STUDIA INFORMATICA 2002
Volume 23 Number 2B (49)

Tomasz GRATKOWSKI
Uniwersytet Zielonogórski, Instytut Informatyki i Elektroniki

ZDALNE WYWOŁYWANIE METOD W JĘZYKU JAVA Z WYKORZYSTANIEM PROTOKOŁU FAST LOCAL INTERNET PROTOCOL (FLIP)

Streszczenie. Artykuł przedstawia sposób wykorzystania protokołu FLIP w celu zaimplementowania rozproszonego środowiska programowania dla języka Java. Proponowane rozwiązanie nie wymaga wykorzystywania łączników jako mechanizmu przezroczystości położenia obiektów. Odnajdywanie procesów udostępniających zdalne obiekty wykonywane będzie na poziomie protokołu sieciowego.

REMOTE METHOD INVOCATION IN JAVA USING FAST LOCAL INTERNET PROTOCOL (FLIP)

Summary. This article presents a method of using FLIP protocol to create a distributed programming platform for Java language. In the present solution the binders to provide transparency of location objects are not used. Processes that make objects available, are searched by means of network protocol.

1. Wprowadzenie

Proces projektowania środowiska do implementowania aplikacji rozproszonych wymusza na projektantach opracowanie kilku podstawowych elementów [7]. Implementowane składowe mają zapewnić użytkownikom prostotę i wygodę w użytkowaniu środowiska rozproszonego, przy jednoczesnym zachowaniu dużej funkcjonalności. Jednym z elementów jest mechanizm dostępu do zdalnych zasobów [1], którego główną właściwością braną pod uwagę jest przezroczystość położenia. W znacznym stopniu ułatwia ona wykorzystanie rozproszonej aplikacji, zwalniając programistę i użytkowników z obowiązku znajomości położenia wymaganych zasobów. Dodatkowo, w pełni zaimplementowana przezroczystość położenia

umożliwia dalszy wzrost funkcjonalności projektowanego systemu o kolejne właściwości, takie jak: przezroczystość wędrówki, przezroczystość zwielokrotnienia, przezroczystość wydajności, przezroczystość awarii oraz przezroczystość dostępu.

1.1. Serwery nazw

Najpowszechniej stosowanym sposobem uzyskania przezroczystości położenia zasobów jest użycie serwerów nazw (ang. *Name Servers*). Serwery nazw umożliwiają przechowywanie informacji o udostępnionym zasobie (np. informację o położeniu zasobu). Jednak usługa serwerów nazw sama nie jest usługą przezroczystą, ponieważ zarówno użytkownik rejestrujący informacje o zasobie, jak i szukający informacji, muszą znać dokładnie położenie serwera nazw. Dodatkowo serwery nazw wymagają od użytkowników specjalnych metod w celu rejestracji i wyrejestrowywania obiektów oraz nie kontrolują stanu przechowywanych zasobów[1]. W sytuacji gdy zasobem jest udostępniony przez proces obiekt i gdy przed usunięciem obiektu nie zostanie odłączony wpis w serwerze nazw, przechowywane informacje będą błędne.

1.2. Protokoły rozgłaszające

Innym rozwiązaniem problemu braku przezroczystości położenia jest mechanizm rozgłaszania przy użyciu protokołów sieciowych. Jednak rozgłaszające protokoły sieciowe mają pewne ograniczenia: stosowane bez umiaru powodują zmniejszenie przepustowości sieci, są przeznaczone głównie dla rozwiązań działających w obrębie jednej podsieci. Istnieją protokoły sieciowe używane pomiędzy ruterami i stacjami (IGMP), które pozwalają rozszerzyć pole działania multicastingu opartego np. na protokole UDP/IP. Protokół UDP/IP posiada jednak mechanizm identyfikacji zasobu w sieci poprzez dwa elementy: nazwę komputera (adres IP) oraz numer portu, na którym usługa została udostępniona. Rozwiązanie to mocno ogranicza zastosowanie protokołu przy implementowaniu rozwiązań udostępniających przezroczystość położenia zasobów. Pierwszym powodem ograniczenia jest zastosowanie usługi nazewniczej, która w nazwie zasobu podaje jego fizyczne położenie. Drugi powód wynika bezpośrednio z pierwszego, brak jest prostej możliwości zmiany położenia zasobu bez zmiany nazwy komputera oraz numeru portu.

Protokół Fast Local Internet Protocol (FLIP) [2] został zaprojektowany dla systemu rozproszonego Amoeba [2, 8]. FLIP jest protokołem bezpołączeniowym warstwy trzeciej według modelu OSI (rys. 1). Podstawową cechą protokołu FLIP jest przezroczystość położenia udostępnianego zasobu. Jest ona możliwa poprzez zastosowanie 64-bitowego identyfikatora, który nie identyfikuje fizycznego położenia zasobu i jest wykorzystywany do realizacji tra-

[1] W rozwiązaniach RMI i CORBA.

sowania sieci. Każdy z węzłów sieci posiada własną tabelę trasowania, którą uzupełnia i aktualizuje w czasie pracy węzła.

	OSI	TCP/IP	FLIP
7	warstwa aplikacji	warstwa aplikacji	warstwa aplikacji
6	warstwa prezentacji		
5	warstwa sesji		RPC / komunikacja grupowa
4	warstwa transportu	warstwa transportu (TCP | UDP	niepotrzebna
3	warstwa sieci	warstwa sieci (IP)	FLIP
2	warstwa łącza danych	warstwa łącza (np. Ethernet)	warstwa łącza (np. Ethernet)
1	warstwa fizyczna	media	media

Rys. 1. Funkcje warstw w OSI, TCP/IP i FLIP
Fig. 1. Layers of functionality in OSI, TCP and FLIP

Tabele trasowania w znacznym stopniu ograniczają rozgłaszanie, które potrzebne jest tylko w sytuacji nieznajomości położenia zasobu lub gdy szukany zasób zmienił miejsce położenia. Dodatkowo w protokole używane są dwa rodzaje identyfikatorów zwiększających bezpieczeństwo, prywatny i publiczny. Identyfikator prywatny jest używany wewnątrz węzła do identyfikacji konkretnego procesu, a identyfikator publiczny jest używany do identyfikacji procesu w sieci. Ponieważ zastosowano tylko jednokierunkową funkcję szyfrowania, węzły, które znają tylko identyfikator publiczny, nie mogą odebrać wiadomości, gdy nie znają identyfikatora prywatnego.

2. Wykorzystanie protokołu FLIP do zdalnego wywoływania metod języka Java (FLIP-RMI)

Przezroczystość położenia procesów uzyskana poprzez protokół FLIP w połączeniu z językiem obiektowym Java daje możliwość zbudowania środowiska rozproszonych obiektów bez potrzeby używania serwerów nazw. Dzięki mechanizmom protokołu FLIP: przechowywaniu tablic trasowania, odszukiwaniu położenia nieznanych procesów oraz ponownym odszukiwaniu przeniesionych procesów, proponowane przez autora rozwiązanie wykorzystujące protokół FLIP do zdalnego wywoływania metod języka Java (nazywane dalej FLIP-RMI), w sposób wydajny i dynamiczny kontroluje udostępniane poprzez procesy obiekty.

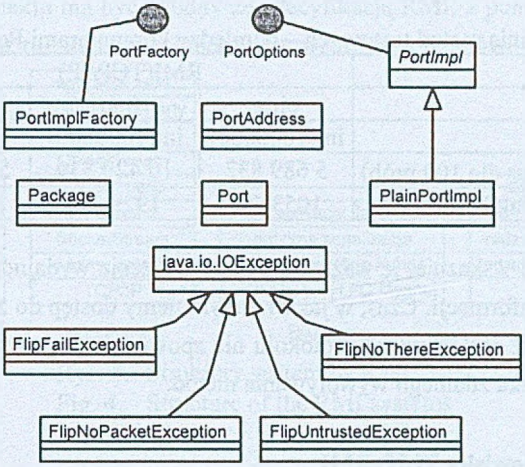
2.1. Obsługa protokołu FLIP w języku Java (jFLIP)

Wraz z wersją Amoeba 5.3 dostępne są kody źródłowe sterowników protokołu FLIP dla systemu Unix Sun Solaris 5.x. W pierwszym etapie realizacji środowiska FLIP-RMI, w ramach opracowanego przez autora pakietu jFLIP języka Java, zaimplementowano procedurę dostępu do sterowników FLIP w języku Java. Implementacja jFLIP wymagała integracji bezpośrednio w kod źródłowy dostarczonych sterowników FLIP, ponieważ pominięto kodowanie identyfikatora prywatnego. Wyeliminowano funkcję szyfrowania, aby umożliwić udostępniającemu zasoby nadawanie im niepowtarzalnej i zrozumiałej nazwy. Nazwa ta powinna być widoczna jako identyfikator publiczny w celu łatwej identyfikacji zasobu. Możliwość nadawania nazw zasobom ogranicza 64-bitowy identyfikator. Wynika to z przeznaczonej przestrzeni w ramce protokołu FLIP dla identyfikatorów odbiorcy i nadawcy komunikatu. Dlatego ograniczono możliwość użycia znaków opisujących zasób do zbioru:

IdentyfikatorZasobu ::= ([a–zA–Z0–9] | '.' | '_')+ (1)

Zaproponowany zbiór 64 znaków umożliwia zakodowanie przy użyciu 6 bitów pojedynczego znaku, co daje możliwość użycia maksymalnie 10 znaków. Wprowadzono dwa znaki przestankowe '.' oraz '_', co daje możliwość logicznego szeregowania zasobów. Dodatkowo metoda rejestrująca zasób rozszerzona została o operację szukania, czy zastosowana dla zasobu nazwa nie jest używana w systemie.

Rodzimy interfejs języka Java (Java Native Interface - JNI) umożliwia implementowanie rozwiązań niezależnych od platformy, dedykowanych dla języka Java, udostępniając funkcje aplikacji i bibliotek napisanych w innych językach. Autor przy użyciu interfejsu JNI zaimplementował pakiet jFLIP. Utworzony został interfejs PortOptions, deklarujący wszystkie wymagane finalne opcje protokołu FLIP. Abstrakcyjna klasa PortImpl implementuje interfejs PortOptions, deklarując pozostałe wymagane opcje oraz sposób wywołania funkcji protokołu FLIP. Klasa konkretna PlantPortImpl, która dziedziczy po klasie PortImpl, wywołuje rodzime metody protokołu FLIP poprzez interfejs JNI. Klasa Package służy do utworzenia pakietu wysyłanych i odbieranych danych. W celu kontroli poprawności nadawania nazwy zasobom utworzono klasę PortAddress. Do utworzenia NSAP (Network Service Access Point) [2] wykorzystywana jest klasa Port. W proponowanym rozwiązaniu utworzono, znany z języka Java, mechanizm tworzenia wytwórni klas, udostępniając interfejs PortFactory oraz klasę PortImplFactory. Omówione klasy oraz dodatkowe klasy do obsługi błędów przedstawiono na diagramie klas (rys. 2).



Rys. 2. Diagram klas do obsługi protokołu FLIP
Fig. 2. Class diagram for FLIP protocol

2.2. Testy wydajności

Na obecnym etapie implementacji prezentowanego projektu przeprowadzono testy na dwóch komputerach o architekturze SPARC: Sun Station 10 (nazwa Cookie) i Sun Ultra 80 (nazwa Porto). Testy, z wykorzystaniem osobnej podsieci Fast Ethernet i komputerów wyłączonych z ogólnego użytku, objęły pomiar czasu wykonania metody na zdalnym obiekcie RMI oraz czasu pobrania informacji o obiekcie przy użyciu pakietu jFLIP. Miały one na celu wykazanie, czy wykorzystanie jFLIP nie będzie znacząco wpływało na zmniejszenie wydajności projektowanego systemu FLIP-RMI. Wybrano trzy metody: ping – bezparametrowa, ping – przekazująca parametr w kierunku do zdalnego obiektu, start – przekazująca parametr w kierunku do i z zdalnego obiektu. W tabeli 1 przedstawiono wyniki, jakie zostały osiągnięte, kiedy wszystkie elementy testu znajdowały się na pojedynczym komputerze.

Tabela 1

Czas wykonania metod testowych – lokalnie na komputerze Porto

Protokół	RMI(TCP/IP)			jFLIP
Metoda	ping	start	ping	ping
Przesłane dane	int[1000000]	int[1000000]		
Czas [μs] (średni czas dla 100 prób)	994 352	1 624 000	464 515	824
Czas metody/czas ping RMI	214%	350%	100%	0,18%

W tabeli 2 przedstawiono wyniki, jakie zostały osiągnięte, kiedy obiekt znajdował się na komputerze Cookie, a klient wysyłał żądanie z komputera Porto.

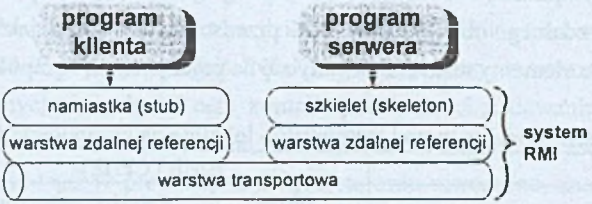
Tabela 2
Czas wykonania metod testowych – pomiędzy komputerami Porto i Cookie

Protokół	RMI(TCP/IP)			jFLIP
Metoda	ping	Start	ping	ping
Przesłane dane	int[1000000]	int[1000000]		
Czas [μs] (średni czas dla 100 prób)	5 689 832	10 420 339	540 087	2 330
Czas metody/czas ping RMI	1053,5%	1929,4%	100,0%	0,50%

Otrzymane wyniki wskazują, iż ważny, z punktu widzenia wydajności, jest sposób serializacji przesyłanych informacji. Czas, w jakim otrzymujemy dostęp do zasobu poprzez protokół FLIP, wskazuje, iż zastosowanie protokołu nie spowoduje pogorszenia wydajności projektowanego środowiska zdalnego wywoływania metod.

2.3. Projekt środowiska FLIP-RMI

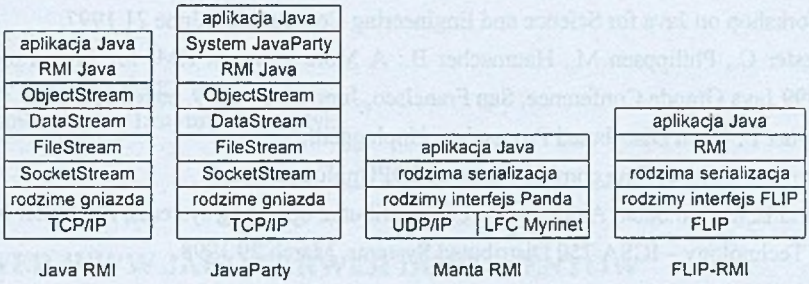
Pakiet RMI (Remote Method Invocation) języka Java umożliwia udostępnianie zdalnych obiektów i wywoływanie na obiektach metod. Architektura RMI oparta jest na warstwach (rys. 3), tak aby mogła być w łatwy sposób modyfikowana. Kolejnym krokiem proponowanego rozwiązania będzie wykorzystanie zaimplementowanego jFLIP i dodanie potrzebnych mechanizmów w warstwie transportowej RMI, aby udostępnić zdalne wywołanie metod na obiektach poprzez protokół FLIP. Na poziomie warstwy transportowej RMI musi być oprogramowana kontrola przesyłanych komunikatów pomiędzy obiektami, tak aby zgodnie ze specyfikacją RMI, była zachowana semantyka co najwyżej jednokrotnego wywołania (ang. *at most once*). Implementacja będzie również wymagała zmiany mechanizmu rejestracji obiektu, tworzony będzie proces udostępniający obiekt. Dodatkowo implementacji wymaga obsługa mechanizmu wędrówki obiektów.



Rys. 3. Architektura warstwowa RMI
Fig. 3. RMI layered architecture

W wielu projektach wykorzystano warstwowy model w celu poprawy właściwości zdalnego wywoływania metod w języku Java. Rys. 4 przedstawia architekturę zastosowaną w projektach RMI, JavaParty [5], Manta [6] oraz w FLIP-RMI. Jednak w JavaParty oraz Manta ingerencji uległ interfejs udostępniony programiście. W proponowanym rozwiązaniu

mechanizm użycia obiektu ma być zgodny ze specyfikacją RMI, z pominięciem znajomości położenia łącznika oraz rozszerzony o mechanizm wędrówki obiektów.



Rys. 4. Struktury systemów RMI
Fig. 4. Structure of the RMI systems

3. Wnioski

Testy wydajności protokołu FLIP wykazały zadowalającą szybkość odpowiedzi zdalnego obiektu, przy zachowaniu dużej skalowalności. Z prac [5, 6] wynika, iż elementem spowalniającym działanie RMI jest zastosowana serializacja. Dlatego przy implementowaniu FLIP-RMI będą wykorzystane opisane w pracach [5, 6] uwagi. Przy zachowaniu zadowalającej wydajności i dodatkowych cechach: przezroczystości położenia obiektów, przezroczystości wędrówki, komunikacji grupowej oraz zachowania stosowanej w RMI semantyki, rozwiązanie wydaje się interesującym środowiskiem do budowy aplikacji rozproszonych.

LITERATURA

1. Coulouris G., Dollimore J., Kindberg T.: Systemy rozproszone. Podstawy i projektowanie. Wydawnictwa Naukowo-Techniczne, Warszawa 1998.
2. Kaashoek M. F., van Renesse R., van Staveren H., Tanenbaum A. S.: FLIP: an Internetwork Protocol for Supporting Distributed Systems, ACM Trans. Comp. Syst. February 1993, pages 73-106.
3. Java™ Remote Method Invocation Specification, Revision 1.7, Java™ 2 SDK, Standard Edition, v1.3.0, December 1999.
4. The Common Object Request Broker: Architecture and Specification, Editorial Revision: CORBA 2.4.2: February 2001.

5. Philippsen M., Zenger M.: JavaParty – Transparent Remote Object in Java, In ACM 1997 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP Workshop on Java for Science and Engineering Computation, June 21 1997.
6. Nester C., Philippsen M., Haumacher B.: A More Efficient RMI for Java, Proc. ACM 1999 Java Grande Conference, San Francisco, June 12-14 1999, pages 152-159.
7. Joyner I.: Open Distributed Processing: Unplugged!, http://homepages.ihug.com.au/~ijoyner/ODPUnplugged.html.
8. Lund E.W.: Amoeba: An overview of a distributed operating systems, Rochester Institute of Technology – ICSA 750 Distributed Systems, March 29 1998.

Recenzent: Dr inż. Przemysław Szmal

Wpłynęło do Redakcji 2 kwietnia 2002 r.

Abstract

Programming language Java makes it possible to implement distributed applications through RMI (Remote Method Invocation) [3] and CORBA (Common Object Request Broker Architecture) [4]. Both solutions use binders, to search objects or to associate unique identifiers with the objects for whole implemented distributed system. Using a binder forces programmers to use operations of binding, unbinding and looking up objects. Therefore localization of the binder must be known for all users or must be searched by broadcasting. Additionally every change of object location must be noted in the binder.

Protocol FLIP (Fast Local Internet Protocol) [2] has built-in position transparency. Protocol FLIP uses for identification of processes a unique port number for whole distributed system. This port number does not determine physical position of a process. Therefore processes can be moved between computers in a network without changing process port number.

This paper presents a way of using FLIP protocol to create a distributed programming platform for Java language. This solution does not use binders, processes are searched by means of protocol FLIP. Processes are the remote objects. This solution presents Java package for handling interconnections between Java and FLIP protocol (Fig. 2). Using of this package enables creating middleware for Java shown at Fig. 4. This middleware has an RMI interface, transparent position, but doesn't have a binder. Basing on the experience described in papers [5, 6], in this middleware a native serialization will be used, which accelerates using RMI for building distributed applications.