektów za pomocą systemu hierarchicznych nazw oraz integruje hierarchię nazw z istniejącymi systemami, takimi jak NIS+ czy LDAP

•Event Service: umożliwia wiązanie obiektów z określonymi kategoriami zdarzeń

•Concurrency Control Service: umożliwia stosowanie pesymistycznych algorytmów kontroli współbieżnego dostępu do obiektów za pomocą systemu blokad

•Transaction Service: umożliwia stosowanie prostych i zagnieżdżonych transakcji z algorytmem koordynacji 2PC

•Relationship Service: oferuje tworzenie i usuwanie związków między obiektami, pozwala na wymuszanie ograniczeń referencyjnych oraz umożliwia nawigację między obiektami za pomocą zdefiniowanych związków

•Query Service: umożliwia dostęp do obiektów poprzez języki zapytań SQL i OQL

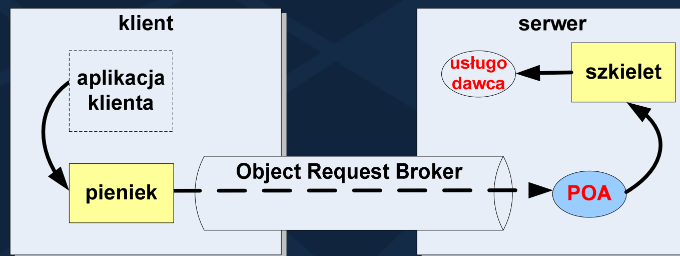
•Properties Service: pozwala na dynamiczne wiązanie z obiektami własności (atrybutów)

•Time Service: oferuje interfejs do synchronizacji czasu w systemie rozproszonym, pozwala na definiowanie czynności aktywowanych przez upływ czasu

Zaawansowane aplikacje internetowe

POA – Portable Object Adapter

- Odpowiedzialność adaptera obiektów
  - Generowanie i interpretacja referencji do obiektów
  - Wywoływanie metod
  - Bezpieczeństwo interakcji między obiektami
  - Aktywacja i dezaktywacja obiektów
  - Rejestrowanie implementacji interfejsów



CORBA (20)

Adapter obiektów (ang. object adapter) to środowisko implementacji obiektów CORBA. Jest to moduł systemowy odpowiedzialny za usługi niskiego poziomu: generowanie i interpretację referencji do obiektów, przekazywanie parametrów i wywoływanie metod lokalnych, zapewnianie bezpieczeństwa interakcjom między obiektami, aktywację i dezaktywację usługodawców, czy rejestrowanie nowych obiektów implementujących interfejsy IDL. Specyfikacja CORBA zawiera definicję jednego standardowego adaptera BOA (ang. basic object adapter). Niestety, implementacje adaptera BOA bardzo istotnie się od siebie różnią w implementacjach CORBA, co bardzo poważnie ogranicza przenaszalność kodu. W związku z tym zaproponowano adapter POA (ang. portable object adapter), którego implementacje powinny być praktycznie takie same dla wszystkich implementacji CORBA. POA jest nowocześniejszą wersją adaptera BOA i powinien być używany we wszystkich nowopowstających aplikacjach. Poza standardowymi operacjami POA umożliwia również transparentną aktywację obiektów, istnienie wielu równoległych tożsamości jednego obiektu, wykorzystywanie obiektów ulotnych, występowanie wielu adapterów POA w ramach jednego serwera. Z każdym ORB związany jest jeden adapter korzenia (ang. root POA), wewnątrz którego istnieje hierarchiczne drzewo zależnych adapterów POA. Każdy adapter POA może definiować własne reguły zarządzania obiektami, referencjami do obiektów i usługodawcami.

Zaawansowane aplikacje internetowe

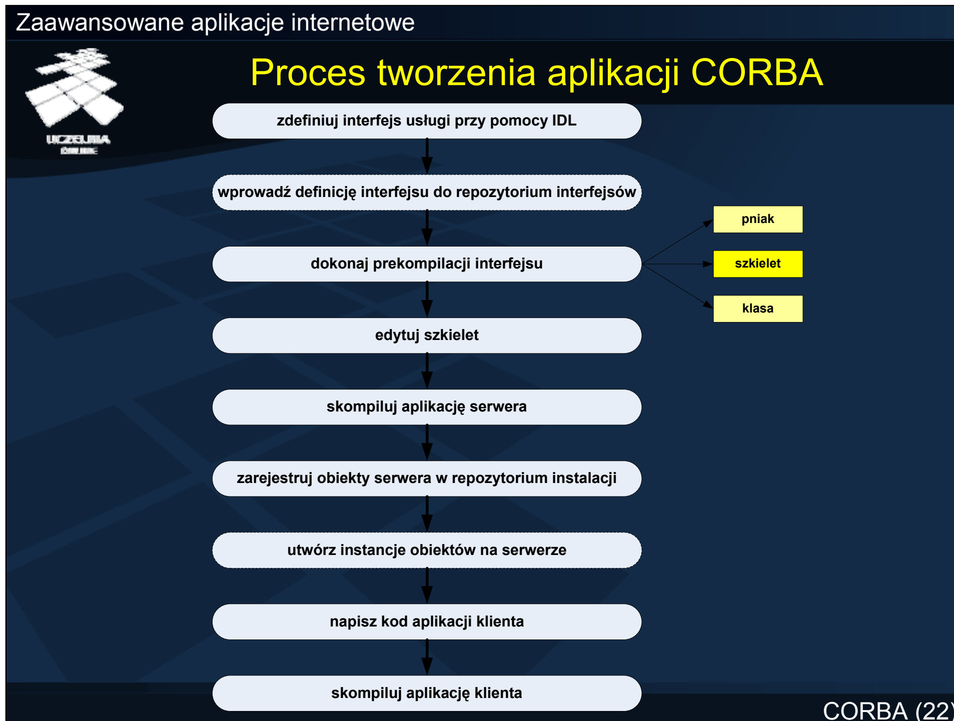
Model komunikacji – protokół IIOP

- General Inter-ORB Protocol
  - IIOP – Internet Inter-ORB Protocol
  - HTIOP – Hypertext Inter-ORB Protocol
  - XIOP – XML Inter-ORB Protocol
- Wspólna reprezentacja danych (CDR)
- Interoperable Object Reference
- Formaty komunikatów: żądanie, odpowiedź, anulowanie, lokalizacja żądania, lokalizacja odpowiedzi, zamknięcie połączenia, błąd, fragment

CORBA (21)

Moduły ORB wchodzące w skład architektury CORBA wykorzystują abstrakcyjny protokół General Inter-ORB Protocol (GIOP) do komunikacji i wymiany żądań, danych i odpowiedzi. Protokół GIOP jest abstrakcyjny i niezależny od warstwy transportowej. Istnieją szczegółowe implementacje tego protokołu, przystosowane do konkretnego środowiska sieciowego. Najpopularniejszą i najczęściej stosowaną implementacją jest protokół Internet Inter-ORB Protocol (IIOP) wykorzystujący w warstwie transportowej protokół TCP/IP. Istnieją również implementacje GIOP przenoszące informacje za pomocą języków-hostów, np. HTIOP (GIOP wykorzystujący hipertekst) lub XIOP (GIOP wykorzystujący język XML).

Na protokół GIOP składają się trzy elementy. Wspólna reprezentacja danych (ang. Common Data Representation, CDR) definiuje odwzorowanie typów OMG IDL na reprezentację niskiego poziomu, która może być przesyłana za pomocą medium komunikacyjnego. Wspólna referencja obiektowa (ang. Interoperable Object Reference, IOR) definiuje format referencji do obiektów zrozumiały i wykonywalny dla wszystkich modułów ORB. Wreszcie GIOP definiuje zbiór komunikatów, które mogą być przesyłane między modułami ORB. Są to: żądanie usługi, odpowiedź z usługi, anulowanie żądania, lokalizacja żądania (sprawdzenie, czy dany serwer rozumie żądanie, a jeśli nie, czy wie, gdzie można żądanie przesłać), lokalizacja odpowiedzi, zamknięcie połączenia z serwerem, sygnalizacja wystąpienia błędu oraz komunikat typu "fragment" kontynuujący poprzedni komunikat (długie komunikaty mogą być rozdzielane na paczki).



Slajd przedstawia proces tworzenia aplikacji CORBA krok po kroku. Na początku należy zdefiniować interfejs usługi przy pomocy języka IDL. Opcjonalnie, definicja interfejsu może zostać wprowadzona do repozytorium interfejsów. Kolejny krok to prekompilacja definicji interfejsu za pomocą specjalnego narzędzia dostarczanego przez implementację CORBA. W wyniku tego kroku powstają: pieniek klienta służący do zgłaszania żądania usługi, implementacja interfejsu w postaci klasy w konkretnym języku programowania, oraz szkielet usługodawcy. Ostatni element wymaga edycji i zaimplementowania logiki zadeklarowanej w interfejsie. Po edycji szkieletu należy skompilować całą aplikację po stronie serwera. Kolejnym krokiem jest uruchomienie modułu ORB po stronie serwera i zarejestrowanie przygotowanego szkieletu w repozytorium instalacji. Na podstawie zawartości repozytorium instalacji adapter obiektów będzie w stanie przekierować żądanie usługi do właściwego szkieletu. Opcjonalnie, można utworzyć i aktywować instancje obiektów po stronie serwera, choć można także pozostawić ten obowiązek na barkach modułu ORB. Ostatni etap to napisanie aplikacji klienta korzystającej z automatycznie wygenerowanego pieńka klienta do wołania usług. Po skompilowaniu aplikacji klienta i uruchomieniu modułu ORB na komputerze klienta aplikacja jest gotowa do działania. Kolejne slajdy przedstawiają przykład aplikacji CORBA napisany w języku Java i wykorzystujący środowisko J2EE Standard Edition 1.5 jako moduł ORB.

Definicja i prekompilacja interfejsu

messageBox.idl

```
module MessageModule {
   typedef sequence<string> MessageSeq;
   interface MessageBox {
        exception boxFull {};
        attribute string reply;
        string leaveMessage(in string msg) raises (boxFull);
        MessageSeq getMessages();
   };
};
```

```
C:\> idlj.exe -fserver messageBox.idl
```
(1)

```
C:\> orbd.exe -ORBInitialPort 1050
```
(2)

Slajd przedstawia plik IDL zawierający interfejs usługi MessageBox. Usługa będzie umieszczona w pakiecie MessageModule i będzie oferować operacje leaveMessage() i getMessage() oraz będzie zawierać atrybut reply (dla której automatycznie zostaną wygenerowane metody dostępowe) i wyjątek boxFull. Środowisko J2SE 1.4.2 (i wyższe) zawiera prekompilator idlj.exe umieszczony w katalogu %JAVA_HOME%\bin. Polecenie dokonujące prekompilacji specyfikacji IDL i generujące wszystkie wynikowe pliki po stronie serwera przedstawiono w kroku (1). Krok (2) przedstawia polecenie konieczne do uruchomienia brokera obiektów ORB na lokalnym komputerze. Implementacja ORB w środowisku J2SE jest uruchamiana za pomocą polecenia (2) (konieczne jest wskazanie portu TCP/IP do nasłuchu), a sam broker orbd.exe jest umieszczony w katalogu %JAVA_HOME%\bin.

Automatycznie generowane pliki

```
[dir] MessageModule
├─ MessageBox.java
├─ MessageBoxOperations.java
├─ MessageBoxPOA.java
├─ MessageSeqHelper.java
├─ MessageSeqHolder.java
└─ [dir] MessageBoxPackage
    ├─ boxFull.java
    ├─ boxFullHelper.java
    └─ boxFullHolder.java
```


W wyniku prekompilacji za pomocą narzędzia idlj.exe powstaje następująca struktura katalogów. Wszystkie pliki są umieszczone w katalogu MessageModule (nazwa modułu ze specyfikacji IDL). MessageBox to interfejs usługi udostępniany klientom zewnętrznym. W rzeczywistości interfejs jest pusty, dziedziczy z interfejsu MessageBoxOperations zawierającego wszystkie operacje wyspecyfikowane w IDL. MessageBoxPOA to szkielet usługodawcy implementujący podstawowe metody interakcji obiektu z POA oraz interfejs org.omg.ProtableServer.Servant, który definiuje metody callback dla POA. Każdy obiekt który ma stanowić usługodawcę musi dziedziczyć z klasy MessageBoxPOA. Klasy MessageSeqHelper i MessageSeqHolder to klasy implementujące zdefiniowany w IDL typ MessageSeq będący sekwencją łańcuchów znaków. Klasa MessageSeqHelper to abstrakcyjna klasa zawierająca wszystkie operacje (wstawienie do sekwencji, odczytanie z sekwencji, usunięcie z sekwencji) zaimplementowane w postaci składowych statycznych. Klasa MessageSeqHolder to klasa finalna wykorzystująca składowe statyczne z MessageSeqHelper. Wreszcie podkatalog MessageBoxPackage zawiera klasę boxFull implementującą zdefiniowany w IDL wyjątek boxFull oraz dwie klasy pomocnicze, boxFullHelper i boxFullHolder. Przeznaczenie i sposób działania dwóch ostatnich klas jest identyczne z zachowaniem analogicznych klas dla typu MessageSeq.

Zaawansowane aplikacje internetowe

Przygotowanie usługi 1/2

MessageBoxImpl.java

```
public class MessageBoxImpl extends MessageBoxPOA {        (1)
  public MessageBoxImpl(String name, int max) { … }

  protected String _reply;
  protected int _max;                                       (2)
  protected Vector _messages;
  private ORB orb;

  public String leaveMessage(String msg)                    (3)
    throws MessageModule.MessageBoxPackage.boxFull {
    if (_messages.size() > max)
      throw new MessageModule.MessageBoxPackage.boxFull();  (4)
    _messages.addElement(msg);
    return reply(); }
  …
```

CORBA (25)

Slajd przedstawia kod klasy implementującej usługę. Aby adapter obiektów POA mógł bez przeszkód zarządzać usługodawcą, w linii (1) klasa usługodawcy dziedziczy z automatycznie wygenerowanego szkieletu MessageBoxPOA, co pozwala adapterowi obiektów POA na zarządzanie usługodawcą poprzez wywoływanie metod callback zdefiniowanych w klasie MessageBoxPOA. W części (2) usługodawca deklaruje właściwe składowe. Istotne jest, że usługodawca posiada referencję do swojego brokera obiektów ORB. Dalsza część kodu to implementacja interfejsu MessageBox. Linia (3) i dalsze przedstawiają implementację metody leaveMessage(String). Linia (4) pokazuje przykład zgłoszenia wyjątku zadeklarowanego w pliku IDL.

Przygotowanie usługi 2/2

MessageBoxImpl.java

```
…
public void setORB(ORB orb) { this.orb = orb; }   (1)

public void reply(String reply) { reply = reply; }

public String reply() { return reply; }   (2)

public java.lang.String[] getMessages() {
  String[] messages = new String[_messages.size()];
   _messages.copyInto(messages);   (3)
  _messages = new Vector(_max);
  return messages;
 }
}
```


Slajd kontynuuje prezentację implementacji usługi. Poza dostarczeniem metody do ustawiania aktualnego ORB klasa MessageBoxImpl posiada również implementację pozostałych metod z interfejsu: metod dostępowych do składowej _reply oraz kod usługi getMessages().

Zaawansowane aplikacje internetowe

Przygotowanie serwera 1/2

MessageServer.java

```
…
public static void main(String[] args) {
  try {
    Properties props = new java.util.Properties();
    props.put("org.omg.CORBA.ORBInitialPort", "1050");      (1)
    props.put("org.omg.CORBA.ORBInitialHost", "localhost");

    ORB orb = ORB.init(args,props);
    POA rootpoa =
      POAHelper.narrow(orb.resolve_initial_references("RootPOA"));   (2)
    rootpoa.the_POAManager().activate();
    …
```

CORBA (27)

Kolejny slajd przedstawia kod serwera. Serwer jest odpowiedzialny za nawiązanie połączenia z brokerem obiektów, pobranie referencji do korzenia hierarchii adaptera obiektów, powołanie do życia jednej bądź wielu instancji usługodawców i zarejestrowanie usługodawców w brokerze obiektów. Klasa MessageServer nie musi dziedziczyć z żadnej innej klasy ani nie musi implementować żadnych konkretnych interfejsów. Zawiera tylko metodę main() przedstawioną na tym i następnym slajdzie. Ustawienie parametrów brokera obiektów odbywa się za pomocą standardowego mechanizmu Properties (1). Serwer wskazuje host docelowy i port docelowy, na którym będzie aktywowany broker obiektów. Kolejnym krokiem jest pobranie referencji do korzenia hierarchii adapterów obiektów, tzw. "RootPOA" i aktywowanie tego adaptera (2).

Przygotowanie serwera 2/2

MessageServer.java

```
…
MessageBoxImpl messageBoxImpl = new MessageBoxImpl("myMessages",5);
 messageBoxImpl.setORB(orb);                                  (1)

 org.omg.CORBA.Object ref = rootpoa.servant_to_reference(messageBoxImpl);
 MessageBox box = MessageBoxHelper.narrow(ref);                (2)

 org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
 NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);   (3)

 NameComponent path[] = ncRef.to_name("MessageServer");
 ncRef.rebind(path, box);                                      (4)
 orb.run();                                                    (5)
```


Kolejny krok to utworzenie nowego obiektu usługodawcy i powiązanie tego obiektu z brokerem obiektów ORB (1). Serwer musi jeszcze pobrać referencję do nowo utworzonego usługodawcy w celu zarejestrowania usługodawcy w usłudze nazewniczej. W tym celu pobiera referencję z adaptera obiektów POA jako generyczny obiekt org.omg.CORBA.Object i następnie zawęża referencję do właściwego typu za pomocą metody narrow() (2). Krok (3) to pobranie referencji do usługi nazewniczej (usługi CORBA są zwyczajnymi obiektami) i zawężenie referencji do właściwego typu. Ostatni krok to wykorzystanie metody rebind() do powiązania obiektu usługodawcy z nazwą symboliczną "MessageServer" (4). Dzięki temu klienci będą mogli lokalizować usługodawcę za pomocą tej nazwy. Wreszcie, program serwera aktywuje brokera obiektów za pomocą wywołania metody run() (5).

Przygotowanie klienta 1/3

```
C:\> idlj.exe -fclient messageBox.idl
```

[dir] MessageModule
├── MessageBoxHelper.java
├── MessageBoxHolder.java
└── _MessageBoxStub.java

Pierwszą czynnością konieczną dla przygotowania programu funkcjonującego jako klient usługi CORBA jest automatyczne wygenerowanie pieńka na podstawie specyfikacji IDL. Używa się do tego celu tego samego prekompilatora IDL (%JAVA_HOME%\bin\idlj.exe) co w przypadku plików serwera. W rzeczywistości można utworzyć wszystkie pliki potrzebne po stronie serwera i klienta za pomocą jednego polecenia z flagą –fall. Do katalogu MessageModule zawierającego cały moduł usługi zostają dopisane trzy pliki. MessageBoxHelper.java to abstrakcyjna klasa zawierająca pomocnicze metody, przede wszystkim metodę narrow() umożliwiającą rzutowanie obiektów CORBA na właściwe typy. MessageBoxHolder.java to definicja drugiej pomocniczej klasy wykorzystywanej w sytuacjach gdy typy IDL są wykorzystywane jako parametry wyjściowe lub wejściowo-wyjściowe. Najważniejszym plikiem generowanym przez prekompilator jest _MessageBoxStub.java, zawierający gotowy pieniek aplikacji klienta.

Przygotowanie klienta 2/3

MessageClient.java

```
…
public static void main(String[] args) {
  try {
    Properties props = new java.util.Properties();
    props.put("org.omg.CORBA.ORBInitialPort", "1050");
    props.put("org.omg.CORBA.ORBInitialHost", "localhost");

    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
    org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
    NamingContext ncRef = NamingContextHelper.narrow(objRef);

    NameComponent nc = new NameComponent("MessageServer", "");
    NameComponent path[] = { nc };
    MessageBox box = MessageBoxHelper.narrow(ncRef.resolve(path));
```

(1) (2) (3) (4)

Powyższy slajd przedstawia kod klienta. Podobnie jak w przypadku serwera, w pierwszej kolejności następuje nawiązanie połączenia z brokerem obiektów. Parametry docelowego brokera są specyfikowane za pomocą tradycyjnego mechanizmu java.util.Properties (1). Statyczna metoda init() powoduje nawiązanie połączenia ze zdalnym brokerem obiektów (2). W powyższym przykładzie klient chce się posłużyć usługą nazewniczą w celu zlokalizowania usługodawcy. Konieczne jest więc odczytanie referencji do obiektu "NameService" i rzutowanie tego obiektu na właściwy typ (3). Kolejny krok to przygotowanie obiektu typu NameComponent zawierającego opis poszukiwanego obiektu i wykorzystanie metody resolve() usługi nazewniczej do pobrania referencji do usługodawcy (4).

Przygotowanie klienta 3/3

MessageClient.java

```
…
if (action.equals("reply")) { box.reply(param); }                                (1)
else if (action.equals("leave")) {
  try {
    System.out.println(box.leaveMessage(param));                                 (2)
  } catch (MessageModule.MessageBoxPackage.boxFull e) { e.printStackTrace(); }
}
else if (action.equals("get")) {
  String[] messages = box.getMessages();
  for(int i = 0; i < messages.length; i++)                                       (3)
    System.out.println("  " + messages[i]);
}
…
```


Ostatni slajd przedstawia drugą część programu klienta. Po wcześniejszym pobraniu referencji do obiektu usługodawcy program klienta wywołuje, w zależności od parametru uruchomieniowego, konkretną metodę z interfejsu IDL. Slajd pokazuje przykład odczytania wiadomości powitalnej (1), pozostawienia nowego komunikatu na sekretarce (2) oraz wyświetlenie całej zawartości sekretarki (3).

Materiały dodatkowe

- CORBA Faq, www.omg.org/gettingstarted/corbafaq.htm
- Object Managemet Group, http://www.omg.org
- "Java Programming with CORBA", G.Brose, A.Vogel, K.Duddy, Wiley Computer Publishing, 2001
- Introduction to CORBA, http://java.sun.com/developer/onlineTraining/corba/