ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 177

Seria: INFORMATYKA z. 30 Nr kol. 1315

Małgorzata STEINDER
Andrzej USZOK
Krzysztof ZIELIŃSKI

ARCHITEKTURA WSPÓŁPRACY ROZPROSZONYCH SYSTEMÓW OBIEKTOWYCH ZGODNYCH ZE SPECYFIKACJĄ CORBA

Streszczenie. Niniejszy artykuł dotyczy problemu współdziałania systemów bądących różnymi implementacjami modelu CORBA. Zaprezentowano sposób rozwiązania tego problemu poprzez użycie modułów *most* i *pół-most* zaproponowanych przez specyfikacją *Universal Networked Objects*. Opisano model funkcjonalny takiego modułu oraz ramową architekturą jego budowy. Szczegółowo rozważono problem translacji referencji obiektów pomiądzy domenami różnych systemów. Jako przykład zastosowania zaprojektowanej architektury zaprezentowano budową modułu *pół-mostu* dla systemu Orbix.

THE ARCHITECTURE FOR COOPERATION OF DISTRIBUTED OBJECT ORIENTED SYSTEMS BASED ON CORBA

Summary. The paper addresses a problem of building a bridge between different CORBA compliant systems. It presents a framework of the bridge based on the UNO approach whose architecture is easily extendable to more sophisticated in parallelizing level and functionality units. A problem of mapping objects defined in CORBA model is described and a few suggestions to deal with it are presented. As a case study implementation of the half-bridge for Orbix is described.

Pracę wykonano w ramach projektu COPERNICUS nr CP940247

DIE ARCHITEKTUR FÜR KOOPERATION VON DISPERGIERTEN OBJEKT-ORIENTED CORBA-GEMÄSS SYSTEMEN

Zusammenfassung. In diesem Artikel beschrieben wird, wie mit CORBA übereinstimmende Systeme zusammenarbeiten können. Die Architektur und Funktion vom Kooperationmodul wird vorgestellt. Die Auslesung von CORBA Objekten wird auch beschrieben, besonders, die Metode von Objektreferenzübersetzung wird erklärt. Als Beispiel, der Kooperationmodul für Orbix wird gezeigt.

1. Wstęp

Intensywny rozwój sieci komputerowych oraz aplikacji rozproszonych stawia nowe wymagania w zakresie unifikacji konstrukcji oprogramowania oraz technik zapewniających współdziałanie. Znaczącą pozycje w tym nurcie badań stanowią prace koordynowane przez OMG (Object Management Group), dotyczące architektury CORBA (Common Object Request Broker Architecture)[1].

Punktem wyjściowym tych prac jest obiektowy model przetwarzania informacji złożony z obiektów sieciowych, pomiędzy którymi za pośrednictwem brokera przekazywane są żądania wykonania operacji. Obiekt sieciowy w architekturze CORBA posiada dwie charakteryczne cechy, tj.: interfejs wyspecyfikowany w języku IDL (Interface Definition Language) oraz referencję jednoznacznie go identyfikującą w sieci. Zadaniem brokera jest przekazanie żądania wykonania operacji od klienta do odpowiedniego obiektu sieciowego a następnie dostarczenie z powrotem wyniku jej realizacji.

Architektura CORBA wnosi dwie bardzo istotne cechy strukturalne, decydujące o własnościach budowanej aplikacji:

- Niezależność implementacji obiektów od specyfikacji ich interfejsów, przy czym interfejs stanowi zbiór operacji realizowanych przez obiekt. Pozwala to na implementację obiektów w różnych językach programowania, a w szczególności wykorzystywanie już istniejącego kodu do realizacji funkcji odpowiadających poszczególnym operacjom. Dodatkowo faza projektowania oprogramowania może dotyczyć wyłącznie poziomu interfejsów.

- Dynamiczne wiązanie klienta wywołującego operacje oraz obiektu, który posiada interfejs zawierający te operacje. Umożliwia to budowę aplikacji na zasadzie łączenia określonych modułów funkcjonalnych już w fazie wykonania programu, przy czym funkcje wiązania realizowane są poprzez brokera. Podejście to umożliwia osiągnięcie pełnej przezroczystości lokalizacji w systemie rozproszonym oraz budowy specjalizowanych obiektów bibliotecznych (Common Object Services) [5,6].

Z punktu widzenia klienta przetwarzanie informacji odbywa się analogicznie do modelu klient/serwer, odmienne są natomiast niektóre mechanizmy wywołania operacji i przekazania wyniku.

Specyfikacja CORBA 1.2 jest wewnętrznie zgodna z modelem, jaki reprezentuje. Opisuje ona zatem architekturę budowy sytemu w kategoriach interfejsów, nie precyzując implementacji ich funkcji oraz sposobu kodowania informacji. Powoduje to, że implementacje modelu CORBA zrealizowane przez różne firmy nie mogą bezpośrednio współpracować.

Celem artykułu jest przedstawienie problemów współdziałania systemów obiektowych zgodnych ze specyfikacją CORBA oraz koncepcji budowy bramy rozwiązującej ten problem w wyniku komunikacji z wybraną bazową implementacją systemu typu ORB.

2. Modele współpracy środowisk typu ORB

Osiągnięcie współdziałania systemów typu CORBA ma na celu umożliwienie przekazania żądania wykonania operacji pochodzącego od klienta, znajdującego się w jedym systemie, do obiektu reprezentującego serwer znajdujący się w innym systemie. Osiągnięcie tego celu jest możliwe na dwoma sposobami:

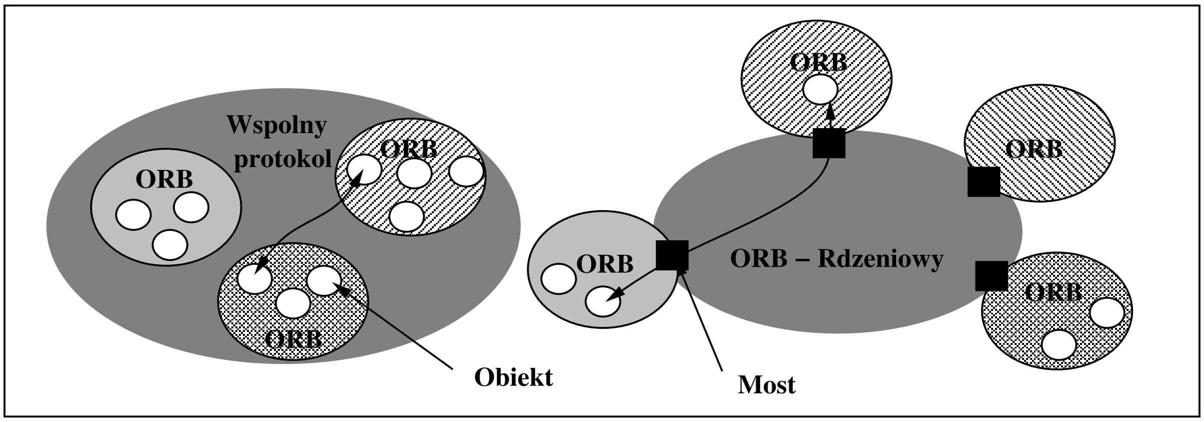
1. Poprzez ustalenie wspólnej reprezentacji danych oraz protokołu przekazywania żądań pomiędzy klientem a obiektem sieciowym reprezentującym serwer.

2. Poprzez translację żądań oraz rezultatów ich wykonania w trakcie przekazywania pomiędzy klientem i serwerem.

Należy podkreślić, że przez systemy typu CORBA rozumie się w niniejszym artykule takie środowiska programowe, których IDL oraz interfejsy mechanizmów systemu są zgodne ze specyfikacją OMG [1].

Wymienione sposoby prowadzą do dwu odmiennych architektur współdziałania systemów typu CORBA, przestawionych na rys.1.

Sposób pierwszy wymaga ujednolicenia implementacji systemów typu CORBA, co nie jest możliwe wobec różnorodnych zastosowań i związnych z tym specyficznych wymagań odnośnie do protokołów komunikacji oraz reprezentacji danych. Ponadto trudno jest sobie wyobrazić tak daleko posunięty konsensus pomiędzy różnymi producentami oprogramowania. Wydaje się zatem, że rozważając ten model należy brać pod uwagę, iż wspólny



Rys. 1. Modele współpracy systemów typu CORBA
Fig. 1. Model of cooperation of CORBA-compliant systems

protokół przekazywania żądań może stanowić mechanizm alternatywny w stosunku do bazowego dla danego środowiska. Prowadzi to do sytuacji, w której każdy obiekt typu klient bądź serwer będzie mógł komunikować się co najmniej za pomocą dwu protokołów.

Drugi sposób zakłada budowę specjalizowanyh obiektów pełniących funkcje bram lub mostów pomiędzy domenami, w których obowiązuje różny protokół przekazywania żądań. Bramy realizują zazwyczaj translacje pomiędzy kilkoma protokołami, realizując dodatkowo funkcję marszrutowania żądań, natomiast mosty są specjalizowane w odniesieniu do dwu wybranych protokołów. Bramy mogą realizować translacje protokołu i formatu żądań pomiędzy dwoma wybranymi formatami bądź pomiędzy danym a wybranym jednym standardowym protokołem współdziałania. Druga możliwość prowadzi do koncepcji jednego wybranego ORB-u, który stanowi rdzeń umożliwiający współdziałanie. Systemy typu CORBA są łączone z ORB-em rdzeniowym za pomocą tzw. pół–mostów. Para pół–mostów wraz z ORB-em rdzeniowym tworzy most. Podejście to łączy pierwszy i drugi sposób osiągnięcia współdziałania pomiędzy systemami typu CORBA i posiada następujące zalety:

1. Jest skalowalne dla dowolnej liczby współdziałających systemów.
2. Pozwala na wydzielenie odrębnych domen w sensie zarządzania - w szczególności bramy mogą realizować kontrolę dostępu do określonych domen.
3. Pozostawia możliwości specjalizcji protokołu przekazywania żądań w poszczególnych domenach.
4. Wprowadza hierarchiczną organizację przepływu żądań w systemie, co ma szczególne znaczenie w dużych systemach sieciowych.

Niewątpliwą wadą rozwiązania ze wspólnym protokołem rdzeniowym jest efektywność. Każde żądanie przekazywane pomiędzy różnymi systemami typu CORBA musi bowiem podlegać translacji dwukrotnie. Jeżeli zatem w aplikacji występują krytyczne uzależnienia czasowe, należy rozważyć użycie rozwiązania wykorzystującego wieloprotokołowe obiekty klientów i serwerów.

Zagadnienia współdziałania stanowią przedmiot specyfikacji OMG znanej pod nazwą CORBA 2.0. (Universal Networked Object) [4]. Specyfikacja ta rozważa różne modele współdziałania pomiędzy systemami typu CORBA oraz precyzuje wspólny protokół i format przekazywania żądań, zwany IIOP (Internet Inter-ORB Protocol). Stwarza to podstawę dla konstrukcji bram oraz wskazówkę, że jednym z protokołów wieloprotokołowych obiektów powinien być właśnie protokół IIOP.

3. Problemy współdziałania sytemów zgodnych z modelem CORBA

Zgodnie z wcześniejszymi rozważaniami, punktem wyjścia dla osiągnęcia współdziałnia w sytemach typu CORBA jest translacja protokołu przekazywania żądań. Translacja ta może być realizowana wewnątrz brokera (ORB), co wymaga znajomości jego wewnętrznej budowy, bądź na zewnątrz przez obiekty, zwane bramami lub mostami, których zadaniem jest przechwytywanie żądań i ich konwersja.

Dalsza klasyfikacja bram oraz mostów wynika ze stopnia ogólności implementacji funkcji translacji żądań. W przypadku, gdy te specjalizowane obiekty mają możliwość odebrania dowolego żądania, to mówimy, iż są one ogólne. Konstrukcja ogólnych mostów i bram wymaga dostępności dynamicznego interfejsu tworzenia żądań (DII - Dynamic Invocation Interface) oraz ich odbioru (DSI - Dynamic Skeleton Interface). O ile pierwszy z wymienionych interfejsów jest zazwyczaj dostarczany w ramach implementcji sytemu typu CORBA, to drugi musi zostać skonstruowany dla każdego systemu objętego współdziałaniem. Obydwa typy interfejsów wymagają dynamicznego dostępu do definicji operacji interfejsów zawartych w Bazie Interfejsów (IR).

Proces translacji żądania napotyka także na szereg trudności, takich jak:

- Konieczność translacji złożonych struktur danych opisanych za pomocą tzw. kodów typów (TypeCode). Reprezentacja tych typów jest różna dla każdego systemu typu CORBA.

- Brak ogólnych zasad translacji (odwzorowania) referencji oraz kontekstu operacji pomiędzy różnymi systemami.

- Nieustalony mechanizm przekazywania translacji wyjątków mogących wystąpić w efekcie wykonania operacji.

- Brak zasad przekazywania informacji o kliencie generującym żądanie (principal) i związanych z tym praw do wykonania określonych operacji. Zagadnienie to wiąże się bezpośrednio z ochroną informacji w sieci.

Powstanie specyfikacji CORBA 2.0 w formie dokumentu Universal Networked Object [4] ustala punkt odniesienia dla prac w zakresie współdziałania systemów typu CORBA, pozostawiając jednakże nadal szereg otwartych problemów technicznych.

Zdefiniowanie samego protokołu przekazywania żądań nie wystarcza dla budowy aplikacji wykorzystującej kilka systemów typu CORBA. W szczególności IIOP nie zawiera w sobie funkcji brokera (ORB) oraz funkcji Adaptera Obiektów (OA), pozostawiając je do implementacji. Sprawia to, że takie elementy systemu, jak np.: Bazy Interfejsów, Bazy Implementacji oraz usług Aktywizacji i Inicjacji obiektów systemu leżą poza rozważaną specyfikacją. Podobnie techniki zapewnienia konsystentności Repozytoriów Interfejsów oraz rozgłaszania usług w systemie, mające na celu znalezienie systemu ORB i referencji obiektu dostarczającego określonej usługi, pozostają nieokreślone.

4. Architektura modułu wspóldziałania – *pół-most*

Celem tego punktu jest ogólne przedstawienie działania i budowy modułu współdziałania pół-most różnych systemów zgodnych z modelem CORBA, który został zaproponowany w specyfikacji Universal Netwoked Objects [3]. Zadania tego modułu zostały dokładnie określone, to znaczy ma on translować żądania wykonania operacji obiektów sieciowych pomiędzy danym systemem zgodnym z CORBA a domeną protokołu IIOP. Dokładna analiza tego zadania przedstawiona poniżej sprawia, że możliwe jest zaprojektowanie ramowej konstrukcji tego modułu.

4.1. Funkcjonalny model modułu *pół-most*

Aby wykonać zadanie translacji wywołań operacji obiektów sieciowych, moduł *pół-most* musi posiadać zdolność: inicjowania się w obu domenach, rozkodowywania żądania wywołania operacji generowanego przez klienta, stworzenia odpowiedniego żądania w drugiej domenie oraz tłumaczenia obiektów zdefinowanych w modelu CORBA, stanowiących elementy żądania.

Inicjalizacja modułu *pół-most*

Obiekt *pół-most* jest zrealizowany w systemie ORB, w którym działa klient, jako normalny obiekt – serwer tego środowiska. Dlatego może być w nim uruchamiany przy użyciu orginalnych procedur i mechanizmu Adaptera Obiektu (OA) tego ORB-a. Po rozpoczęciu działania musi on także aktywować się w drugim systemie ORB, do którego będzie przekazywał żądania wywołania operacji generowanego przez klienta. Po wykonaniu procedur inicjujących oczekuje on na żądania od klienta.

Przyjmowanie i obsługa wywołania klienta

Żądanie wykonania operacji w systemach zgodnych ze specyfikacją CORBA jest przejmowane przez moduł OA obiektu. W przypadku *pół-mostu* moduł ten używa mechanizmu DSI do dalszej obsługi żądania.

Mechanism DSI został zaproponowany dopiero w specyfikacji CORBA 2, [3] dlatego istniejące systemy nie posiadają jego implementacji. Powoduje to, że dodanie funkcjonalności DSI do modułu *pół-most* staje się istotnym zadaniem w trakcie jego budowy.

Jednym z podstawowych elementów mechanizmu DSI jest obiekt *ServerRequest* [3], który poprzez swój interfejs udostępnia dostęp do nazwy, parametrów i innych informacji związanych z wywołaniem. Aby zbudować ten obiekt, żądanie, które nadeszło od klienta, musi zostać rozpoznane i rozkodowane na podstawie definicji wywołanej operacji zawierającej typy jej parametrów. Definicja ta odczytywana jest z Bazy Injterfejsów środowiska.

Funkcjonalność DSI zawarta jest w metodzie *invoke* obiektu *DynamicImplementation*. Metoda ta w swoim ciele intensywnie wykorzystuje dane dostępne poprzez *ServerRequest*. Jej zadaniem jest podjęcie akcji odpowiednich do rodzaju wywołanej operacji. W przypadku *pół-mostu* buduje ona nowe wywołanie w drugim systemie ORB, tłumacząc wszystkie paramerty żądania do formy obowiązującej w tym środowisku. Następnie wysyła ona to żądanie i oczekuje na jego powrót. W momencie, gdy to nastąpi, tłumaczy ona wszystkie dane powrotne do formy pierwszego ORB i umieszcza je w *ServerRequest*. Mechanizm DSI jest teraz odpowiedzialny za zwrócenie tych rezultatów do klienta.

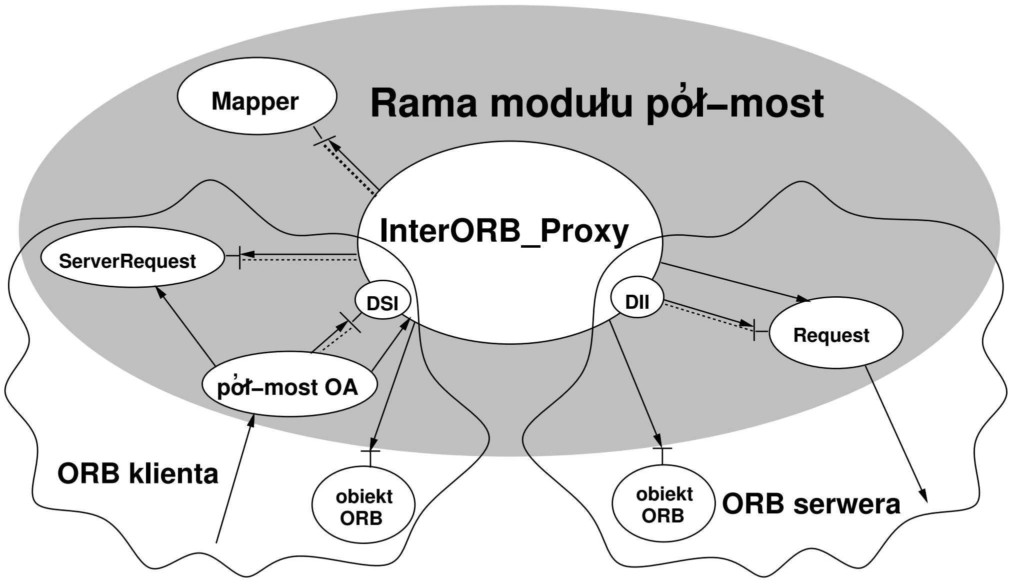
Translacja obiektów modelu CORBA

Jak stwierdzono wcześniej, specyfikacja CORBA 1.2 pozostawiła niezdefinowane pewne części obiektowego środowiska rozproszonego. W rezultacie obiekty jednego systemu nie mogą być w łatwy sposób przekazywane do innego systemu, zbudowanego zgodnie z modelem CORBA. Dlatego bardzo istotnym elementem funkcjonalności modułu *pół-most* jest translacja tych specyficznych obiektów pomiędzy dwoma formatami. Najistotniejszymi obiektami, które należy obsługiwać, są: *ObjectReferences*, *TypeCode*, *Principle*, *Context* i *ServiceContext*. Rozważany moduł *pół-mostu* współpracuje zawsze z domeną IIOP, zatem translacja żądań do formatu tego protokołu i odwrotnie jest niezbędna. Informacje konieczne do wykonania tego zadania mogą być uzyskane w trakcie inicjacji modułu *pół-mostu* lub przy użyciu innych zewnętrznych mechanizmów.

4.2. Ramowa architektura modułu *pół-mostu*

Przedstawiony funkcjonalny model modułu *pół-mostu* pozwolił na zaproponowanie jego ramowej architektury, która może być dostosowywana do potrzeb konkretnego systemu. Jest to tym łatwiejsze, że moduł *pół-mostu* ma jedną stronę całkowicie zdefiowaną, a mianowicie tę, która pracuje w domenie IIOP.

W przedstawionej ramie wyodrębniono te elementy, które są ogólne, to znaczy muszą zostać zrealizowane tylko raz oraz te, które należy budować każdorazowo dla nowego środowiska. Ramową architekturę modułu przedstawiono na rysunku 4.2..



Rys. 2. Architektura ramy modułu współdziałania – *pół-most*

Fig. 2. Architecture of the framework for the interoperability module – half-bridge

Jądro architektury – *InterORB_Proxy*

Obiekt *InterORB_Proxy* stanowi centralny element ramy modułu *pół-most*. Jest to miejsce, gdzie zachodzi połączenie systemu zgodnego z specyfikacją CORBA z domeną IIOP. Obiekt ten używa jedynie ściśle określonych interfejsów specyfikacji CORBA dla tłumaczenia żądania. Posiada on *dynamiczną implementację*, która jest częścią mechanizmu DSI. Jego podstawowa funkcjonalność jest zawarta w metodzie *invoke*, którą dziedziczy z klasy *DynamicImplementation*. Metoda ta tworzy nowe żądania w drugim ORB i używając mechanismu DII (*Dynamic Invocation Interface*) przekazuje je do tego systemu.

InterORB_Proxy został zrealizowany jako wzorzec zparametryzowany poprzez nazwy modułów CORBA należących do sąsiadujących ORB-ów (rysunek 4.2.). Nazwy te są w rzeczywistości stałe w danym module *pół-mostu* i są określane w momencie jego kompilacji. Otrzymany moduł CORBA jest uzupełniany o pewne elementy ze specyfikacji CORBA 2.0, będące niezbędne dla konstrukcji modułu *pół-mostu*, na przykład *ServerRequest*.

```
template <class CORBA_client, class CORBA_server>
class InterORB_Proxy :public virtual INTERORB_PROXY_BASE_IMP,
                      public virtual DynamicImplementation {
  // It is a stringified object reference of a partner representing a remote server
  char * PeerRef;
  // Reference of the Mapper responsible for translating Objects, Typecodes, Principals,
  // Contexts and ServiceContexts from a CORBA_client to CORBA_server representations
  Mapper * my_Mapper;

  public:

  InterORB_Proxy (CORBA_client::ORB *, CORBA_server::ORB *, char *,
                  REF_TYPE ref, Mapper*) :INTERORB_PROXY_BASE_INIT(ref);
  ~InterORB_Proxy ();
  void invoke ( CORBA_client::ServerRequest *&, CORBA_client::Environment &);
};
```

Rys. 3. Implementacja wzorca InterORB_Proxy
Fig. 3. Implementation of InterORB_Proxy tamplate

Warto również zauważyć, że obiekt *InterORB_Proxy* jest bezstanowy. Jego stan nigdy nie ulega zmianie w wyniku wykonania metody *invoke()*. Pozwala to realizować prezentowaną architekturę modułu *pół-most* w wersji jednowątkowej jak i wielowątkowej.

Specificzny *Object Adapter* **obiektu** *pół-mostu*

Stworzenie obiektu *InterORB_Proxy* i początkowa obsługa wywołania przed uruchomieniem metody *invoke()* jest zależna od konkrentego systemu ORB i jest realizowana przez *pół-most OA*. Ten mechanizm musi zostać stworzony jako modyfikacja orginalnego *Object Adapter* lub nawet napisany od początku. Kiedy obce referencje obiektu pojawią się wewnątrz modułu *pół-most*, mechanizm nowego OA musi stworzyć dynamiczną implementację obiektu *InterORB_Proxy*, jeśli taka jeszcze nie istnieje. Parametrami konstruktora tego obiektu są: wskaźniki do obiektów ORB dwóch łączonych systemów i obiektu, który będzie odpowiedzialny za tłumaczenie obiektów modelu CORBA, a także referencja obiektu reprezentowanego przez *InterORB_Proxy* w postaci napisu. Tak stworzony obiekt musi zostać zarejestrowany w orginalnym OA systemu, by żądania skierowane do niego mogły go odnaleźć. Rejestrację w OA umożliwia parametr *ref*, który jest referencją tego *InterORB_Proxy* w danym środowisku ORB. Pojawienie się wywołań skierowanych do obiektu o referencji *ref* powoduje utworzenie instancji klasy *ServerRequest* i przkazanie go do do metody *invoke()*.

Moduł Mapper

Translacja obiektów specyfikacji CORBA, które mogą mieć różną fromę w różnych systemach, stanowi jedną z podstawowych funkcji modułu *pół-most*. Jest to oczywiście zadanie, którego rozwiązanie jest zależne od danej impelemntacji modelu CORBA. Dlatego w prezentowanej architekturze wyodrębniono obiekt – *Mapper*, który jest za to odpowiedzialny. Jego interfejs został ustalony, natomiast jego realizacja musi być wykonywana osobno dla każdego nowego systemu. Podstawowym obiektem, który jest translowany przez ten moduł jest referencja obiektu.

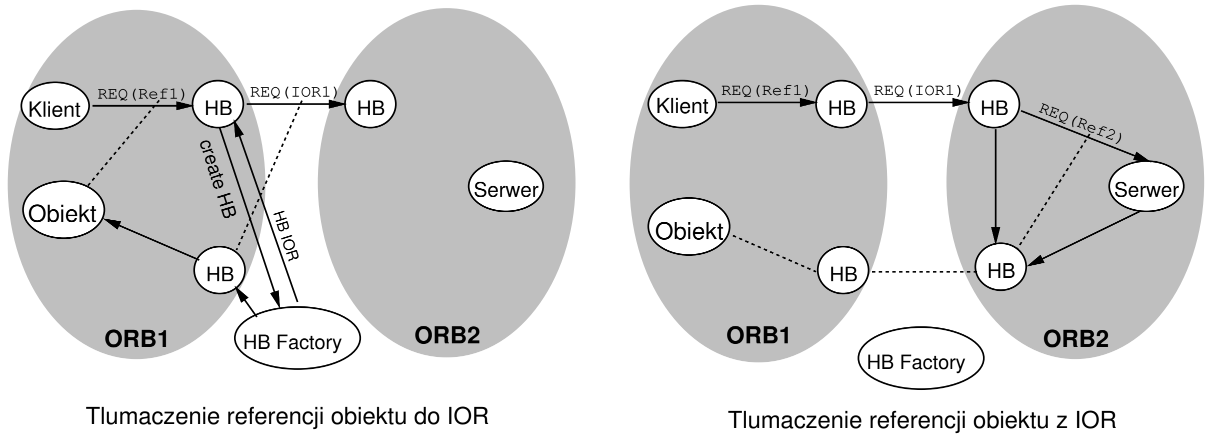
Translacja referencji obiektów

Klient, który wywołuje operację obiektu może umieścić referencje innych obiektów jako parametry tego wywołania. Referencje te wskazują na obiekty sieciowe z jego domeny. Takie referencje nie będą zrozumiałe na zewnątrz tej domeny. Dlatego muszą one być przetranslowane na format obowiązujący w domenie IIOP, to znaczy *Interoperable Object Reference* (IOR) [3]. Co więcej, dla każdej takiej referencji musi zostać stworzony nowy moduł *pół-mostu*, który będzie reprezentował ją w domenie IIOP.

Orginalna referencja obiektu nie jest w procesie translacji zmieniana, lecz zapisywana do pola *object_key* struktury *ProfileBody* referencji IOR. Natomiast polom *host* i *name* tej struktury przypisywane są nazwa komputera i numer portu obiektu z domeny IIOP, który jest w stanie obsłużyć żądanie kierowane do tej referencji. Mogą to być dane dotyczące nowo utworzonego obiektu *pół-mostu* – tryb translacji typu *eager mapping* lub informacje dotyczące obiektu *HB Factory* domenie IIOP – tryb translacji typu *lazy mapping*. Wykorzystanie któregokolwiek z tych dwu trybów translacji referencji jest niezauważalne z funkcyjnego punktu widzenia, lecz wpływa na efektywność przekazywania i tłumaczenia wywołań.

Rozwiązanie typu *eager mapping* (rysunek 4)

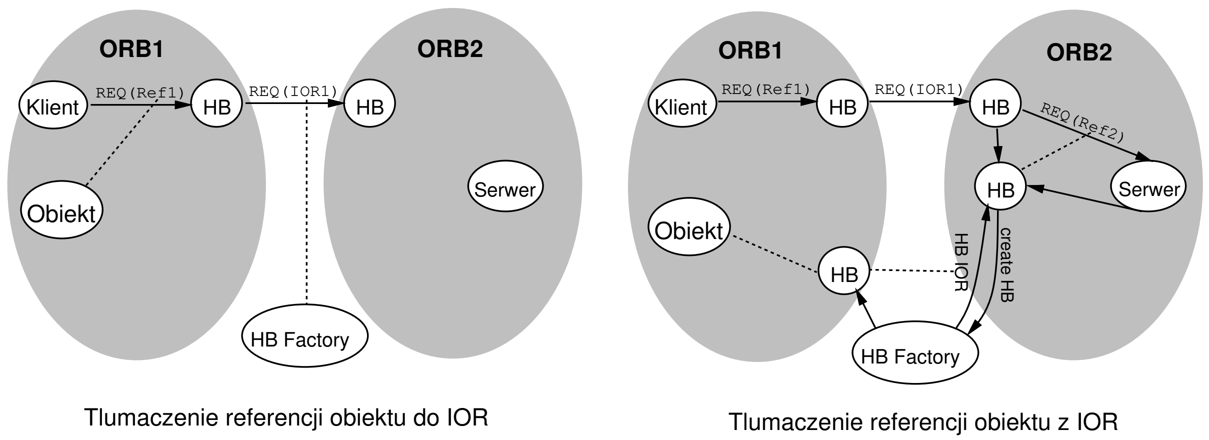
W tym przypadku nowy moduł *pół-mostu*, który udostępni dostęp z domeny IIOP do obiektu, wskazywanego poprzez translowaną referencję, jest tworzony natychmiast przez specjalny obiekt domeny IIOP – *HB Factory*. Przetłumaczona referencja – IOR, zawierająca nazwę komputera i numer portu nowo utworzonego modułu, jest przekazy-



Rys. 4. Translacja referencji obiekty w trybie *eager mapping*

Fig. 4. Object reference translation in the eager mode

wana do *pół-mostu* aktywnego w domenie serwera. Ten moduł tworzy osobny moduł *pół-most* w tym środowisku, który będzie się kontaktował ze swoim partnerem po stronie klienta. Referencja modułu *pół-most* z domeny serwera jest przekazywana do niego w przetłumaczonym wywołaniu. Używając jej *pół-most* może skontaktować się z obiektem w domenie klienta.



Rys. 5. Translacja referencji obiektu w trybie lazy mapping

Fig. 5. Object reference translation in the lazy mode

Rozwiązanie typu *lazy mapping* (rysunek 5)

To rozwiązanie opóźnia tworzenie nowych modułów *pół-mostu* do momentu, kiedy na referencji obiektu zostanie wykonane wywołanie. Nazwa komputera i numer portu związane z obiektem *HB Factory* są używane do wypełnienia struktury IOR w czasie translowania referencji po stronie klienta. Po stronie serwera tworzony jest moduł *pół-most*. W momencie kiedy jest on wywoływany, jako zastępca danego obiektu w tym środowisku, kontaktuje się on z *HB Factory* i prosi o utworzenie partnera po stronie klienta.

5. Realizacja modułu współdziałania dla środowiska Orbix

W tym punkcie opisany zostanie sposób konstrukcji modułu *pół-most* łączącego środowisko *Orbix* ze środowiskiem zbudowanym na bazie protokołu *IIOP*. Pełną współpracę środowiska *Orbix* i domeny IIOP umożliwia istnienie dwóch modułów *pół-most*: *pół-most Orbix-IIOP* i *pół-most IIOP-Orbix*. Moduły te posiadają niemal identyczną implementacę obiektu *Mapper*. Z tego względu obiekt ten zostanie opisany oddzielnie.

5.1. Implementacja modułu *pół-mostu Orbix-IIOP*

InterORB_Proxy jako obiekt systemu *Orbix*

Aby *InterORB_Proxy* mogło być widziane przez elementy środowiska *Orbix* jako implementacja interfejsu wywołanego przez klienta, musi ono zostać w nim zarejestrowane z odpowiednią referencją. Jest to możliwe dzięki zastosowaniu dziedziczenia interfejsu *BOA*. W podejściu tym po skompilowaniu za pomocą kompilatora IDL interfejsu *InterORB_Proxy_base* istnieje klasa *InterORB_Proxy_baseBOAImpl*, której konstruktor dokonuje rejestracji obiektów tego typu jako implementacji interfejsu *InterORB_Proxy_base*. Klasę tę poszerzono o dodatkowy konstruktor, umożliwiający rejestrację obiektu z dowolną nazwą interfejsu, będącą parametrem wywołania konstruktora. Dzięki temu *InterORB_Proxy*, które dziedziczy z *InterORB_Proxy_baseBOAImpl*, może być uznawane za implementację dowolnego interfejsu środowiska *Orbix*.

Dynamiczne tworzenie *InterORB_Proxy*

W celu realizacji ogólnego *pół-mostu*, czyli takiego, który może reprezentować dowolny serwer, konieczne jest dynamiczne tworzenie *InterORB_Proxy* w momencie, gdy klient wykona operację *_bind()* lub po raz pierwszy wywoła operację żądanego interfejsu. *Orbix* umożliwia dynamiczne tworzenie obiektów za pomocą obiektu *Loader*, którego operacja *load()* jest wywoływana, gdy nastąpi wywołanie dotąd nie zarejestrowanego obiektu. Jednym z parametrów wywołania tej operacji jest nazwa żądanego interfejsu. Metoda *load* tworzy obiekt *InterORB_Proxy* rejestrując go jako implementację tego interfejsu.

Obsługa wywołania operacji

W środowisku *Orbix* dla każdego interfejsu zdefiniowanego w języku IDL istnieje obiekt posiadający metodę *dispatch*, która wywoływana jest po rozpoznaniu żądania wywołania operacji i której jako parametr przekazany jest obiekt typu *CORBA::Request*, zawierający wszystkie potrzebne informacje. Metoda ta zwykle odczytuje z tego obiektu nazwę operacji i na jej podstawie ustala odpowiednie wartości argumentów, wywołuje właściwą operację i zapisuje wartości wynikowe. Dla obiektu związanego z *InterORB_Proxy_base* metoda ta została zmodyfikowna w taki sposób, że niezależnie od żądanej operacji wywoływana jest funkcja *InterORB_Proxy::invoke*. Wartości argumentów tej operacji przekazane są wraz z całym obiektem *CORBA::Request* wewnątrz jej parametru typu *CORBA2::ServerRequest*.

Konstrukcja obiektu *CORBA2::ServerRequest*

Klasa *ServerRequest* wraz z jej funkcjonalnością stanowi część mudułu DSI, będącego podstawowym uzupełnieniem specyfikacji CORBA 2 w stosunku do CORBA 1.2.

Definicja klasy *ServerRequest* została umieszczona w klasie CORBA 2, dziedziczącej z klasy CORBA. W ten sposób dokonano adaptacji środowiska *Orbix* do wymagań stawianych przez *InterORB_Proxy*. Podstawową metodą obiektu tej klasy jest operacja *params()*, umożliwiająca odczyt wartości argumentów wywoływanej operacji zakodowanych wewnątrz obiektu *CORBA::Request*. W tym celu wykorzystywany jest strumieniowy interfejs tego obiektu. Dekodowanie parametrów odbywa się na podstawie ich definicji typów reprezentowanych w tym środowisku przez napisy.

5.2. Implementacja modułu *pół-most IIOP-Orbix*

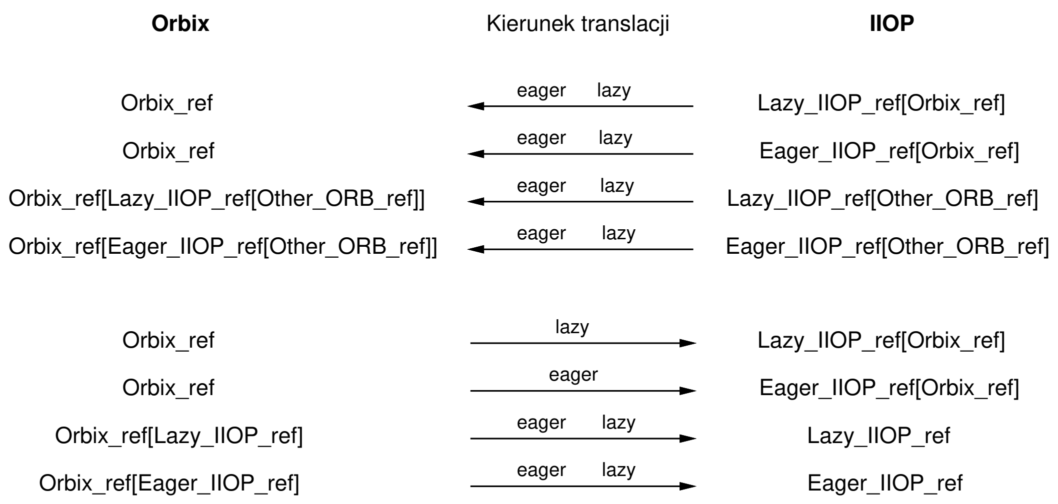
Jako implementację protokołu IIOP wykorzystano oprogramowanie firmy SunSoft – *Inter-ORB_Engine*. W miejscach, gdzie interfejsy tego środowiska lub ich funkcjonalność nie jest zgodna ze specyfikacją CORBA 2, zostały one rozszerzone lub zmodyfikowane. *InterORB_Proxy_base*, stanowiące klasę bazową dla *InterORB_Proxy*, dziedziczy z klasy *CORBA2::Object* tego systemu i może być rejestrowane z dowolną referencją przekazaną jako parametr konstruktora. *InterORB_Proxy_base* posiada jedną metodę – *my_skeleton*, która wywołuje operację *InterORB_Proxy::invoke()*. Obiekt *InterORB_Proxy* tworzony jest na początku działania mudułu *pół-most IIOP-Orbix* i posiada referencję przekazaną jako argument wywołania procesu. W środowisku *Inter-ORB_Engine* istnieje obiekt *ServerRequest* i jest on tworzony poprzez niższe warstwy systemu bezpośrednio po rozpoznaniu wywołania operacji. Również system troszczy się o przesłanie rezultatów umieszczonych wewnątrz tego obiektu do klienta. Funkcjonalności te wraz z implementacją operacji *invoke()* stanowią pełny moduł *DSI* w środowisku *Inter-ORB_Engine*.

5.3. Implementacja obiektu *Mapper*

Tranaslacja obiektów specyfikacji CORBA

W celu umożliwienia pełnej współpracy środowisk *Orbix* i *Inter-ORB_Engine* konieczne jest dokonanie translacji obiektów, takich jak: *Context*, *Principal* i *TypeCode*. O ile dla dwóch pierwszych obiektów polega ona na użyciu specyficznych dla danego środowiska metod tych obiektów w celu odczytu i zapisu ich danych, o tyle translacja obiektu *TypeCode* wymaga dokonania tłumaczenia reprezentacji napisowej w środowisku *Orbix* na reprezentacje w domenie IIOP – sekwencję oktetów i odwrotnie. Dopiero po stworzeniu odpowiedniej dla danego środowiska reprezentacji możliwe jest powstanie właściwego obiektu. Dla przeprowadzenia tej opercji konieczne jest zatem dogłębne rozpoznanie reprezentacji tego obiektu w obu środowiskach.

Translacja referencji obiektów



Rys. 6. Reguły translacji referencji w modułach *pół-most Orbix-IIOP* i *pół-most IIOP-Orbix*

Fig. 6. Rules of object reference translation in Orbix-IIOP and IIOP-Orbix half-bridges

Translacja referencji obiektów między dwoma różnymi środowiskami łączy się z koniecznością utworzenia nowych modułów *pół-mostu* reprezentujących te referencje. To, kiedy te moduły powstaną, zależy od strategii przyjętej przez obiekt *Mapper*: albo powstaną one jednocześnie z dokonaniem translacji (translacja *eager*), albo dopiero przy pierwszym wywołaniu na nich operacji (translacja *lazy*). W przypadku istnienia większej ilości domen zadanie to komplikuje się ze względu na konieczność zachowania warunku, że klient i serwer w różnych środowiskach CORBA powinny komunikować się przez co najwyżej dwa moduły *pół-mosty*. Z tego względu konieczna jest głębsza analiza referencji – obiekt *Mapper* musi wykryć sytuację, w której otrzymana przez niego referencja stanowi translację referencji pochodzącej z jakiegoś innego środowiska. W tym celu wykorzystuje się konwencję dotyczącą budowy klucza obiektu, co zostanie wyjaśnione poniżej.

Referencja *Orbix*-a składa się z 6 części:

NAZWA WĘZŁA SERWERA : NAZWA SERWERA :
IDENTYFIKATOR OBIEKTU : NAZWA WĘZŁA SERWERA INTERFACE REPOSITORY :
NAZWA SERWERA INTERFACE REPOSITORY : NAZWA INTERFEJSU

Referencja w protokole IIOP zbudowana jest w następujący sposób:

NAZWA PROTOKOŁU = iiop : WERSJA IIOP = 1.0 /
NAZWA WĘZŁA SERWERA : NUMER PORTU SERWERA // KLUCZ OBIEKTU

przy czym referencja IIOP typu *lazy* różni się tym od referencji typu *eager*, że w polu *KLUCZ OBIEKTU* rzeczywisty klucz obiektu, do którego odnosi się referencja, poprzedzony jest kluczem Half-bridge Factory - "IIOP_Factory".

W ogólnym przypadku translacji referencji z formatu *IOR* do *Orbix*a przyjęto zasadę enkapsulacji referencji *IOR* w referencji *Orbix*-a w polu *IDENTYFIKATOR OBIEKTU*. W przypadku translacji referencji do formatu protokołu IIOP referencja ta umieszczana jest w polu *KLUCZ OBIEKTU* referencji *IOR* oraz poprzedzana nazwą domeny, z której pochodzi. Rysunek 5 przedstawia sposób translacji referencji w przypadku środowisk *Orbix* i *Inter-ORB_Engine*.

6. Wnioski

Zagadnienie konstrukcji elementów architektury współdziałania systemów typu CORBA wymaga rozwiązania szeregu nowych problemów technicznych. Do najważniejszych należą: konstrukcja modułu DSI, zapewnienie dostępu do Repozytorium Interfejsów, tryb i technika translacji referencji oraz konwersja opisów typów danych. Bardzo wymagające z punktu widzenia techniki programowania jest napisanie procedur serializacji i deserializacji struktur danych. Przeprowadzone prace [7, 8] wykazały, że w celu zapewnienia działania modułów typu *pół-most* konieczne jest wzbogacenie architektury proponowanej przez UNO [4] o takie elementy, jak *HB-Factory*, Repositorium Usług oraz dostąp do Repozytorium Interfejsów. Rozwiniącie tych zagadnień stanowi osobny problem badawczy.

LITERATURA

[1] *CORBA 1.2 Revision Draft*, OMG Report 93-12-43, Object Management Group (OMG) Inc, 1993.

[2] *ORB Interoperability. Joint SunSoft / Iona Submission to the ORB 2.0 Task Force Initialization & Interoperability Request for Proposals*, OMG Inc., TC Document 94-3-1, 1994.

[3] *Universal Networked Objects*, OMG Inc., TC Document 94-9-32, 1994.

[4] *Interface Repository*, OMG Inc., TC Document 94-11-7, 1994.

[5] *ORB Initialization Specification*, OMG Inc., TC Document 94-9-46, 1995.

[6] Uszok A., Czajkowski G., Zieliński K., *Interoperability Gateway Construction for Object Oriented Distributed Systems*, Proceedings of 6th Nordic Workshop on Programming Environment Research, Lund, Szwecja, 1994.

[7] Steinder, M., Uszok A., Zieliński K.: *A Framework for Inter-ORB Request Level Bridge Construction*, Proceedings of IFIP/IEEE International Conference on Distributed Platforms ICDCP'96, Drezno, Niemcy, Chapman & Hall, 1996.

[8] *Orbix Programmers Guide*, IONA Technologies Ltd., 1995.

Recenzent: Dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 21 grudnia 1995 r.

Abstract

Modern object-oriented systems are mostly built according to the CORBA 1.2 specification, which was adopted by the Object Management Group in 1993. Many implementations of this model occurred, e.g. Iona's Orbix, IMB's SOM, PostModern's ORBeline or Digital's ObjectBroker. However, because of lacks in the CORBA 1.2 specification their objects are not able to interoperate, what was the goal of the OMG. This paper addresses a problem of building a *generic* half-bridge between different CORBA compliant systems. It presents a framework of such a module based on the UNO approach whose architecture is easily extendable to more sophisticated in parallelizing level and functionality units. A problem of mapping objects defined in CORBA model is described. The detailed ObjectReferences mapping is presented. As a case study implementation of a half-bridge for Orbix is described.