STUDIA INFORMATICA 2001
Volume 22 Number 1 (43)

Tomasz FOLGA, Henryk KRAWCZYK
Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki

ŚRODOWISKA AGENTOWE JAKO PLATFORMY BUDOWY APLIKACJI ROZPROSZONYCH

Streszczenie. Przedstawiono istniejące standardy oraz opisano środowiska mobilnych agentów. Porównano funkcjonalność i użyteczność tych środowisk oraz możliwości tworzenia różnego typu aplikacji rozproszonych..

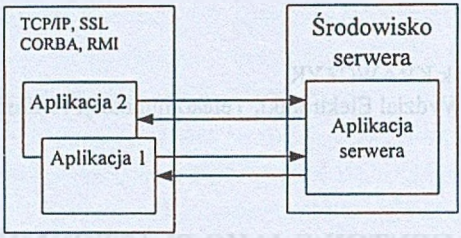
AGENT ORIENTED ENVIRONMENTS AS DESIGN PLATFORMS OF DISTRIBUTED APPLICATIONS

Summary. The paper presents existing standards and describes computing environments with mobile agents, Moreover their functionality and usability is compared and possibility of creation of different type of distributed applications in such environments are discussed.

1. Wprowadzenie

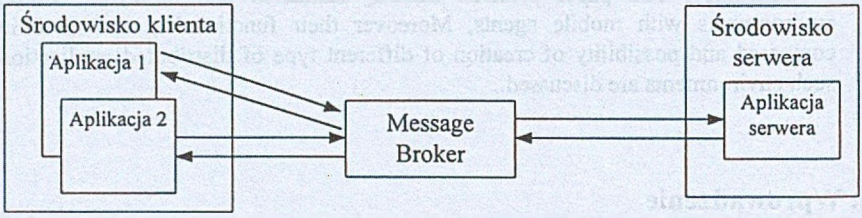
Od czasu gwałtownego rozwoju sieci komputerowych, zwieńczonego powstaniem ogólnoświatowej sieci komputerowej Internet, zaczęły powstawać aplikacje i systemy informatyczne wykorzystujące zalety takiej globalnej sieci. Przyglądając się architekturze różnych systemów informatycznych wykorzystujących komunikację sieciową można zauważyć, że większość z nich opiera swe funkcjonowanie na modelu klient-serwer (rys. 1), gdzie wykorzystuje się mechanizm RPC (*Remote Procedure Call*), opisujący komunikację komputer-komputer jako możliwość wywołania procedur na innym komputerze. Każda wiadomość przesyłana przez sieć jest albo żądaniem wywołania procedury wraz z określonymi parametrami albo potwierdzeniem jej wykonania wraz z danymi przesyłanymi jako wynik wywołania. Przed dokonaniem zdalnego wywołania należy ustalić tzw. *protokół*,

który jest zgodą obydwu stron na zakres działania i rodzaj przekazywanych parametrów (oraz wyników) zdalnych procedur.



Rys. 1. Klasyczny model klient-serwer
Fig. 1. Classical client-server model

Jedną z wad takiego modelu współpracy aplikacji sieciowej jest konieczność aktywnego udziału obu stron w komunikacji. W odróżnieniu od synchronicznej komunikacji RPC istnieją inne rozwiązania typu *Messaging-Oriented Middleware (MOM)* (rys. 2) lub bardziej ogólnie modele klient-pośrednik-serwer, które polegają na wysłaniu asynchronicznych komunikatów do kolejki komunikatów (tzw. *message brokers*), które działają na zasadach kolejek komunikatów. Model ten jest przydatny, gdy jedna z komunikujących się stron posiada niepewne połączenie sieciowe bądź gdy komunikują się ze sobą urządzenia przenośne: *personal digital assistants (PDA's)*.

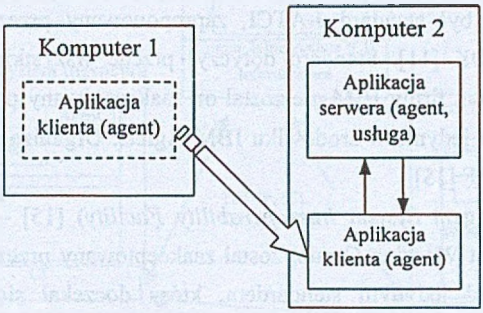


Rys. 2. Model klient-pośrednik-serwer
Fig. 2. Client-broker-server model

Dalszym udoskonaleniem komunikacji jest niezależność od architektury sprzętowej i języka programowania, czego przykładem może być *Common Object Request Broker Architecture (CORBA)*. Wykorzystując technologię obiektową każdą aplikację można traktować jako zbiór rozproszonych obiektów. Co więcej, w tym przypadku model klient-pośrednik-serwer umożliwia jednakowe operowanie na obiektach bez względu na to, na którym komputerze one znajdują się.

Model środowiska agentowego (rys. 3) bazuje na paradygmacie mobilnych agentów, które rozumiane są jako 'inteligentne' obiekty, mogą się przemieszczać pomiędzy

komputerami w celu wykonania określonego zadania. Model ten swoją funkcjonalnością obejmuje wymienione wcześniej modele, ponieważ agenty mogą zachowywać się jak rozproszone (stacjonarne) obiekty CORBY oraz mogą wywoływać zdalne procedury (RPC). Podstawową różnicą w stosunku do poprzednich modeli jest fakt, że agenty mogą przemieszczać się z jednego komuputera na drugi i wykonywać tam określone funkcje, wykorzystując przy tym udostępniane usługi.



Rys. 3. Model środowiska agentowego
Fig. 3. Agent environment model

W celu wykorzystania modelu agentowego buduje się odpowiednie środowiska, umożliwiające działanie agentów. Zmienia to całkowicie sposób wykonania aplikacji. Poza tym wykonanie całej aplikacji odbywało się na wydzielonym superkomputerze, który miał wykonywać wiele zadań różnych użytkowników. Klient wiedział jedynie, jaką procedurę wywołać, natomiast serwer dostarczał zasobów, czasu procesora oraz logiki wykonania (*know-how*). Obecnie w dobie coraz szybszych komputerów domowych zaciera się różnica pomiędzy superszybkim serwerem i terminalem. Dobrze znany paradygmat klient/serwer nie sprawdza się już tak dobrze jak poprzednio. Właśnie model mobilnych agentów daje dużo większą elastyczność wyboru, kto powinien dostarczyć zasobów obliczeniowych oraz logikę aplikacji. Wysyłając agenta na określony komputer dostarczamy tam logikę wykonania naszego zadania, jednocześnie korzystamy z jego zasobów. Zaciera się różnica pomiędzy rolami, jakie pełnią strony w modelu klient-serwer, zaś połączone w ten sposób komputery tworzą grupę równoprawnych komputerów.

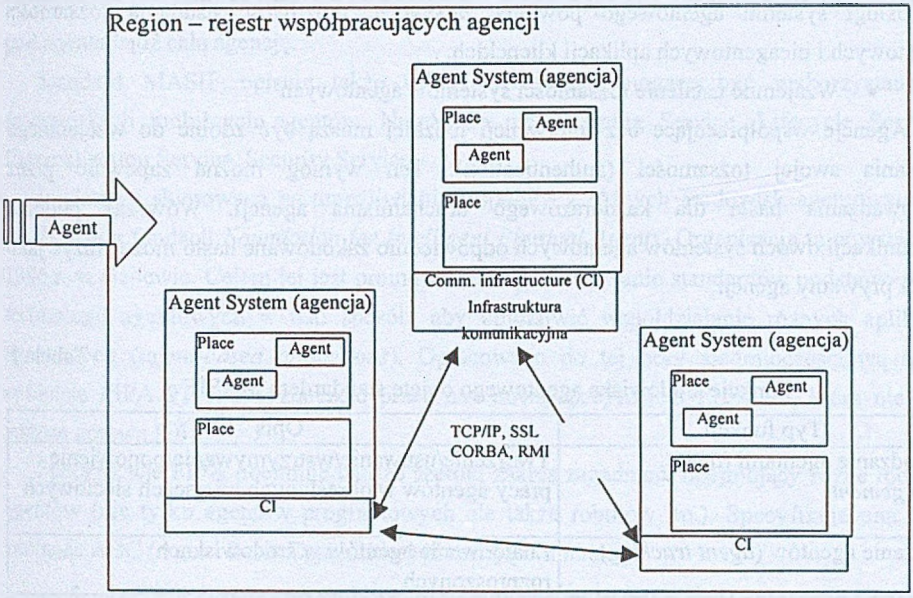
W pracy przedstawiono podstawowe standardy związane ze środowiskiem agentowym oraz dokonano przeglądu istniejących środowisk agentowych. Zaprezentowano możliwości ich wykorzystania do budowy różnych aplikacji rozproszonych.

2. Standard MASIF i FIPA

Różne implementacje modelu klient-agent-serwer lub inaczej środowisk agentowych wymagały opracowania standardów współpracy. Standard miał także pomóc twórcom środowisk rozwiązać podstawowe problemy związane z zarządzaniem, transportem i nazewnictwem agentów. Pierwszym standardem, który obejmował swym zakresem technologie agentów, był standard J-ATCI, zaproponowany przez firmę IBM – twórcę środowiska Aglets SDK [11]. Standard dotyczył przede wszystkim protokołów warstwy aplikacji. Pomimo starań firmy IBM nie został on zaakceptowany przez organizację OMG i zaimplementowano go jedynie w środowiku IBM Aglets. Organizacja ta wprowadziła swój własny standard MASIF [15].

MASIF (*Mobile Agent System Interoperability Facility*) [15] – opracowany w latach 1995-1998 przez Agent Working Group, został zaakceptowany przez OMG w dniu 9 marca 1998 r. Jest jak dotąd jedynym standardem, który doczekał się pełnej implementacji (*Grasshopper*). Standard ten definiuje pewien model koncepcyjny (*common conceptual model*), opisujący podstawowe pojęcia i powiązania pomiędzy poszczególnymi jego składnikami. Podaje także własności i funkcje niezbędne do współpracy (*interoperatibility*) środowisk agentów różnych producentów. Nie zapewnia jednak przenośności środowisk pomiędzy różnymi architekturami sprzętowymi czy językami programowania, zostawiając ten problem twórcom poszczególnych środowisk. W wyniku takiego podejścia lokalne operacje na agentach, takie jak serializacja i deserializacja zależą od konkretnej implementacji. MASIF nie definiuje także zasad komunikacji pomiędzy agentami powołując się na istniejący standard komunikacji pomiędzy rozproszonymi obiektami, to znaczy CORBĄ.

Standard MASIF oferuje określoną terminologię, której użycie ma umożliwić jednolite zrozumienie koncepcji środowiska mobilnych agentów. Podane zostały między innymi definicje: *Agent*, *Stationery Agent*, *Mobile Agent*, *Agent State*, *Agent Authority*, *Agent Location*, *Agent System*, *Agent System Type*, *Agent System to Agent System Interconnection*, *Place*, *Region*. Według tego standardu środowisko agentów składa się z tzw. regionów (*regions*), w których znajdują się systemy agentów lub inaczej agencje (*Agent Systems*). Agencje składają się z kolei z miejsc (*places*), które są środowiskami wykonania poszczególnych agentów (patrz rys. 4). Komunikacja pomiędzy systemami agentów odbywa się za pomocą tzw. *Communication Infrastructure*.



Rys. 4. Architektura środowiska agentowego wg MASIF
Fig. 4. Architecture of standard agent environment defined by MASIF

Zdefiniowano trzy podstawowe typy interakcji pomiędzy agentami:

- Zdalne utworzenie agenta (*remote agent creation*), na żądanie klienta będącego innym agentem bądź dowolną aplikacją, przy czym do utworzenia agenta klient musi posiadać odpowiednie prawa.
- Przemieszczenie agenta (*agent transfer*) wymagające dokładnego wyspecyfikowania nazwy i adresu docelowego oraz jakości komunikacji, która jest już definiowana w ramach konkretnych implementacji.
- Wywołanie metody agenta (*agent methon invocation*) lub innego obiektu po uprzednim podaniu jego nazwy i adresu oraz jakości usługi.

W ramach standardu MASIF wyspecyfikowano funkcje, jakie powinno spełniać środowisko mobilnych agentów: Funkcje te podaje tabela 1.

Grupa AWG (Agent Working Group - część organizacji OMG) podczas prac nad standardem uwzględniła także możliwe zagrożenia i ataki na system mobilnych agentów i zaproponowała szereg strategii zabezpieczenia systemu. Między innymi środowisko agentowe powinno zapewniać:

- Ustalanie tożsamości klienta agencji (Client authentication)

Usługi systemu agentowego powinny zapewniać możliwość ustalania tożsamości agentowych i nieagentowych aplikacji klienckich.

- Wzajemne ustalenie tożsamości systemów agentowych

Agencje współpracujące bez interwencji ludzkiej muszą być zdolne do wzajemnego ustalania swojej tożsamości (authentication). Ten wymóg można zapewnić przez wprowadzania hasła dla każdorazowego uruchamiania agencji. Wówczas podczas komunikacji dwóch systemów agentowych odpowiednio zakodowane hasło może służyć jako klucz prywatny agencji.

Tabela 1

Funkcje środowiska agentowego objęte standardem MASIF

Typ funkcji	Opis
Zarządzanie agentami (*agent management*)	Tworzenie/usuwanie/wstrzymywanie/ponowienie pracy agentów w określonych miejscach sieciowych
Śledzenie agentów (*agent tracking*)	Znajdowanie agentów w środowiskach rozproszonych
Transport agentów	Metody przenoszenia agentów, ładowania kodu ich klas
Komunikacja agentów (*agent communication*)	Nie specyfikuje
Tworzenia nazw agentów, systemów agentów i lokacji sieciowych	Składnia nazw agentów umożliwiająca rozpoznanie ich typu a także ogólnie przyjęta składania opisu adresów miejsc sieciowych
Zapewnienie odpowiedniego poziomu bezpieczeństwa	Zabezpieczenie zasobów komputerów, na których działają agenty.

- Dostęp agencji do informacji o poziomie zaufania do danego agenta

Agencja musi mieć dostęp do informacji o agencie znajdującym się w innym systemie agentowym, który chce się skomunikować z jej wybranym agentem. Innymi słowy chęć komunikacji z agentem musi być poprzedzona sprawdzeniem, czy komunikujący się agent posiada odpowiednie uprawnienia.

- Potwierdzanie tożsamości agenta

Podczas migracji klucz potwierdzający tożsamość agenta powinien być przesłany razem z nim. Podczas zlecania wykonania zadania przez agenta innemu agentowi (np. brokerowi) powinien zostać przekazany również klucz agenta inicjującego, dzięki czemu możliwość wymiany zadań pomiędzy agentami odbywa się z zachowaniem kontroli uprawnień.

- Spójność, poufność, wykrywanie powtórzeń

Każda komunikacja powinna odbywać się przy zachowaniu poufności i spójności przesyłanych informacji. Ważne jest także, aby środowisko agentowe umożliwiało

wykrywanie powtórnych przesłań komunikatów, które mogą być oznaką próby podszycia się pod agenta bądź całą agencję.

Standard MASIF opisuje także usługi CORBA'y, mogące być wykorzystane w środowiskach mobilnych agentów. Należą do nich Naming Service, Lifecycle Service, Externalization Service, Security Service.

Inicjatywą skierowaną na umożliwienie współpracy różnych środowisk agentowych są też działania fundacji *Foundation for Intelligent Physical Agents*. Organizacja ta powstała w 1996 r. w Genewie. Celem jej jest promowanie i opracowywanie standardów podstawowych technologii agentowych w taki sposób, aby umożliwić współdziałanie różnych aplikacji agentowych (*agent-based aplications*). Opracowano do tej pory siedmioczęściową specyfikację FIPA 97 oraz rozpoczęto pracę nad nową specyfikacją FIPA 98, która nie jest jeszcze gotowa [16].

Specyfikacja FIPA obejmuje bardzo szeroki zakres zagadnień, obejmujący różne rodzaje agentów (nie tylko agentów programowych ale także robotów itp.). Specyfikuje ona tzw. zdolność ABC (Agent Basic Capabilities) określające interakcję:

- agentów z oprogramowaniem, sprzętem i ludźmi,
- agentów z innymi agentami – typy komunikatów, prymitywy komunikacyjne, zawartość wiadomości,
- społeczeństwa agentów – usług i bibliotek dostępnych dla społeczeństwa agentów.

Specyfikacja ta definiuje także rodzaje aplikacji agentowych, wyróżniając aplikacje: usługowe, biznesowe, wytwórcze i rozrywkowe oraz definiuje wymagania na język stosowany do komunikacji pomiędzy agentami tzw. ACL (Agent Communication Language). Standard FIPA nie doczekał się jeszcze pełnej implementacji w żadnym istniejącym środowisku mobilnych agentów (z wyjątkiem środowiska Grasshopper, które wspiera częściowo ten standard – odpowiednie klasy są dostępne jako rozszerzenie).

3. Przegląd środowisk agentowych

Standardy środowisk mobilnych agentów powstały w odpowiedzi na rosnącą liczbę różnych implementacji tworzonych zarówno w celach naukowych, jak i komercyjnych. Jak już sygnalizowano prawie żadne środowisko nie spełnia pełnych wymogów standardów MASIF czy FIPA, ponieważ większość z nich powstała przed ich opublikowaniem. Poza tym przyjęte pierwotnie założenia (np. wydajność) uniemożliwiały lub utrudniały późniejsze do nich przystosowanie.

Istnieje spora liczba środowisk mobilnych agentów (około 80). Różnią się znacznie oferowaną funkcjonalnością. Jedynie nieliczne tworzone są na zasadach komercyjnych produktów. Większość jednak tworzona jest jako projekty akademickie w celu zbadania możliwości paradygmatu agentowego. W niniejszej pracy ograniczono się do prezentacji kilku z nich, które są najbardziej znane w środowisku internetowym (listy dyskusyjne) i istnieją najdłużej. Szczegółowe porównanie ograniczono do czterech środowisk, które posiadają podobną ilość oferowanych funkcji. Poniżej przyjęto jednolity opis tych środowisk, zaś pełne wyniki ich porównania przedstawiono w tabeli 2.

Odyssey

Organizacja tworząca: General Magic Inc

Data powstania: 1997 r.

Aktualna wersja: projekt już zaniechany, wcześniej wykorzystywano język Tcl (Telescript)

Usługi nazw: brak

Typ migracji kodu: klasy ładowane z serwera wysyłającego

Bezpieczeństwo: brak

Mechanizmy komunikacji lokalnej i globalnej: brak

Standardy: brak

Uwagi: General Magic jako pierwsza firma wprowadziła środowisko mobilnych agentów zwane Telescript, oparte na języku skryptowym Tcl (AgentTcl). W odpowiedzi na rosnące znaczenie języka Java, szczególnie w zastosowaniach sieciowych, firma postanowiła zbudować nowe środowisko oparte na tym właśnie języku. Projekt, pomimo iż był jednym z pierwszych rozwiązań w tej dziedzinie, nie był następnie odpowiednio ulepszany czy rozwijany i ostatnio jego rozwój został zaniechany.

Concordia [12]

Organizacja tworząca: Mitsubishi

Data powstania: 1997

Aktualna wersja: 1.1.4 (styczeń 2000)

Usługi nazw: globalne i lokalne przy użyciu tekstowych identyfikatorów, Directory Manager jako dodatkowa usługa

Typ migracji kodu: na żądanie z wysyłającego hosta, na żądanie z serwera kodu

Bezpieczeństwo: szyfrowanie agentów podczas transmisji, checkpointing

Mechanizmy komunikacji lokalnej i globalnej: Wprowadzono dwa rodzaje wymiany komunikatów (*selected events* oraz *group events*). Pierwszy z nich znajduje zastosowanie w sytuacji, gdy nadawca komunikatu nie zna odpowiedniego odbiorcy. W tym przypadku odbiorca rejestruje się u odpowiedniego *EventManager'a* określając, jakie zdarzenia będzie

przyjmował. *EventManager* dba o dostarczanie wysyłanych wiadomości do zarejestrowanych odbiorców. Drugi rodzaj przesyłania wiadomości opiera się na utworzeniu grupy agentów. Agent chcąc wysłać komunikat do tej grupy używa specjalnego obiektu bez konieczności posiadania informacji o członkach całej grupy. Oba rodzaje komunikacji umożliwiają ponadto komunikację synchroniczną jak i asynchroniczną.

Standardy: rozpoczęcie przygotowanie do FIPA ACL

Uwagi: środowisko rozprowadzane jest na zasadach komercyjnej licencji, a do zastosowań akademickich jako wersja ewaluacyjna.

IBM Java Aglets [11]

Organizacja tworząca: IBM Tokyo Research Laboratory

Data powstania: 1996

Aktualna wersja: 1.0.3 / 1.1 beta 2

Usługi nazw: program Finder wprowadzony eksperymentalnie w wersji beta 1.1

Typ migracji kodu: na żądanie z wysyłającego hosta oraz z tzw. serwera kodu

Bezpieczeństwo: uwierzytelnianie tzw. bezpiecznej domeny, kontrola dostępu do komunikatów, mechanizmy bezpieczeństwa Java2,

Mechanizmy komunikacji lokalnej i globalnej: specjalna klasa reprezentująca wiadomości, wykorzystanie proxy jako mechanizm zdalnego dostępu do agentów

Standardy: przygotowane interfejsy zgodne z MASIF

Uwagi: Zespół rozwojowy firmy IBM z siedzibą w Tokio już w 1996 r. rozpoczął prace nad projektem, którego celem było stworzenie systemu mobilnych agentów. W efekcie w 1998 r. powstała pierwsza wersja Aglets SDK [11]. W sierpniu 1998 r. zespół poprawiając błędy zgłaszane przez użytkowników wypuścił wersję 1.0.3. Najnowszą dostępną wersją jest ASDK 1.1b2 (beta). W wersji tej postawiono przede wszystkim na bezpieczeństwo wprowadzając bezpieczeństwo zewnętrzne, oparte na koncepcji zaufanej domeny oraz wewnętrzne, oparte na listach dostępu znanych z JDK 1.2. Niestety, środowisko to na początku bardzo prężnie rozwijające się nie utrzymało tej tendencji do dziś. Wadą środowiska jest brak wsparcia dla stosowania JDK 1.2 (tylko JDK1.x) oraz brak zapewnienia całkowitej przeźroczystości lokacji (*location transparency*).

Voyager Orb Professional

Organizacja tworząca: ObjectSpace Inc

Data powstania: 1997 r.

Aktualna wersja: 3.2

Usługi nazw: dostępna jako uniwersalna usługa dla CORBA'Y, RMI i DCOM

Typ migracji kodu: na żądanie z wysyłającego hosta

Bezpieczeństwo: brak

Mechanizmy komunikacji lokalnej: lokalne wywołanie metod

Mechanizmy komunikacji globalnej: wywołanie metod przez dynamicznie generowane proxy, Voyager Remote Messaging Protocol (VRMP)

Standardy: CORBA

Uwagi: Jest to właściwie implementacja CORBA'y rozszerzona o możliwość tworzenia agentów. Oprócz tradycyjnego modelu programowania rozproszonego występującego w aplikacjach wykorzystujących CORBA'e środowisko umożliwia tworzenie aplikacji opartych na technologiach agentowych. Środowisko przeznaczone jest przede wszystkim do pracy z Java'ą. Istotną cechą wyróżniającą to środowisko jest współpraca i możliwość przezroczystego łączenia różnych standardów komunikacji rozproszonej. Oprócz standardowego protokołu specyficznego dla Voyagera możliwe jest wykorzystanie RMI, CORBA i DCOM (Distributed Component Object Model). Kolejną ważną zaletą środowiska jest mała objętość kodu implementacji ORB'a w porównaniu do innych środowisk. Agenty w tym środowisku traktowane są jak normalne obiekty w standardzie CORBA.

Grasshopper [13]

Organizacja tworząca: IKV++ Gmbh

Data powstania: sierpień 1998 r.

Aktualna wersja: 2.2 beta

Usługi nazw: region registry zgodny z MASIF

Typ migracji kodu: automatycznie z wysyłającego hosta lub serwera kodu

Bezpieczeństwo: Bezpieczeństwo jest podzielone na zewnętrzne, obejmujące przesyłanie informacji pomiędzy agencjami oraz wewnętrzne, zapewniające ochronę zasobów wewnętrznych agencji. Zapewnienie bezpieczeństwa zewnętrznego obejmuje szyfrowane połączenia (SSL), Message Authentication Code (MAC), zapewniające spójność przesyłanych wiadomości (MD5) i podpisy cyfrowe zapewniające identyfikację agentów (RSA 1024). Bezpieczeństwo wewnętrzne oparte jest na wbudowanym w JDK menedżerom bezpieczeństwa. Dużo uwagi poświęcono także zapewnieniu trwałości. Wykorzystanie SSL wymaga zainstalowania dodatkowego oprogramowania (IAIK).

Mechanizmy komunikacji lokalnej: lokalne wywołanie metod, synchroniczne i asynchroniczne

Mechanizmy komunikacji globalnej: wywołanie metod przez dynamicznie generowane proxy oraz poprzez znane z CORBA'Y DII (Dynamic Interface Invocation), synchroniczne i asynchroniczne

Standardy: MASIF i częściowo FIPA

Uwagi: Cechą charakterystyczną tego produktu jest możliwość współpracy z wieloma protokołami komunikacji: CORBA IIOP, MAF IIOP, RMI/Seucre RMI, plain/secure sockets.

Pegaz[14]

Organizacja tworząca: IPI PAN Warszawa

Data powstania: 1999 r.

Aktualna wersja: 2 marzec 2001 r., 1.3.8 z kwietnia 2000 r.

Usługi nazw: brak, każdy węzeł posiada jedynie listę sąsiadów

Typ migracji kodu: klasy dostępne wszystkim węzłom poprzez CLASSPATH

Bezpieczeństwo: brak

Mechanizmy komunikacji lokalnej i globalnej: specjalny interfejs i klasy reprezentujące wiadomości, brak możliwości wywoływania dowolnych metod

Standardy: w planach ACL (Agent Communication Language)

Uwagi: Celem powstania tego środowiska było stworzenie infrastruktury dla modelowania tzw. AVO's (Agent Virtual Organizations). Architektura systemu jest zbliżona do Concordii. Główna różnica polega na tym, że Pegaz jest środowiskiem w pełni rozproszonym, nie posiadającym centralnego punktu sterowania (Administration Manager w Concordii). Dzięki temu uzyskano możliwość dynamicznego (run-time) łączenia oddzielnych środowisk (sieci) poprzez uruchomienie węzła należącego do obydwu sieci. Zaleta ta jest okupiona bardziej złożonym systemem nazewnictwa i komunikacji. Agenty komunikują się ze sobą poprzez usługi udostępniane przez tzw. węzły (node). Usługi mogą być dynamicznie dodawane przez użytkownika. Nierozwiązane zostały natomiast podstawowe takie problemy jak zapewnienie bezpieczeństwa i trwałości.

4. Przykłady zastosowań środowisk agentowych

Metodologia agentowa pozwala na tworzenie różnorodnych aplikacji użytkowych. Nie ma jednego typu aplikacji, dla których byłaby ona przeznaczona. Istnieje za to wiele rodzajów aplikacji, dla których zastosowanie środowisk agentowych jest potrzebne. Poniżej przedstawiono kilka tego typu aplikacji w celu zademonstrowania przydatności tych środowisk.

Tabela 2

Porównanie właściwości wybranych środowisk agentowych

Kryteria\Środowiska	Concordia	Aglets
Komunikacja	Wiadomości mogą być wysyłane synchronicznie i asynchronicznie **selected-events** używane, gdy odbiorcy i nadawcy mogą nie wiedzieć o swoim istnieniu. Odbiorcy mogą wybrać, które wiadomości będą odbierać. **group-oriented events** używane do szybkiej komunikacji z określoną grupą agentów.	**Now-type** – synchroniczne; **oneway** – bez potwierdzania; **future** – asynchroniczne z możliwością sprawdzenia, czy wiadomość dotarła.
Przeźroczystość lokacji Location Transparency	Pełne, wykorzystanie **proxy** przy komunikacji grupowej (GroupProxy). Uwzględniane są migracje agentów.	Częściowe wykorzystanie tzw. **proxy** służącego jako łącze z agentem. Nieuwzględniane są migracje agentów.
Usługi nazw	**Directory Manager** niezbyt rozbudowany system katalogowy w formie obiektu dostępnego lokalnie oraz przez RMI. Umożliwia rejestrowanie statycznych obiektów pełniących rolę usług tzw. services dotyczących bazy danych.	Lokalna lista agentów w danym miejscu sieciowym. Eksperymentalne wprowadzenie serwera rejestrującego agenty **Finder** (niezgodny ze standardem MASIF).
Praca w trybie "thin client"	**Light-weight Agent Transporter API** wykorzystujący RMI	Projekt **Fiji**, który został potem zaniechany (agencja w applecie).
Trwałość .	Tylko w wersji komercyjnej	Niepełna
Umieszczenie agenta w pamięci masowej	Środowisko automatycznie	Agent
Automatyczne odtwarzanie agentów po upadku systemu	Tak	Jako eksperymentalna funkcjonalność
Transakcyjne przesyłania agentów (buforowanie)	Tak - Agent Queuing Support	Brak
Cykl życia agenta	Agenty istnieją wiecznie, (usuwanie poprzez samego agenta, lub użytkownika/właściciela).	Agenty istnieją wiecznie, (usuwanie poprzez samego agenta, lub użytkownika/właściciela).
Bezpiecz. Wewnętrzne	Tylko w wersji komercyjnej oparte na Security Manager dostępnym JDK	Oparte na Security Manager dostępnym JDK
Rule based security	Tak	Tak
Identity based security	Tak	Tak
Bezpiecz. Zewnętrzne	Tak	Ograniczone
Identyfikacja (authentication)	Oparte na zaufaniu do serwerów wchodzących w skład określonej domeny	Oparte na zaufaniu do serwerów wchodzących w skład określonej domeny.
Spójność (integrity)	Korzysta z możliwości, jakie daje Java CryptoGraphy Extension (IDEA, DES, RC4, RC5).	Brak
Poufność (confidentality)	Szyfrowane połączenia SSL v3	Brak
Inne zabezpieczenia	Brak	Wykorzystanie firewall oraz HTTP tuneling
Protokoły komunikacji	TCP/IP sockets, RMI	TCP/IP sockets RMI, w planach wykorzystanie CORBA'y

cd. tabeli 2

Kryteria\Środowiska	Voyager	Grasshopper
Komunikacja	**Sync** – synchroniczne; **oneway** – bez potwierdzenia; **future** – asynchroniczne z możliwością sprawdzenia czy wiadomość dotarła.	**Synchronous** – synchroniczne; **Asynchronous** – asynchroniczne z potwierdzeniem wiadomości lub z automatycznym powiadamianiem; **Dynamic Communication** – możliwość wywoływania procedur bez znajomości struktury serwera (**CORBA DII**); **Multicast Communication** - możliwość jednoczesnego wyw. procedur na kilku serwerach.
Location Transparency	Pełne wykorzystanie tzw. proxy służącego jako łącze z agentem. Uwzględniane są migracje agentów.	Pełne wykorzystanie tzw. proxy jako łącza z agentem. Uwzględniane są migracje agentów.
Usługi nazw	**Universal Namin Service** umożliwiający dostęp do JNDI (Java Naming Directory Interface), CORBA name service, RMI Registry, DCOM Active Directory.	Poprzez dostęp do centralnej bazy **Region Registry**. Istnieje możliwość dostępu poprzez interfejsy zgodne ze standardem MASIF (**MAFFinder**).
Praca w trybie "thin client"	**Voyager Ultra-light Client** – klient ORB o zajętości kodu 15KB.	Tzw. dynamic proxy umożliwiające zdalne kontrolowanie agencji i agentów.
Trwałość	Niepełna	Pełna
Umieszczenie agenta w pamięci masowej	Brak możliwości (planowane)	Agent oraz użytkownik środowisko co zadany przez użytkownika lub agenta kwant czasu.
Automatyczne odtwarzanie agentów po upadku systemu	Brak	Tak
Transakcyjne przesyłania agentów (buforowanie)	Zgodne z CORBA OTS bez buforowania	Tak
Cykl życia agenta	Agent żyje dopóki są do niego zdalne referencje (CORBA); przez określony czas; do określonego momentu czasu; do momentu, aż będzie przez określony czas nieaktywny; wiecznie.	Agenty istnieją wiecznie, (usuwanie poprzez samego agenta, lub użytkownika/właściciela)
Bezpiecz. Wewnętrzne	Oparte na Security Manager dostępnym JDK	Oparte na Security Manager dostępnym JDK
Rule based security	Brak	Tak
Identity based security	Tak	Tak
Bezpiecz. Zewnętrzne	Brak	Pełne
Identyfikacja (authentication)	Brak	Certyfikaty zgodne z X.509
Spójność (integrity)	Brak	Wiadomości szyfrowane za pomocą MAC's (Message Authentication Code) Md5
Poufność (confidentality)	Połączenia SSL tylko w wersji Professional	Połączenia klient-serwer szyfrowane SSL (RC4 128) i wymianą kluczy (RSA 1024)
Inne zabezpieczenia	Wykorzystanie firewall SOCKS5, HTTP tuneling	Brak
Protokoły komunikacji	CORBA IIOP, RMI, DCOM	MAF IIOP, CORBA IIOP, RMI/secure RMI, socket/secure socket, SSL po zainstalowaniu (IAIK)
Standardy	Brak	MASIF, FIPA po dołączeniu załączników

4.1. Monitorowanie i powiadamianie

Agenty ukazują swoją przewagę nad zwykłymi aplikacjami w przypadku systemów monitorowania i powiadamiania. Konieczność obserwowania i kontroli urządzeń zewnętrznych oraz stanu systemu operacyjnego wymaga ciągłego dostępu do tych zasobów i może być realizowana przez sieć komputerową, gdy użytkownik jest na stale podłączony do sieci. Przykładem takiej aplikacji może być agent, który monitoruje działanie urządzenia zewnętrznego, np. drukarki sieciowej. W momencie wystąpienia problemów np. braku tonera lub papieru przemieszcza się (lub wysyła wiadomość) do komputera pracownika technicznego odpowiedzialnego za sprawność drukarki. Może także, gdy wykryje błąd w funkcjonowaniu drukarki, wysłać e-maila do serwisu producenta drukarki.

4.2. Wspomaganie użytkowników

Przykładem zastosowania mobilnych agentów mogą być aplikacje działające jako elektroniczni asystenci podczas nieobecności użytkownika. Agenty mogą wykonywać wówczas zadania w imieniu swoich właścicieli, np. mogą wyłączać komputery o określonej godzinie, włączać/wyłączać określone usługi, automatycznie odpowiadać na listy (np. że użytkownika nie ma w biurze), mogą także negocjować z innymi agentami terminy spotkań analizując wszystkie dostępne terminy każdego użytkownika i dobierać czas spotkania, gdy wszyscy mają wolny czas.

4.3. Równoważenie obciążenia

Wiele komputerów posiada dużo nadmiarowej mocy obliczeniowej, ponieważ są one wykorzystywane przez swoich użytkowników tylko w ograniczonym zakresie. Z drugiej strony jest wiele różnych problemów z takich dziedzin nauki, jak: chemia czy fizyka, w których zapotrzebowanie na moc obliczeniową jest olbrzymia, np. przy symulacji tego typu zjawisk. Dlatego opracowuje się wiele środowisk umożliwiających rozproszone obliczenia, np. PVM, MPI i inne typu GRID, które zapewniają wykorzystanie wolnych mocy obliczeniowych. Wspierają one równoważenie obciążenia różnych węzłów poprzez migrację procesów. Polega to na przenoszeniu aktualnie wykonywanych procesów z jednego komputera na drugi z zachowaniem dotychczasowych obliczeń. W pewnym sensie środowiska te realizują funkcje mobilnych agentów, nie dają jednak możliwości decydowania przez twórcę programu kiedy i gdzie przenieść proces. Wykorzystanie tam mobilnych agentów może okazać się nieocenione, zwłaszcza gdy dokonanie migracji nie jest oczywiste, a przeniesienie procesu wiąże się ze stratą czasu na zapamiętanie jego stanu i transmisją przez

sieć. W takich przypadkach to agent związany z daną aplikacją lepiej rozpoznaje zaistniałą sytuację i ogranicza migrację do niezbędnego minimum, co prowadzi do zwiększenia jej efektywności funkcjonowania.

4.4. Rozproszone wyszukiwania informacji

Agenty dobrze spisują się podczas wyszukiwania danych w systemach rozproszonych, w których istnieje wiele systemów baz danych posiadających różne interfejsy dostępu. W tego typu systemach należy najpierw określić, gdzie szukać konkretnych informacji. Przykładami takich systemów mogą być cyfrowe biblioteki czy bazy multimedialne. W tradycyjnych rozwiązaniach użytkownik musi połączyć się z każdą z tych baz z osobna i dokonać wyszukania. Następnie na podstawie dostępnych informacji dokonuje ponownego wyszukania. Wykorzystanie agentów może eliminować takie postępowanie poprzez automatyzację równoległego wyszukania we wszystkich dostępnych bazach. Szczególnie przydatne może być zastosowanie agentów do wyszukania danych multimedialnych, na przykład określonych zdjęć archiwalnych.

Użytkownik chce wyszukać zdjęcia podobne do tego, które aktualnie posiada. Porównywanie zdjęć jest problemem złożonym i na serwerze z bazą danych jest dostępnych tylko kilka prostych algorytmów wyszukiwania. Użytkownik dysponuje jednak własnym i efektywnym algorytmem porównywania zdjęć, który nie jest zaimplementowany na serwerze bazy danych. Jeżeli zechce on wyszukać je tym swoim algorytmem, to przy użyciu tradycyjnej techniki będzie to wymagało przesłania całej zawartości bazy na lokalny komputer użytkownika i uruchomienia tego algorytmu. Jeżeli jednak aplikacja zostanie napisana przy użyciu paradygmatu agentowego, to użytkownik stworzy po prostu odpowiedniego agenta, który zostanie wysłany na serwer z bazą danych, gdzie wykonując ten algorytm dokona wyszukania zdjęć poprzez lokalny dostęp do bazy danych. Następnie powróci wraz ze zbiorem podobnych zdjęć. Zauważmy, że zysk czasowy może być ogromny. Tak więc nieefektywne aplikacje mogą być wykowane w akceptowalnym czasie.

4.5. Wspomaganie handlu elektronicznego

Transakcje handlowe wymagają dostępu on-line do zasobów takich jak giełdy akcji czy elektroniczne sklepy. Także interakcje pomiędzy stronami transakcji handlowych wymagają zazwyczaj wymiany danych on-line. Agenty idealnie spełniają wymagania pracy on-line, ponieważ możliwość migracji do miejsca docelowego i interakcja lokalna uniezależniają je od zmian w obciążeniu sieci.

Kolejną zaletą metodologii agentowej w tworzeniu aplikacji jest łatwość osiągnięcia dużej personalizacji zachowania aplikacji. W aplikacjach handlu elektronicznego, szczególnie w wirtualnych domach maklerskich, występuje potrzeba dostosowania algorytmu kupowania/sprzedawania/wymiany towarów do wymagań konkretnych użytkowników. W tradycyjnych aplikacjach można umieścić wiele reguł postępowania ale na pewno przyjdzie moment, że wystąpi potrzeba użycia akurat takiego, który nie został przewidziany podczas projektowania aplikacji. Elastyczność aplikacji agentowych umożliwia dodawanie nowych algorytmów podczas pracy systemu. Twórcy systemów i aplikacji mogą nawet nie zdawać sobie sprawy, jakie algorytmy wykorzystają użytkownicy wprowadzając swoich własnych agentów. Dla przykładu można sobie wyobrazić agentów nastawionych na jak najszybsze kupowanie towarów bez względu na cenę oraz na drugim biegunie agentów kupujących towary bez ograniczenia czasowego długo wybierających najtańszą ofertę. Innym przykładem mogą być agenci, którzy zachowywaliby się jak profesjonalni maklerzy, kupowali akcje spółek, gdy są „tanie", a sprzedawali, gdy są „drogie". Każdy użytkownik w takim przypadku chciałby sam ustalić sposób rozpoznawania, że akcje są „drogie" lub „tanie". Dla jednych byłaby to bezwzględna wartość, dla innych być może określona ilość sesji poniżej jakiejś ceny.

5. Wnioski końcowe

Aplikacje budowane na podstawie środowisk agentowych mają przed sobą dużą przyszłość. Warunkiem ich przydatności jest jednak ciągły rozwój ich funkcjonalności oraz zapewnienie odpowiedniego poziomu bezpieczeństwa. Innym wyzwaniem może stać się opracowanie efektywnych algorytmów agentowych związanych ze współpracą różnego typu agentów. Dotyczy to nie tylko algorytmów komunikacji czy wyszukiwania informacji ale również, lub przede wszystkim, algorytmów negocjacji czy też perswazji, które znajdują coraz szersze zastosowanie zarówno w aplikacjach handlu elektronicznego, jak też jeszcze bardziej złożonych aplikacji związanych z kreowaniem miasta czy państwa cyfrowego.

LITERATURA

1. Gustawvsson R.: *„Agents with Power"*, Communications of the ACM, Marzec 1999/Vol. 42. No 3.

2. Maes P., Guttman R. H., Moukas A. G.: *„Agents than buy and sell"*, Communications of the ACM, Marzec 1999/Vol. 42. No 3.
3. Hattori F., Ohguro T., Yokoo M., Matsubara S., Yoshindas.: *„Multiagent systems for supproting Network Communities"*.
4. Jain A.K., Singh M.P.: *"Agents for process coherence in Vitual Enterprises"*, Communications of the ACM, Marzec 1999/Vol. 42. No 3.
5. Lange D.B., Wong D., Paciorek N., Moore D.: *„Java-based Mobile Agents"*, Communications of the ACM, Marzec 1999/Vol. 42. No 3.
6. Lange D.B.: "*Programming and Deploying Mobile Agents with Aglets*", Addison-Wesley, 1998.
7. Franklin S., Graesser A.: "*Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*", Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.
8. Oshima M., Karjoth G, Ono K.: "Aglets Specification 1.1 Draft", IBM Tokyo Research Lab, 1998, http://www.trl.ibm.co.jp/aglets
9. Walsh T., Paciorek N., Wong D.: "Security and Reliability in Concordia", 1998, *http://www.merl.com/HSL/Projects/Concordia/*
10. IKV GmbH, "Grasshopper Programers Guide 2.1", 1998, www.grasshopper.de
11. Ambroszkiewicz S., Nowak T.: „Platforma agentowa 1.3.5", luty 2000, http://pegaz.ipipan.waw.pl/pa_1_3.html
12. Glass G.: "Overview of Voyager: ObjectSpace's", CTO ObjectSpace, 1999.
13. Papaioannou T.: „Mobile Agents: Are They Useful for Establishing a Virtual Presence in Space?", MSI Research Institute, Department of Manufacturing Engineering, Loughborough University, Loughborough 1999.
14. Silva A., Delgado J.: "The Agent Pattern for Mobile Agent Systems", INESC & IST Technical University of Lisbon, 1998.
15. OMG Group, "CORBA Mobile Agent Facility specification", styczeń 2000.
16. FIPA, "Abstract Architecture Specification", Geneva, Switzerland, styczeń 2001.

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 28 marca 2001 r.

Abstract

In distributed applications one of three co-operation models can be implemented in practise. The traditional approach is RPC mechanisms (see Fig. 1) which are used in client-server model. Owing to such drawbacks as low usability and performance a new solution is proposed. It is named client-broker-server, and its implementation is CORBA architecture. Such a model (see Fig. 2) allows to perform application objects regardless of where these objects are located. The further improvements of the model lead to agent-oriented environments, where mobility of some objects is added (see Fig. 3). There are various possibilities to create such environments. Therefore to make possible co-operations of different implementations some standards are proposed. In the paper MASIF architecture (see Fig. 4) is described and its relevance to CORBA standard is also analysed. Moreover, FIPA standard describing co-operation of different agent-oriented applications is also mentioned.

The paper reviews also the most representative agent-oriented environments, such as: Voyager, Grasshopper, Concordia and Aglets. Taking into considerations such attributes as: functionality, transparency, name services, persistence and security, these environments are compared, and their suitability to design different distributed applications are described. Besides, examples of such applications are also presented.