STUDIA INFORMATICA 2011
Volume 32 Number 2B (97)

Igor WALIGÓRA, Bożena MAŁYSIAK-MROZEK, Dariusz MROZEK
Politechnika Śląska, Instytut Informatyki

UNIWERSALNA PLATFORMA WIELOAGENTOWA UMAP

Streszczenie. Systemy wieloagentowe pozwalają tworzyć własne rozwiązania programowe oparte na grupach współpracujących ze sobą agentów, realizujących wspólne cele. W niniejszym artykule przedstawiono nową, uniwersalną platformę wieloagentową UMAP, która dostarcza gotowych rozwiązań w zakresie tworzenia agentów programowych, pozostawiając programistom jedynie konieczność zaimplementowania logiki działania agentów, adekwatnie do realizowanego projektu. W artykule przedstawiono informacje dotyczące działania utworzonej platformy, jej architekturę oraz podstawy teoretyczne i standardy, na podstawie których została ona zaprojektowana.

Słowa kluczowe: systemy wieloagentowe, agenci programowi

UMAP – UNIVERSAL MULTI-AGENT PLATFORM

Summary. Multi-agent systems allow to build custom software solutions that work on the basis of groups of cooperating agents, realizing a common goal. In the paper, we present a new, universal multi-agent platform (UMAP), which provides ready to use solutions for the implementation of software agents, leaving the programmers only the necessity to implement business logic of agents, according to the expected deliverables. We also show various considerations related to the functioning of the UMAP platform, its architecture, theoretical foundations and standards underlying the design of the platform.

Keywords: multi-agent system, program agent

1. Wprowadzenie

Rozwój informatyki od czasów jej powstania umożliwia tworzenie coraz bardziej złożonych systemów, których skomplikowanie wymagało zmiany podejścia do tworzenia oprogramowania. Zarówno koncepcja projektowania programów, jak i paradygmaty ich tworzenia

uległy zmianie. Programowanie imperatywne (proceduralne) przestało być wystarczające, gdyż nie pozwalało w jasny i czytelny sposób tworzyć złożonych programów. Istotą problemu były między innymi trudności w opisaniu, realizacji oraz testowaniu projektu jako całości. W związku z tym uczyniony został naturalny krok w kierunku programowania modularnego oraz programowania obiektowego – rozbicie problemu na mniejsze części. Szybki wzrost sieci komputerowych, zarówno globalnych, jak i lokalnych, dał użytkownikom możliwość rozproszenia systemów na wiele logicznych podsystemów. Podejście takie ułatwia zaprojektowanie oraz wykonanie systemu. Jednak należy wspomnieć, że związane są z nim problemy, takie jak na przykład sposób komunikacji czy też jej zawodność.

Spojrzenie na problem złożonych systemów jak na grupę niezależnych, lecz komunikujących się ze sobą bytów, prowadzi do określenia kolejnego paradygmatu programowania – programowania agentowego. Przy tym podejściu zakładamy, że projekt systemu to wiele niezależnych i podejmujących decyzje agentów, komunikujących się ze sobą w celu osiągnięcia wspólnych celów. Agenta cechują dwie właściwości – zdolność podejmowania akcji oraz komunikacji z innymi agentami. Zbiór wielu agentów współpracujących ze sobą w celu osiągnięcia wspólnego celu nazywamy właśnie systemem wieloagentowym.

Szczególną rolę systemy wieloagentowe pełnią w systemach wspomagania decyzji (systemy ekspertowe), gdzie dane niezbędne do podjęcia decyzji rozproszone są pomiędzy bazy danych różniące się dziedziną przechowywanych informacji. Wiele rozwiązań w informatyce przemysłowej również opiera się na agentach, ponieważ systemy takie są odporne na awarie, potrafią operować na niepełnych danych oraz w szczególnych przypadkach podejmować właściwe decyzje nawet w przypadku otrzymania błędnych danych [1].

Największy potencjał systemów wieloagentowych wynika z rozproszenia wiedzy oraz funkcji systemu pomiędzy poszczególne jego składowe. Dzięki temu możliwe jest rozwiązywanie złożonych problemów przy wykorzystaniu zróżnicowanych dziedzinowo zasobów czy też systemów rozwiązujących różne klasy zagadnień. Wyzwaniem w takim przypadku jest umożliwienie nie tyle integracji, co wzajemnej komunikacji pomiędzy poszczególnymi systemami przez wytworzenie standardów określających sposób oraz format komunikowania się systemów oraz agentów [2, 3, 4].

W niniejszym artykule przedstawiono ogólne informacje na temat nowej, zaprojektowanej i zaimplementowanej przez autorów, uniwersalnej platformy wieloagentowej UMAP (ang. *Universal Multi-Agent Platform*), która umożliwia uruchamianie współpracujących ze sobą agentów, działających oraz komunikujących się zgodnie z określonymi standardami. Platforma udostępnia jednolity szkielet agenta, na którym może się opierać dowolne rozwiązanie agentowe. Agent platformy UMAP ma ustalony interfejs, który umożliwia komunikację oraz zarządzanie agentem w ramach platformy.

2. Wymagania funkcjonalne dla platformy UMAP

Kluczowym elementem przy tworzeniu oprogramowania będącego szkieletem (ang. *framework*) jest zdefiniowanie funkcjonalności, którą spełnia, a także wymagań stawianych wobec twórcy rozwiązania na nim opartego.

Celem autorów było zaprojektowanie i zaimplementowanie platformy umożliwiającej działanie agentów programowych w określonym środowisku. Aby system był pełny, wymagane są mechanizmy uruchamiania, nadzorowania działania oraz komunikacji pomiędzy agentami zarówno wewnątrz instancji systemu, jak i pomiędzy wieloma systemami uruchomionymi na jednym bądź wielu fizycznych maszynach podłączonych do sieci.

2.1. Standardy określające funkcjonowanie agentów

Wraz z rozwojem problematyki systemów wieloagentowych powstawały standardy określające architekturę, sposób opisu oraz przede wszystkim komunikację pomiędzy agentami. Powstanie określonych norm oraz reguł jest dla systemów wieloagentowych szczególnie ważne ze względu na wymóg współpracowania ze sobą systemów tworzonych w różnych technologiach.

FIPA (*Foundation for Intelligent Physical Agents*) jest organizacją zawiązaną w celu opracowania specyfikacji dla niejednorodnych systemów wieloagentowych. Organizacja od czasu założenia w 1996 r. stworzyła kilkadziesiąt dokumentów zawierających wymagania stawiane przed architekturą i odnoszące się do sposobu komunikacji czy zarządzania agentami. Podzbiór dwudziestu pięciu dokumentów tworzy standard FIPA, opisując architekturę systemów, wymagania co do zarządzania agentami oraz przebieg i format komunikacji.

FIPA określa pewne elementy składowe platformy wieloagentowej, na które składają się Agent Management System (AMS), Directory Facilitator (DF) oraz Message Transport System (MTS). Moduł AMS odpowiedzialny jest za uruchamianie nowych agentów, sprawuje także kontrolę nad działającymi już agentami. Moduł DF umożliwia rejestrację i wyszukiwanie usług oferowanych przez agentów. Moduł MTS odpowiada za przesyłanie wiadomości pomiędzy agentami wewnątrz platformy, a także za komunikację ze zdalnymi platformami [5, 6].

2.2. Zarządzanie agentami

Zapewnienie ścisłej kontroli nad działaniem poszczególnych agentów jest pierwszym zagadnieniem, które należy zdefiniować. Standard FIPA [5] wymaga, aby każdy agent uruchamiany był w kontrolowanym środowisku. W szczególności system zarządzania agentami musi być świadomy uruchomionych w jego obrębie agentów. System taki powinien posiadać

możliwość kontroli stanu agentów oraz przerywania ich działania. Dla zapewnienia działania systemu wieloagentowego niezbędne są również mechanizmy rejestrowania agentów, a także wymiana informacji o ich obecności pomiędzy kilkoma instancjami aplikacji.

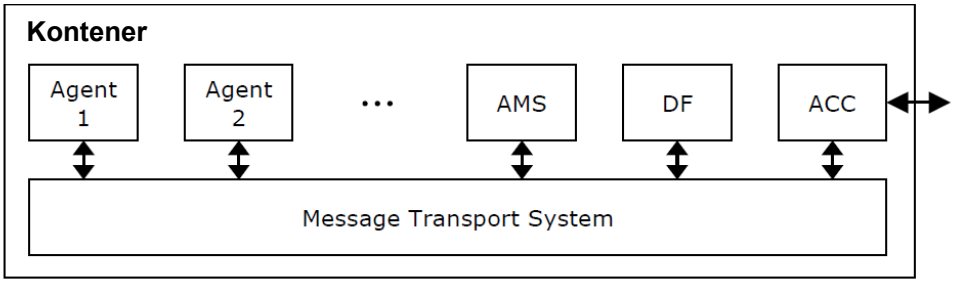
Projektując opisywany system, kierowano się standardami FIPA, opisującymi składowe platformy wieloagentowej, o których wspomniano w rozdziale 2.1. Standard nie określa, w jaki sposób te najważniejsze składowe systemu powinny zostać zaimplementowane. Chociaż logicznie są to rozdzielne moduły, mogą one być stworzone jako jeden komponent. W prezentowanym przez autorów rozwiązaniu zdecydowano się na stworzenie systemu zarządzania (AMS) będącego jednocześnie środowiskiem uruchomieniowym, wewnątrz którego działają agenci. Usługi transportu wiadomości (MTS) oraz usługi katalogowania (DF) działają jako komponenty uruchamiane i działające bezpośrednio pod kontrolą środowiska uruchomieniowego (AMS).

2.3. Komunikacja

Definicja systemów wieloagentowych mówi, iż możliwość wzajemnego komunikowania się agentów jest niezbędna do działania systemu. Istnieje co najmniej kilka języków wykorzystywanych przez systemy agentowe, z czego dwa najpopularniejsze to FIPA-ACL (*Agent Communication Language*) i KQML (*Knowledge Query and Manipulation Language*). Na potrzeby niniejszej pracy skupimy się na języku FIPA-ACL.

Modułem systemu odpowiedzialnym za transport wiadomości jest System Transportu Komunikatów (MTS, *Message Transport System*), który jest obowiązkowym elementem platformy wieloagentowej tworzonej na podstawie standardu FIPA [6]. Usługa transportu wiadomości powinna być niezawodna, dokładna oraz zachowywać kolejność komunikatów. Na rysunku 1 przedstawiono umiejscowienie systemu transportu wiadomości w obrębie kontenera, w który może pracować wielu agentów, sposób komunikacji z agentami i pozostałymi składowymi systemu. Kanał komunikacyjny (ang. *Agent Communicator Channel*) jest odpowiedzialny za komunikację ze zdalnymi kontenerami.

Zarówno struktura, jak i przebieg wysyłania komunikatów muszą być ujednolicone, tak aby zapewnić kompatybilność pomiędzy agentami działającymi na heterogenicznych platformach. Do realizacji platformy UMAP niezbędne było zaimplementowanie struktury wiadomości opisanej w standardzie [7] oraz stworzenie mechanizmu kodowania wiadomości, która ma zostać wysłana.



Rys. 1. System Transportu Komunikatów (MTS)
Fig. 1. Message Transport System (MTS)

3. Przegląd istniejących rozwiązań

Systemy agentowe wykorzystywane są do rozwiązywania różnych problemów, począwszy od systemów sterowania procesami przemysłowymi do systemów wspomagania decyzji czy analizy rynków walutowych. Zależnie od potrzeb oraz wymagań są to systemy tworzone od podstaw lub na bazie gotowych platform, oferujących niezbędne komponenty odpowiedzialne za działanie systemu, pozostawiając do zaimplementowania jedynie logikę działania agentów. W niniejszym rozdziale przedstawiono przykłady konkurencyjnych rozwiązań wieloagentowych.

3.1. Platforma JADE

JADE (*JAVA Agent DEvelopment Framework*) [9] to najbardziej rozwinięty projekt platformy wieloagentowej, napisany zgodnie ze specyfikacją FIPA. System powstał w Telecom Italia Lab (TILAB). Jest systemem typu otwartego, z dostępem do kodu źródłowego. Dystrybuowany jest zgodnie z formułą licencyjną LGPL (*Lesser General Public License Version 2*). W skład Komitetu Zarządzającego projektem wchodzi pięciu członków: Telecom Italia, Motorola, Whitestein Technologies AG, Profactor GmbH i France Telecom R&D. System JADE jest napisany w języku Java i możliwe jest jego wykorzystanie w dowolnej wersji tej platformy: J2SE, J2EE i J2ME. Agenci platformy JADE mogą komunikować się ze sobą zgodnie ze standardem języka komunikacji ACL (ang. *Agent Communication Language*).

3.2. Platforma SPADE

SPADE (*Smart Python multi-Agent Development Environment*) [10] jest platformą wieloagentową, bazującą na technologii XMPP/Jabber, zaimplementowaną w języku Python. Platforma SPADE udostępnia użytkownikom funkcje, które ułatwiają budowę systemów wie-

loagentowych. Jako przykład takich funkcji można wymienić zdefiniowane kanały komunikacyjne czy rozszerzalny protokół komunikacji bazujący na XML, podobnie jak FIPA-ACL. Należy zaznaczyć, że SPADE, jako pierwsza platforma wieloagentowa, opiera się na technologii XMPP. XMPP (*Extensible Messaging and Presence Protocol*) jest technologią pozwalającą na komunikację w czasie rzeczywistym, która udostępnia szeroki zakres zastosowań, takich jak błyskawiczne przesyłanie wiadomości, konwersacje wielosobowe, połączenia głosowe oraz wideo [11].

3.3. FIPA-OS

FIPA-OS (*FIPA Open Source*) [12] opisywana jest jako zbiór narzędzi umożliwiających sprawną implementację systemu wieloagentowego, zgodnego ze specyfikacją FIPA. FIPA-OS jest systemem typu otwartego, z dostępem do kodu źródłowego. System ten został po raz pierwszy wydany w sierpniu 1999 r. System FIPA-OS został zaimplementowany przy użyciu środowiska Java. Dostępne są dwie wersje tego systemu: Standard FIPA-OS oraz MicroFIPA-OS. Pierwsza wersja obejmuje dwie dystrybucje, kompatybilne odpowiednio z Java 2 i Java 1.1. MicroFIPA-OS to rozszerzenie wersji Standard kompatybilnej z Java 1.1. Wersja ta została utworzona przez naukowców z Uniwersytetu w Helsinkach. Została ona zaprojektowana tak, aby współpracować z urządzeniami mobilnymi, takimi jak PocketPC czy Compaq iPaq.

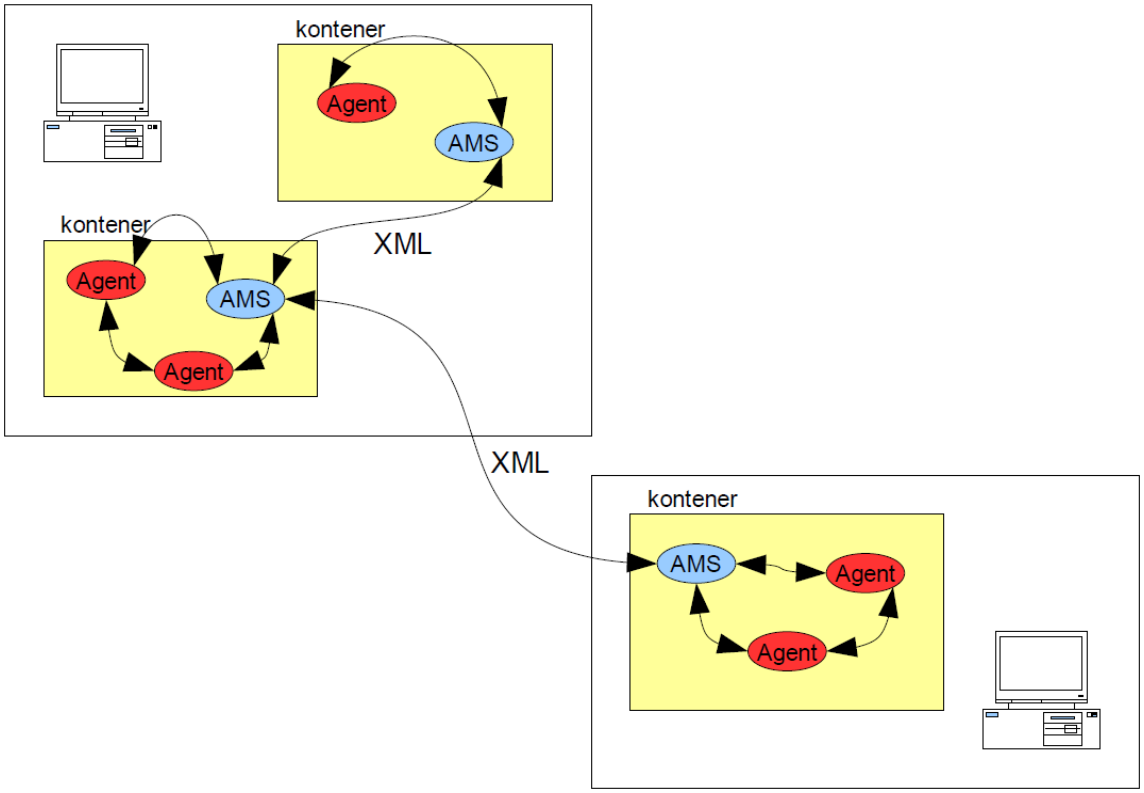
4. Platforma UMAP

Platforma UMAP (ang. *Universal Multi-Agent Platform*) powstała w 2010 r. w Instytucie Informatyki Politechniki Śląskiej w Gliwicach. Platforma ma charakter uniwersalny, tzn. pozwala tworzyć dowolne rozwiązania programowe oparte na kooperacji wielu agentów programowych. W niniejszym rozdziale przedstawiono najważniejsze cechy platformy, począwszy od jej architektury, poprzez ogólny interfejs agenta, aż po opis środowiska uruchamiania agentów na platformie UMAP.

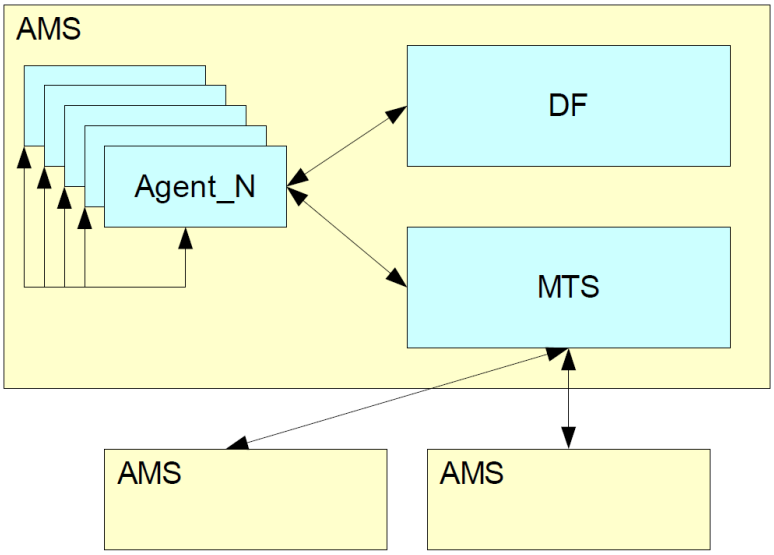
4.1. Architektura platformy UMAP

Architekturę platformy wieloagentowej UMAP przedstawiono na rysunku 2. Platforma agentowa tworzy kontenery, wewnątrz których uruchamiane są procesy agentów. Wewnątrz kontenera działa system zarządzania agentami (AMS, ang. *Agent Management System*), który zawiera w sobie elementy, takie jak usługę katalogowania agentów (DF, ang. *Directory Faci-*

litator), system transportu wiadomości (MTS, ang. *Message Transport System*). System zarządzania agentami umożliwia jednak przede wszystkim uruchamianie oraz nadzorowanie pracy agentów.



Rys. 2. Ogólna architektura platformy wieloagentowej UMAP
Fig. 2. Overview of UMAP multiagent platform architecture



Rys. 3. Wewnętrzna organizacja kontenera z modułem AMS, DF i MTS
Fig. 3. Internal arrangement of container with AMS, DF and MTS modules

Agenty działają wewnątrz platformy, która ma możliwość nadzorowania ich pracy, w tym również zakończenia działania konkretnego agenta. Agenty mogą komunikować się ze sobą

bezpośrednio w obrębie jednego kontenera, natomiast komunikację z agentami działającymi na zdalnym komputerze (lub fizycznie tym samym, lecz wewnątrz innego kontenera) umożliwia system transportu wiadomości (MTS). Każdy kontener udostępnia również usługę katalogu agentów (DF), zawierającą informacje o znanych agentach oraz ich opis. Katalog agentów dostępny jest również dla agentów zdalnych, przez system transportu wiadomości (rys. 3).

Na jednym komputerze może być uruchomionych wiele kontenerów. Komunikacja pomiędzy agentami działającymi na różnych kontenerach odbywa się przez gniazda sieciowe, a wiadomości przesyłane są w formacie XML.

4.2. Instalacja platformy UMAP

Platforma wieloagentowa UMAP dostarczona jest jako aplikacja okienkowa, służąca do uruchamiania oraz zarządzania agentami. Proces instalacji jest automatyczny, odbywa się za pomocą dostarczonego instalatora. Instalator ponadto sprawdzi, czy w systemie dostępne jest środowisko .NET Framework 2.0 i w razie potrzeby zleci użytkownikowi pobranie odpowiedniego pakietu z zasobów firmy Microsoft.

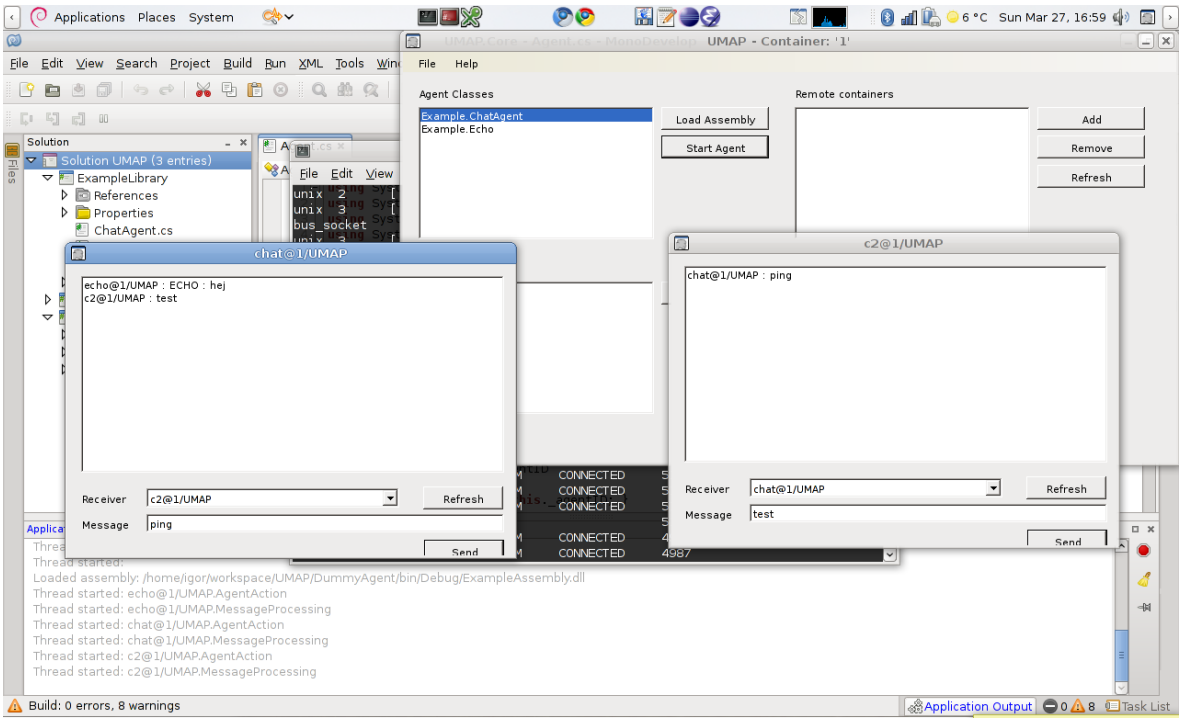
Po zainstalowaniu aplikacja jest gotowa do użycia, można ją wywołać przez skrót utworzony na pulpicie pod nazwą „UMAP Managemet System". Poza instalacją aplikacji okienkowej, umożliwiającej pracę platformy, do katalogu docelowego instalacji kopiowana jest biblioteka UMAP.Core.dll. Biblioteka ta jest niezbędna do implementacji własnego agenta. Zawiera ona klasę bazową dla agenta oraz wszystkie składniki platformy wieloagentowej, poza obudowującym je interfejsem graficznym.

4.3. Implementacja systemów agentowych

Agenty programowe tworzone na podstawie platformy UMAP dystrybuowane oraz uruchamiane są jako zestawy .NET (ang. *.NET Assemblies*). Zestawy obejmujące klasy agentów zawierają w sobie referencje do szkieletu klasy agenta dostarczonego wraz z platformą UMAP, co umożliwia ich identyfikację oraz uruchomienie pod kontrolą kontenera. Implementacja konkretnego agenta możliwa jest w jednym z języków kompilowanych do kodu pośredniego (IL, ang. *Intermediate Language*), spośród których najbardziej rozpowszechnione to C#, Visual Basic.NET oraz C++/CLI.

Platforma UMAP została zaimplementowana na podstawie platformy programistycznej Microsoft.NET w wersji 2.0. Platforma .NET posiada obecnie dwie implementacje w pełni zgodne ze standardem 2.0 – Microsoft.NET oraz projekt MONO [13]. Systemy zbudowane na podstawie platformy UMAP mogą więc działać w systemach UNIX-owych.

Na rys. 4 przedstawiono przykładową implementację agentów prowadzących wzajemną konwersację, uruchomionych pod kontrolą platformy programowej MONO.



Rys. 4. Agenci prowadzący wymianę komunikatów, pracujący na platformie MONO
Fig. 4. Agents exchanging messages, working at MONO platform

4.4. Interfejs agenta UMAP

Celem utworzonej platformy UMAP jest umożliwienie zbudowania na jej podstawie gotowych systemów opartych na agentach. Dlatego też istotne jest precyzyjne określenie wspólnego interfejsu klasy agenta, na którym wzorowane będą powstające implementacje. Interfejs agenta UMAP przedstawiono na listingu 1.

W przedstawionym rozwiązaniu agent jest klasą abstrakcyjną, w której należy zaimplementować cztery metody, z czego tylko dwie są obowiązkowe. Utworzenie własnego agenta sprowadza się zatem do implementacji logiki postępowania przy odbiorze wiadomości (metoda `HandleMessage()`) oraz, jeśli to konieczne, do zaimplementowania działań agenta (metoda `Run()`). Opcjonalnie możliwe jest również zdefiniowanie zachowania w momencie tworzenia, przerywania działania procesu agenta.

```
public abstract class Agent
{
        /// <summary>
        /// Override this method to perform agent initialization
        /// when it is being activated
        /// </summary>
        protected void OnActivate() { }

        /// <summary>
        /// Override this method to perform cleanup
```

```
        /// before agent thread is terminated
        /// </summary>
        protected void OnDeactivate() { }

        /// <summary>
        /// Override this method to impelment message processing logic
        /// </summary>
        /// <param name="message"></param>
        protected abstract void HandleMessage(Message message);

        /// <summary>
        /// Override this method to implement all continous agent jobs
        /// </summary>
        public abstract void Run();
}
```

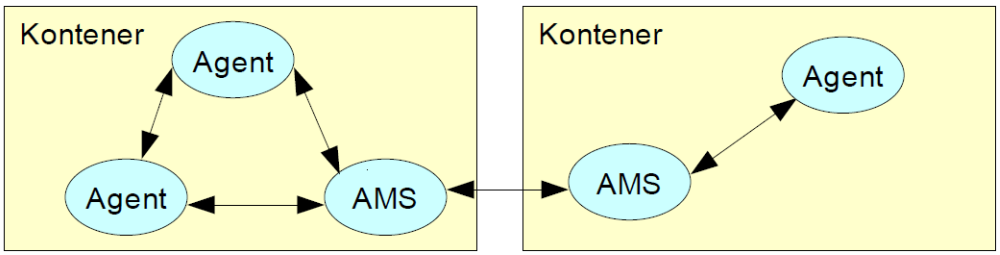
Listing 1. Klasa abstrakcyjna *Agent*

Tak skonstruowany interfejs umożliwia dowolną implementację agenta, jako programu działającego w tle aplikacji okienkowej, czy też obudowania istniejącej już aplikacji (ang. *wrapper*). Twórca logiki agenta nie musi skupiać się na implementacji przesyłania wiadomości czy nadzoru pracy agentów, dzięki czemu może skupić się na logice systemu.

4.5. Środowisko uruchamiania agentów

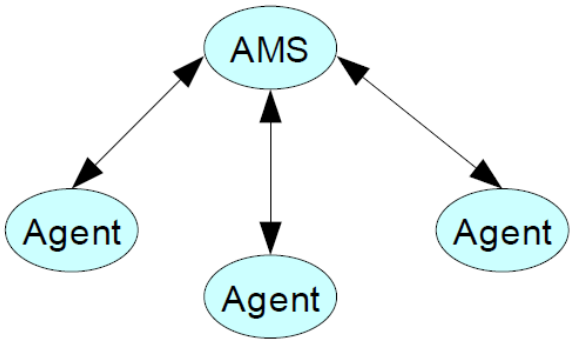
Podczas projektowania systemu wieloagentowego należy określić sposób, w jaki uruchamiane będą agenty. Wybór odpowiedniego sposobu działania programu agenta jest bardzo istotny, gdyż określa możliwość komunikacji agenta z systemem zarządzania agentami, zdolność do komunikacji wewnątrz kontenera oraz przede wszystkim możliwość kontroli pracy programu agenta przez kontener, z którym jest logicznie powiązany. Analiza istniejących rozwiązań pokazuje nam co najmniej dwa podejścia do problemu uruchamiania agentów.

Pod kontrolą platformy JADE, klasy agentów są ładowane z podanych plików, a ich uruchomienie następuje w procesie potomnym wobec kontenera, do którego należą. Podejście takie zapewnia pewną kontrolę nad poprawnością ładowanego agenta, daje możliwość bardzo wydajnej komunikacji (przez pamięć współdzieloną), a także daje pełną kontrolę nad działającym agentem. Schemat działania agentów na platformie JADE przedstawiono na rys. 5 (dla uproszczenia elementy składowe systemu oznaczono jako jeden element AMS). Kontrola nad procesem czy wątkiem, w którym działa agent, jest sprawą bardzo istotną, gdyż dokumentacja FIPA wymaga, aby kontener był w stanie kontrolować, a w szczególnych przypadkach przerywać pracę agenta niezależnie od jego działania. W związku z tym niezbędna jest taka kontrola nad agentem, która pozwoli na niezależne od implementacji i natychmiastowe przerwanie pracy agenta.



Rys. 5. Agenci pracujący pod kontrolą kontenerów platformy JADE
Fig. 5. Agents working under control of containers on JADE platform

Odmienne podejście przyjęte zostało przez twórców platformy SPADE, gdzie uruchamianie agenta nie jest częścią zadań platformy. Agenci to programy działające niezależnie od platformy czy kontenera, do którego logicznie należą, jedynym wymaganiem jest sposób komunikacji z kontenerem. Podejście takie daje większą elastyczność co do implementacji agentów i ich lokalizacji na fizycznych maszynach, jednak ogranicza to możliwość kontroli działania agentów, komplikuje także realizację wydajnej komunikacji agentów działających na tym samym komputerze, co jest istotną kwestią, jeśli chodzi o wydajność systemu. Podejście takie zostało przedstawione na rys. 6.

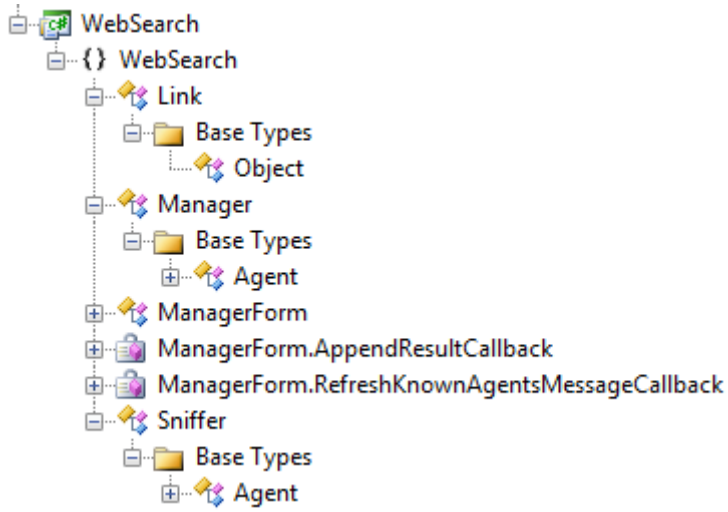


Rys. 6. Niezależni agenci uruchomieni na platformie SPADE
Fig. 6. Independent agents working on SPADE platform

Biorąc pod uwagę możliwości i ograniczenia platformy .NET, zdecydowano się na przyjęcie strategii znanej z platformy JADE, czyli uruchamiania agentów pod kontrolą działającego kontenera. Problemem, który należało jednak rozwiązać, jest w tym przypadku mechanizm tworzenia obiektów klas, których definicji czy nawet deklaracji nie znamy w momencie kompilowania. Należało również określić sposób komunikacji tak utworzonych obiektów z kontenerem, w obrębie którego działają.

Aby umożliwić działanie agentów w podobny sposób jak to ma miejsce w przypadku platformy JADE, wymagane było zaprojektowanie oraz zaimplementowanie systemu zarządzania agentami AMS, zdolnego do pobierania oraz uruchamiania klas z bibliotek ładowanych w czasie działania aplikacji. Określone zostały sposób tworzenia biblioteki kompatybilnej z systemem UMAP oraz abstrakcyjna klasa bazowa agenta, definiująca wspólny interfejs umożliwiający komunikację agenta z platformą.

Przykład klas dostarczonych przez bibliotekę DLL, przystosowaną do współpracy z platformą UMAP, przedstawiono na rys. 7.



Rys. 7. Projekt biblioteki kompatybilnej z UMAP
Fig. 7. Project of library compatible with UMAP

Należy wyróżnić tutaj dwie klasy (`Manager` oraz `Sniffer`) dziedziczące z klasy bazowej `Agent`, które rozpoznane będą jako klasy agentów, co umożliwia ich uruchomienie pod kontrolą platformy UMAP. W celu identyfikacji klasy spełniającej wymagania systemu agentowego, spośród klas dostarczonych w bibliotece wybierane są te, których klasą bazową jest klasa abstrakcyjna `UMAP.Core.Agent`. Zbiór wszystkich klas agentów jest przechowywany przez lokalny system zarządzania agentami AMS.

Aby utworzyć agenta na podstawie klasy pobranej z zewnętrznej biblioteki, posłużono się mechanizmem .NET Reflections, który udostępnia metody tworzenia obiektów na podstawie opisu typu.

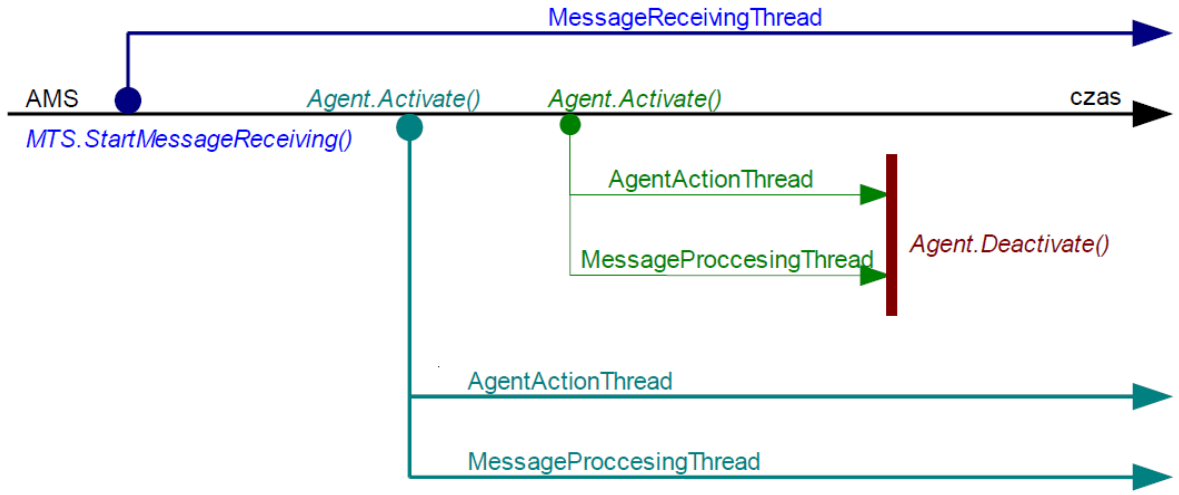
Po utworzeniu agenta należy zapewnić możliwość realizacji jego zadań zaimplementowanych w metodzie `Run()`, a także przetwarzania wiadomości przez metodę `HandleMessage()`. Ponieważ w obrębie jednego kontenera działa wielu agentów, każdemu z nich należy umożliwić wykonywanie akcji, co można by zrealizować na przykład za pomocą pętli przedstawionej na listingu 2.

```
foreach (Agent agent in this._agents)
{
    agent.Run();
}
```

Listing 2. Szeregowe wywołanie akcji agentów

Rozwiązanie takie jest jednak bardzo niepraktyczne, gdyż nie zapewnia równego dostępu do czasu procesora, oraz niewydajne. Przedstawione podejście nie rozwiązuje także problemu przetwarzania wiadomości przez agentów. W platformie UMAP zdecydowano się na uru-

chamianie agentów w osobnych wątkach. Dla każdego agenta tworzone są dwa wątki: wątek obsługi wiadomości i wątek, w którym wykonuje się akcja zaimplementowana w ciele metody `Run()`. Przebieg tworzenia wątków dla agentów UMAP przedstawiono na rysunku 8.



Rys. 8. Tworzenie wątków kontenera platformy UMAP
Fig. 8. Creation of threads for container of UMAP platform

Wątki tworzone są w momencie wywołania metody `Agent.Activate()` i działają do momentu wywołania metody `Agent.Deactivate()`.

Jak można zaobserwować na rysunku 8, kontener platformy UMAP tworzy również dwa własne wątki: jeden, służący do obsługi aplikacji okienkowej zarządzającej kontenerem, oraz drugi wątek, który odbiera wiadomości od zdalnych kontenerów. Ponadto, dla każdego agenta tworzone są także dwa wątki: przetwarzania nadchodzących wiadomości oraz realizacji zadań cyklicznych agenta.

4.6. Komunikacja pomiędzy agentami

Usługi transportu wiadomości są jedną z głównych funkcji narzędzi służących do tworzenia systemów wieloagentowych. Jest to, obok środowiska uruchamiania, główna funkcjonalność każdej platformy. Sposób komunikacji pomiędzy agentami działającymi pod kontrolą platformy UMAP zbudowany jest na podstawie odpowiadających wytycznych standardu FIPA. System transportu wiadomości platformy UMAP przewiduje przesyłanie wiadomości przez pamięć współdzieloną lub przez protokół TCP/IP, z wykorzystaniem XML. Rozwiązanie takie umożliwia wydajną komunikację pomiędzy agentami działającymi w obrębie jednego kontenera oraz, zgodną ze standardami FIPA, metodę komunikacji pomiędzy agentami działającymi na zdalnych maszynach, również w środowiskach heterogenicznych. Komunikaty pomiędzy agentami UMAP przesyłane są jako instancje klasy `Message`. Struktura klasy `Message` została wygene-

rowana na podstawie stworzonego schematu XML Schema, co umożliwia jej serializację do formatu XML z użyciem wbudowanych mechanizmów platformy .NET.

Przykład wiadomości zapisanej w formacie XML znajduje się na listingu 3.

```
<?xml version="1.0" encoding="utf-8"?>
<message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="UMAP:ACL">
  <performative>inform</performative>
  <sender>
    <name>bob@container_1/UMAP</name>
  </sender>
  <receivers>
    <AgentID>
      <name>alice@container_2/UMAP</name>
    </AgentID>
  </receivers>
  <reply_to>
    <name>bob@container_1/UMAP</name>
  </reply_to>
  <content>Hello Alice</content>
  <encoding>UTF-8</encoding>
  <ontology>simple chat</ontology>
  <conversation-id>8bac057a-85b7-4dd3-8375-e9ca15985e40</conversation-id>
</message>
```

Listing 3. Przykładowa wiadomość w formacie XML

5. Podsumowanie

Systemy wieloagentowe stosowane są w wielu dziedzinach, a ich rozwój związany jest ze standaryzacją pewnych zachowań agentów, tak aby możliwa była współpraca programów osadzonych w różnych środowiskach. Podczas tworzenia platformy wieloagentowej UMAP postawiono sobie za cel dostarczenie szkieletu systemu, który upraszcza implementacje agentów, oferując gotowe środowisko, w którym agenci działają oraz komunikują się ze sobą. Bardzo ważne było tutaj założenie nieograniczania się do specyficznego obszaru wykorzystania agentów. Dzięki temu, na podstawie gotowej platformy, możliwe jest utworzenie rozwiązań z dziedziny gromadzenia wiedzy, wspomagania decyzji czy też asystentów personalnych. Równie ważnym aspektem przy projektowaniu oraz wykonaniu systemu było opracowanie jasnego i zrozumiałego dla użytkownika (programisty) interfejsu, dzięki któremu możliwe jest tworzenie własnych agentów. Ograniczenie kroków niezbędnych do implementacji metod agenta ma na celu wprowadzenie pewnej warstwy abstrakcji, tak aby nie było konieczne skupianie się na szczegółach implementacji platformy wieloagentowej.

Projekt systemu odzwierciedla architekturę zaproponowaną przez standard FIPA, z uwzględnieniem zarządzania agentami, systemu transportu wiadomości oraz usługi katalogowania agentów. System zarządzania agentami został rozwiązany jako prosta aplikacja sterująca środowiskiem wykonywania agentów.

BIBLIOGRAFIA

1. Shoham Y., Leyton-Brown K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, 2009.
2. Nwana H.S.: Software Agents: An Ovierview. Knowledge Engineering Review, Vol. 11, No. 3, September 1996, s. 1÷40.
3. Wooldridge M.: An Introduction to Multiagent Systems. 1st edition, John Wiley & Sons, 2002.
4. Padgham L., Winikoff M.: Developing Intelligent Agent Systems: A Practical Guide. Halsted Press, New York, USA 2004.
5. Standard FIPA – Agent Management Specification. http://www.fipa.org/specs/fipa00023/ (Stan na 18.10.2010).
6. Standard FIPA – Abstract Architecture Specification. http://www.fipa.org/specs/ /fipa00001/ (Stan na 18.10.2010).
7. Standard FIPA – Message Structure Specification. http://www.fipa.org/specs/fipa00061/ (Stan na 18.10.2010).
8. Standard FIPA – Communicative Act Library Specification. http://www.fipa.org/specs/ fipa00037/ (Stan na 18.10.2010).
9. Bellifemine F., Caire G., Poggi A., Rimassa G.: JADE, A White Paper. http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf (Stan na 18.10.2010).
10. Strona domowa projektu SPADE2. http://code.google.com/p/spade2/ (Stan na 18.10.2010).
11. Opis standardu XMPP. http://xmpp.org/about-xmpp/technology-overview/ (Stan na 18.10.2010).
12. Podręcznik użytkownika platformy FIPA-OS. http://fipaos.sourceforge.net/docs/Developers_Guide.pdf (Stan na 18.10.2010).
13. Strona platformy, http://www.mono-project.com/ (Stan na 12.04.2011).

Recenzenci: Prof. dr hab. inż. Henryk Rybiński
Dr inż. Tomasz Traczyk

Wpłynęło do Redakcji 31 stycznia 2011 r.

Abstract

Multi-agent systems allow to build custom software solutions that work on the basis of groups of cooperating agents, implementing a common goal. Actually, multi-agent systems play the specific role in many decision support systems (expert systems), where the data necessary to make a decision are scattered among databases, which differ in the domain of stored information. Many solutions in the applied industrial informatics is also based on agents, because such systems are fault tolerant and able to operate on incomplete data, and in special cases, make the right decisions, even if they receive incorrect data.

The greatest potential for multi-agent systems results from the fragmentation of knowledge and functions between its various components. This makes it possible to solve complex problems using domain-specific resources, or systems that solve different classes of problems. However, the challenge does not remain in the integration, but in mutual communication between different cooperating systems. This can be achieved by creating standards for methods and formats of communication of these systems and agents.

In the paper, we present a new, universal multi-agent platform (UMAP), which provides ready to use solutions for the implementation of software agents, leaving the programmers only the necessity to implement business logic of agents, according to the expected deliverables. UMAP platform provides a unified agent skeleton, which constitutes a base for any agent solution. UMAP agent has a fixed interface that enables communication and management of agents within the platform. We also show theoretical considerations related to the functioning of the UMAP platform, its architecture and runtime environment for software agents working under the control of the UMAP platform.

Adresy

Igor WALIGÓRA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, igor.waligora@gmail.com.

Bożena MAŁYSIAK-MROZEK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, bozena.malysiak@polsl.pl.

Dariusz MROZEK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, dariusz.mrozek@polsl.pl.