9.

Pułapki tworzenia oprogramowania agentowego na przykładzie Systemu Wspomagania Podróży

**Maciej Gawinecki, Marcin Paprzycki,
Maria Ganzha**

9.1. Agentowy System Wspomagania Podróży

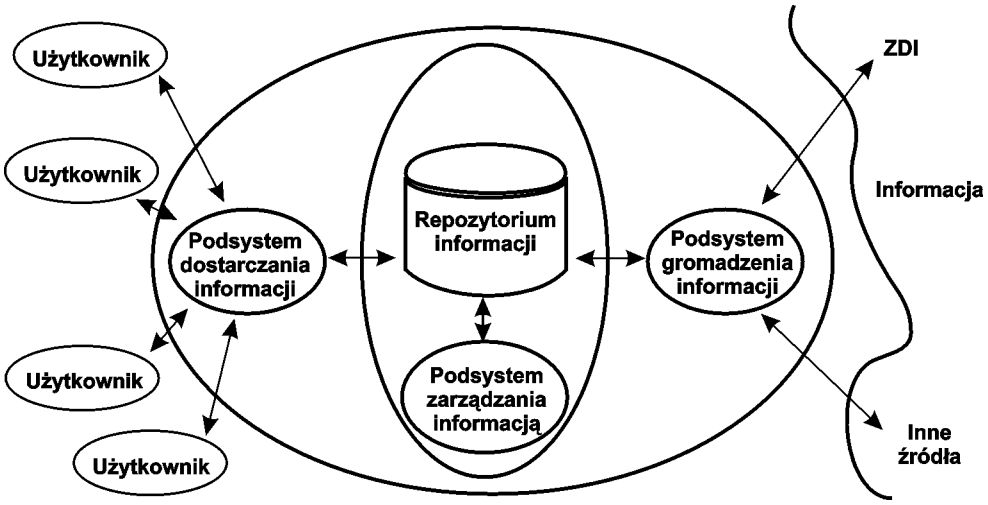
Stworzenie systemu wspomagania podróży jest jednym ze standardowych celów agentowego podejścia do tworzenia oprogramowania [30, 28]. Wiąże się to z naturalną metaforą jaką stanowi agent pracujący w biurze podróży. Typowy scenariusz z jakim mamy do czynienia to:

Hanna Krzemińska, pracująca w (małej) firmie Lupsis w Krakowie otrzymuje polecenie wyjazdu służbowego do Dijon. W tym celu musi kupić bilet lotniczy (Hanna nie ma sekretarki, która by za nią zorganizowała wyjazd), zarezerwować pokój w hotelu, etc. Hanna mogłaby udać się do biura podróży i zapłacić wysoką prowizję za usługę, lub skorzystać z systemu agentowego. Hanna wybiera tę drugą opcję, łączy się z agentem osobistym, informuje go o parametrach wyjazdu (jutro rano do Dijon) i zajmuje się przygotowaniem do spotkania z klientem. W tym czasie agent pracuje autonomicznie i kiedy zakończył swoją pracę dostarcza wynik Hannie: (1) rezerwacje na samolot do Paryża, (2) informację o sposobie dotarcia z lotniska CDG do dworca Lyon w Paryżu, (3) rozkład jazdy pociągów Paryż–Dijon, (4) rezerwacje w hotelu Kyriad Miranda, (5) rezerwacje w restauracji La Bonne Fourchett, oraz darmowy aperitif przy zamówieniu menu d'jour, (6) odpowiednie mapy (Paryża i Dijon).

Wyjaśnieniem tego scenariusza, stanowiącego punkt wyjścia dla tworzenia agentowych systemów wspomagania podróży, jest następujące: Tak samo, jak agent w biurze podróży uczy się wymagań klienta, tak samo system rekomendujący uczy się wymagań użytkownika. Tak więc system ten wie już, że gdy w Dijon, Hanna zatrzymuje się w hotelu Kyriad Miranda i lubi dania serwowane w restauracji La Bonne Fourchette – dzięki swoim umiejętnościom może wynegocjować darmowy aperitif dla swojej użytkowniczki.

Interesujące, że wprawdzie bardzo często system taki jest opisywany jako cel badań, to ciągle pozostaje on marzeniem. W roku 2001 przeprowadziliśmy analizę zawartości stron internetowych i literatury na temat agentowych stron wspomagania podróży i okazało się, że większość projektów polegała na stworzeniu niewielkiego fragmentu systemu, a następnie została porzucona [25]. Typowym przykładem może być projekt CRUMPET (zorientowany na wspomaganie podróżnego przez dostarczanie map do podręcznych urządzeń – takich, jak na przykład PDA), który był finansowany w latach 1999–2002 przez Unię Europejską. Projekt ten zakończył się sukcesem [35], ale trudno jest doszukać się jego długotrwałych efektów; np. strona WWW projektu już nie istnieje.

Podjęliśmy więc prace, mające prowadzić do stworzenia agentowego systemu wspomagania podróży, którego głównym celem była personalizacja dostarczanej informacji. Prace nad projektem zaczęły się w 2001 r. [3, 4, 9]. Obecna architektura systemu, będąca wynikiem procesu ewolucyjnego, została przedstawiona na rysunku 9.1 (szczegółowe omówienie znaleźć można w [18]).



Rysunek 9.1. Architektura Systemu Wspomagania Podróży

Źródło: opracowanie własne.

Determinują ją trzy funkcjonalności (realizowane przez osobne podsystemy):

1. *gromadzenie informacji* – agregacja informacji potrzebnych w systemie od zaufanych dostawców i ze źródeł internetowych (*podsystem PGI*),
2. *zarządzanie informacją* – czuwanie nad poprawnością i aktualnością zgromadzonych danych (*podsystem PZI*),
3. *dostarczenie informacji* – dostarczanie użytkownikowi spersonalizowanej informacji, spełniającej określone przez niego kryteria (*podsystem PDI*).

9.2. Faza planowania systemu

Zanim opiszemy, co i w jaki sposób realizują poszczególne funkcjonalności, przyjrzyjmy się ogólnie fazie planowania systemu, kiedy to projektant podejmuje decyzję o wyborze konkretnej metodologii, w tym abstrakcji do modelowania problemu. W przypadku podejścia agentowego należy pamiętać, że jest to abstrakcja stosunkowo młoda i może zawieść w tych przypadkach, gdzie tradycyjne podejścia (architektura klient-serwer, paradygmat programowania obiektowego, etc.) i technologie (Web Serwisy, Java RMI, Systemy Zarządzania Treścią – CMS, etc.) mają swoje zasłużone miejsce, potwierdzone długą praktyką przemysłową. Wielu projektantów naiwnie twierdzi, że agenci są „złotym środkiem" w dziedzinie tworzenia oprogramowania, podczas gdy argument ten w znacznej mierze pozostaje niezweryfikowany w praktyce [40]. Dlatego też autorzy Systemu Wspomagania Podróży postanowili zweryfikować powyższy argument empirycznie, przeprowadzając eksperyment. Polegał on na wykorzystaniu agentów do realizacji niemal wszystkich funkcjonalności. Wróćmy teraz do szczegółowego opisu systemu.

9.3. Szczegółowa architektura systemu

Podsystem gromadzenia informacji, zajmuje się dostarczaniem informacji wykorzystywanej i przetwarzanej w pozostałych podsystemach. Wiarygodność informacji gwarantują *Zweryfikowani Dostawcy Informacji* (*Verified Content Providers*), do których wliczamy dostawców oferujących treść w ustalonej formie (w postaci stopek RSS) oraz zweryfikowane strony WWW, z których informacja jest ekstrahowana okresowo [10, 11]. Drugorzędną rolę pełnią *inne źródła internetowe*. Ich liczba oraz różnorodność formy i treści ogranicza możliwości skutecznego wykorzystania zawartej tam informacji. Ponadto należy wziąć pod uwagę, że mogą one pojawiać się i znikać, raptownie zmieniać formę, jak również zawierać nieprawdziwe dane. Zebrane i przetransformowane do formy semantycznie

ustrukturalizowanej (OWL) dane trafiają do *Repozytorium Informacji* (wykorzystującego technologię Jena [22]). W chwili obecnej *Repozytorium* zawiera informacje o ponad 9856 lokalach gastronomicznych w Polsce. Dane te pochodzą z serwisu *Chefmoz dining guide*, gdzie zostały udostępnione w formie pliku RDF/XML. Przy tak ogromnej ilości informacji w pliku nie ustrzeżono się błędów semantycznych i syntaktycznych, z których tylko te drugie, dzięki możliwości zautomatyzowania procesu, udało nam się naprawić [15]. Serwis nie udostępnia bezpośrednio żadnej ontologii (nazywanej dalej ontologią domeny, czy też ontologią dziedziny wiedzy), definiującej sposób przetwarzania danych, a jedynie krótki opis użytych konceptów. W związku z tym, opierając się na dostarczonych danych, dokonaliśmy odtworzenia potrzebnej ontologii [14, 15]. Jej fragment opisuje rysunek 9.2. Aby zapewnić zgodność z ontologią, dane zostały przeformatowane. Zaprezentujemy teraz opis restauracji w Dijon (w notacji N3):

```
:France_La_Bonne_Fourchette_Restaurant
  a res:Restaurant ;
  loc:city "Dijon" ;
  loc:country "France" ;
  loc:streetAddress "52, rue Berbisey" ;
  loc:phone "03 80 30 77 16" ;
  res:URL
   "http://www.linternaute.com/restaurant/restaurant/7150/la-bonne-fourchette.shtml" ;
  res:cuisine
    ccd:Regional, ccd:Cafeteria ;
  res:alcohol
    alc:WineBeer ;
  res:feature
    ftc:ExtensiveWineList,
    ftc:Winery, ftc:WineTasting .
```

Podsystem Zarządzania Informacją (PZI) zarządza *Repozytorium Informacji*. Celami jego działania są między innymi:

- uzupełnianie niepełnych informacji, na przykład brakujący numer telefonu lokalu,
- podejmowanie decyzji w przypadku informacji sprzecznych, na przykład dwa numery telefonu dla tej samej restauracji,
- aktualizacja danych podatnych na zmiany, na przykład repertuar kina czy godziny otwarcia muzeum.

Celem *Podsystemu Dostarczania Informacji* (PDI) jest dostarczanie użytkownikowi spersonalizowanej informacji odpowiadającej jego zapytaniu. Jest to w chwili obecnej najlepiej rozwinięta część projektu i dlatego poświęcimy jej oddzielny podrozdział.

9.4. Architektura Podsystemu Dostarczania Informacji

Podsystem ten realizuje dwa scenariusze: (1) dostarcza formularz wyszukiwania restauracji oraz (2) znajduje restauracje odpowiednio do podanych kryteriów, spersonalizowane pod kątem profilu użytkownika. Funkcjonalności te są realizowane przez następujących agentów:

Agent Proxy (*PrA*). Dla każdego medium istnieje Agent Proxy, stanowiący zewnętrzny interfejs systemu. Do jego zadań należy: bezpośrednie przyjmowanie żądania użytkownika, przetłumaczenie żądania na formę wiadomości ACL, przekazanie wiadomości do *Agenta Obsługi Sesji*; następnie odebranie od niego wyników i wysłanie odpowiedzi z powrotem do klienta. Taka funkcjonalność pozwala na uniezależnienie pracy systemu od rodzaju medium klienta.

Agent Obsługi Sesji (*AOS*). Wszystkie żądania użytkownika przechodzą przez *PrA* do *AOS*, który jest odpowiedzialny za realizację żądania użytkownika (przygotowanie odpowiedzi) lub delegowanie innego agenta (w tym *Agenta Osobistego*) do realizacji tego zadania. Ponieważ odpowiedzi, które ostatecznie zwraca *AOS*, mają być personalizowane, konieczne jest gromadzenie informacji na temat zachowania użytkownika. Dlatego naturalne stało się wydelegowanie *AOS* do gromadzenia tych informacji. Informacje te udostępnia *AOS Agentowi Zarządzającemu Profilami*.

Agent Osobisty (*AO*). Obsługuje on żądanie użytkownika i wykorzystuje do tego również innych agentów. W planowanym rozwoju systemu ma on być odpowiedzialny za personalizację zwracanej informacji.

Agent Serwisu Restauracyjnego (*ASR*). Agent zapewnia innym agentem dostęp do Repozytorium Informacji z restauracjami.

Agent Transformujący Widok (*ATW*). Agent odpowiedzialny za transformację danych semantycznych (RDF/XML) do formatu rozumianego przez medium klienckie (HTML, WML, itp.). Wykorzystuje do tego celu serwer Raccoon [27].

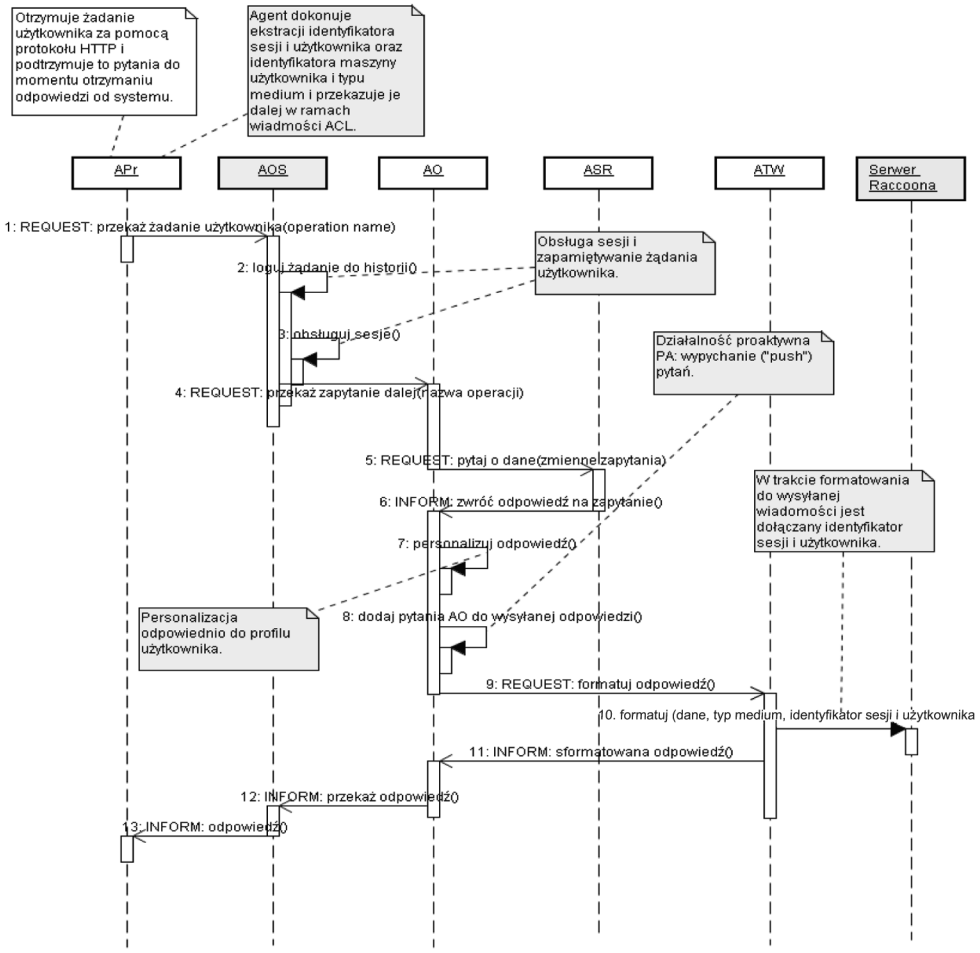
Agent Zarządzający Profilami (*AZP)*. Agent ten pełni dwie role:

1. przetwarza profil użytkownika, to jest inicjalizuje go i adaptuje,
2. przechowuje i oferuje innym agentom dostęp do danych osobistych (profili użytkowników).

Analizując możliwe sposoby komunikowania się użytkownika z systemem doszliśmy do wniosku, że możemy tylko założyć minimalne wymagania dla urządzenia: przyłączenie do Internetu i wykorzystanie protokołu HTTP. Wszystkie inne możliwości są dodatkami, ale nie mogą być wymagane [13]. W chwili obecnej możliwe jest

połączenie za pomocą przeglądarki WWW lub telefonu komórkowego obsługującego WAP. Interfejs opisany jest za pomocą języka – odpowiednio – HTML lub WML.

Celem lepszego zrozumienia działania systemu przedstawiamy na rysunku 9.2 diagram sekwencyjny opisujący interakcje między agentami przy realizacji żądania użytkownika.



Rysunek 9.2. Diagram sekwencyjny obsługi żądania użytkownika

Źródło: opracowanie własne

Załóżmy, że wspomniana wcześniej Hanna Krzemińska poszukuje w Dijon restauracji serwującej posiłki kuchni regionalnej. *APr* otrzymuje z jej przeglądarki zapytanie w formie Q*ueryString*, które tłumaczone jest przez agenta na postać rozumianą

wewnątrz systemu, a następnie przekazywane jest do *AOS*. Ponieważ Hanna jest już zarejestrowana i zalogowana w systemie, *AOS* deleguje do realizacji tego zadania jej *Agenta Osobistego*. *AO* wysyła pytanie do ASR, który wykonuje zapytanie zgodnie z zadanymi kryteriami na modelu Jeny zawierającym opisy restauracji. *ASR* odsyła listę znalezionych restauracji do *AO*, który przekazuje ją razem z dodatkowymi informacji, które mają pojawić się na stronie (m.in. dostępnymi linkami) do *AOS*. Tak przygotowane dane *AOS* każe *ATW* sformatować do postaci rozumianej przez przeglądarkę Hanny, czyli do HTML. *ATW* wykorzystuje do tego celu serwer Raccoon i zestaw szablonów. Gotowy dokument wraca do *AOS*, a dalej do *APr*, który zwraca odpowiedź do przeglądarki w ramach połączenia HTTP.

Pragniemy zaznaczyć, że przedstawiamy tutaj wyłącznie skrótowy opis systemu, potrzebny do zrozumienia opisywanych dalej problemów. Szczegółowe informacje znajdzie czytelnik w [11]. Kod, ontologie i ich dokumentację znajdują się na stronie [32].

9.5. Problemy odnotowane przy tworzeniu systemu

Omówimy teraz cztery problemy, na które natrafiliśmy przy tworzeniu systemu z użyciem agentów. Zaproponujemy też rozwiązania, które można w tych przypadkach zastosować. Równocześnie wymienione problemy i rozwiązania dają się uogólnić na klasy problemów i stanowią najważniejszy wkład tego rozdziału.

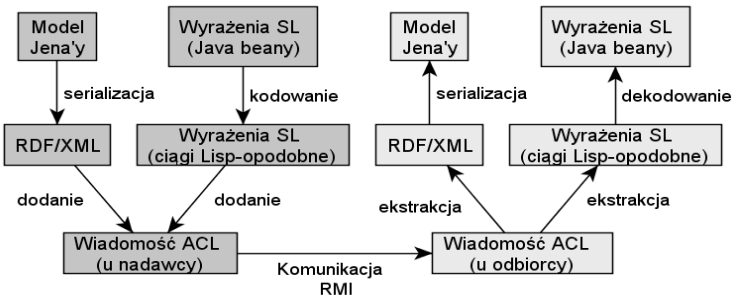
9.5.1. Wykorzystanie agentów jako wrapperów dla istniejącego oprogramowania

Wykorzystanie agentów jako wrapperów do istniejącego oprogramowania zostało już wcześniej zaproponowane jako rozwiązanie dla problemu integracji oprogramowania heterogenicznego. Jest ono charakterystyczne dla ogólnego podejścia zorientowanego na komunikację za pomocą wiadomości, pozwalającego na zwiększenie elastyczności, przenaszalności i interoperacyjności aplikacji rozproszonych [27]. W przypadku agentów takie podejście jest jednak, naszym zdaniem, usprawiedliwione tylko w jednym z dwóch przypadków: (a) nie ma innego oprogramowania pośredniczącego, które mogłoby spiąć heterogeniczne części aplikacji, lub (b) wykorzystanie agentów rozszerzy opakowywane oprogramowanie o dodatkową funkcjonalność (jak ma to miejsce przy użyciu standardowych wzorców projektowych Obserwatora lub Adaptatora [10]). Zilustrujmy zatem rozważaną sytuację przykładem z naszego systemu. Agent JADE używa dwóch semantycznie bogatych języków: SL i ACL (Precyzyjnie rzecz ujmując, SL jest formą serializacji (syntak-

tyki) dla ACL, który określa jedynie semantykę wiadomości) [8]. Dla przykładu w poniższej wiadomości agent *ASR* informuje *AO* o zamawianych restauracjach:

```
(request
  :sender ...
  :receiver ...
  :ontology tss-ontology
  :language fipa-sl
  :content"
      (result
        (action (find-restaurants :query .... ) )
        („<?xml version=„1.0"?>
      <rdf:RDF>
       <res:Restaurant rdf:ID=„
        France_La_Bonne_Fourchette_Restaurant">
         <loc:streetAddress> 52, rue Berbisey
         </loc:streetAddress>
       </res:Restaurant>
       ...
      </rdf:RDF>")
    )
  "
)
```

Wiadomość ta, zawiera także dane w formie RDF/XML, które opisują restauracje. Proces tworzenia wiadomości ACL przez *ASR* i czytania jej przez *AO* został przedstawiony na rysunku 9.3. Dane OWL opisują restauracje i są przechowywane w trwałych modelach Jena'y, z których muszą zostać zserializowane do postaci RDF/XML. Pozostała część treści wiadomości (*content slot*) jest konstruowana z użyciem Java beansów reprezentujących koncepty w ontologii wykorzystywanej do komunikacji, a następnie kodowane do postaci łańcuchów Lispopodobnych. Wszystkie powstałe elementy wiadomości komponowane są w jedną wiadomość ACL i wysyłane do *AO* z użyciem technologii Java RMI (standardowa technologia, której JADE używa do przesyłu wiadomości). *AO* czyta treść wiadomości w dokładnie odwrotnym porządku. Proces ten pochłania czas i zasoby i tym bardziej nie znajduje uzasadnienia w naszym systemie, w którym agenci nie wykorzystują w pełni właściwości języka ACL. Co więcej, jak zostało to opisanego przy okazji wrapperów, zdalne modele Jena przechowywane w bazie danych mogą być osiągalne za pomocą prostego połączenia bazodanowego, z pominięciem kosztownej serializacji tych modeli i umieszczania ich w wiadomościach ACL.



Rysunek 9.3. Proces komunikacji między dwoma agentami JADE umieszczonymi na dwóch komputerach (tzw. kontenerach) w ramach jednej platformy agentowej

Źródło: opracowanie własne.

Podejściem alternatywnym byłoby zatem (1) użycie prostych połączeń bazodanowych, zamiast interfejsowania dostępu do źródeł danych za pomocą wrapperów, (2) wprowadzenie tradycyjnych technologii takich, jak Java RMI, do delegowania żądań do zdalnych serwisów (takich, jak transformacja widoku) lub (3) jeśli to możliwe, zintegrowania współpracujących agentów w jednego agenta. Uogólniając, interoperacyjność między elementami aplikacji może być w prosty sposób osiągnięta dzięki zaimplementowanym interfejsów do tych elementów w Javie.

9.5.2. Zamiana tradycyjnych technologii na agentów

Wielu niedoświadczonych inżynierów widzi agentów wszędzie i w ten sposób, zgodnie z wizją Nwana i Ndumu [30], próbuje wykorzystać ich do realizacji wszystkich funkcjonalności aplikacji. Jest to jednak powszechny błąd, zwłaszcza jeśli część systemu może pracować z wykorzystaniem tradycyjnych rozwiązań i technologii [39]. My popełniliśmy ten błąd celowo, sprawdzając praktycznie sens wykorzystania agentów dla każdej funkcji. Przyjrzymy się zatem naszym rozwiązaniom i zaprezentujmy wynikające z nich wnioski.

Wtedy gdy w systemie występuje wiele sposobów (mediów) do interakcji z nim, naturalnym wydaje się oddzielenie warstwy prezentacji od warstwy danych. Odpowiedzią jest wzorzec projektowy *Model-Widok-Kontroler*, zaadaptowany na potrzeby systemu wieloagentowego, co przedstawia tabela 9.1. W związku z tym głównym scenariuszem systemu jest dostarczanie treści, w ramach architektury klient-serwer, gdzie system gra pasywną rolę serwera. Również w obrębie samego systemu zostało wykorzystane podejście klient-serwer, w formie protokołu interakcji *FIPA Request* [8], w ramach którego *Initiator* gra rolę klienta, a *Responder* –

rolę serwera. Na przykład, *AOS* (*Initiator*) składa żądanie do *ATW* (*Respondera*), aby ten przygotował widok dla podanego modelu danych. Oddzielna funkcjonalność została przydzielona agentowi *PrA*, któy opakowuje serwer HTTP. Wszyscy ci agenci grają rolę w zaadptowaniu wzorca Model-Widok-Kontroler (MWK) w środowisku agentowym.

Tabela 9.1: Porównanie technologii agentowej oraz tradycyjnej przy realizacji wzorca Model-Widok-Kontroler

Rozwiązanie	Model	Widok	Kontroler	Serwer HTTP
Agentowy System Wspomagania Podróży	AOS/AO przygotowuje model z czystymi danymi.	ATW opakowujący serwer Raccoon, przygotowuje widok (HTML/WML) reprezentujący model.	AOS odbiera żądania od Apr i przekazuje je do ATW, a następnie zwraca jego odpowiedź do APr.	APr opakowuje „ręcznej roboty" serwer w Javie.
Podejście tradycyjne (*Content Management System – CMS*)	Klasy z danymi + obiekty DAO + źródła danych (np. baza danych)	Procesor szablonów, np. Velocity, Freemaker	Szkielet dla aplikacji J2EE, np. Spring używający kontenera (Tomcat, Jetty, Resin)	Wydajny serwer HTTP, np. Apache.

Źródło: opracowanie własne.

Zasadniczo, wzorzec MWK wykorzystujący protokół HTTP może być opisany jako:

- bezstanowy – każde żądanie użytkownika jest traktowane niezależnie w stosunku do pozostałych, w ten sposób, że rezultaty odpowiedzi na jedno żądanie nie mają wpływu na realizację innego, analogicznie do bezstanowości protokołu HTTP;
- reaktywny – komponenty MWK pozostają nieaktywne między obsługą, zatem reagują jedynie na zewnętrzne żądania, dokładnie tak, jak *aktywne obiekty* (*active objects*) [19];
- synchroniczny – ponieważ proces realizujący żądanie użytkownika jest sekwencją kroków, z których każdy następny nie może być wykonany, jeśli poprzedni nie został zrealizowany: otrzymanie żądania HTTP, przygotowanie modelu danych, przygotowanie widoku i zwrócenie odpowiedzi HTTP;
- równoległy (*parallel*), ale nie współbieżny (*concurrent*), równoległość jest wykorzystywania do zmniejszenia przestojów między operacjami wejścia/wyjścia.

W tej sytuacji, powszechnie znane cechy agentów takie, jak proaktywność, asynchroniczna komunikacja, stanowość i współbieżność nie mogą być wykorzystane.

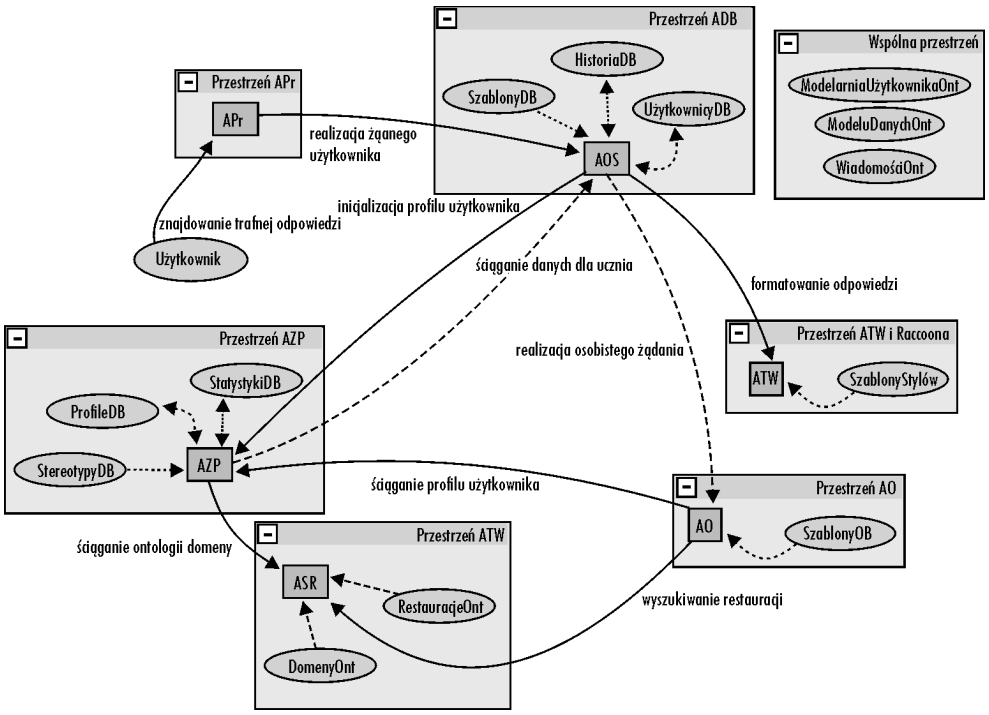
Zauważmy, że wzorzec MWK jest powszechnie realizowany za pomocą tradycyjnych technologii takich, jak Spring Framework [20], których efektywność potwierdziła praktyka przemysłowa. Zestawienie różnic architektury dla obu rozwiązań wzorca MWK – tradycyjnego (CMS) i agentowego – przedstawiono w tabeli 9.1. W przypadku wykorzystania agentów do realizacji tego wzorca pojawiły się następujące niedogodności w obszarze naszego systemu:

- problemem stała się integracja zaproponowanego podejścia z tradycyjnymi systemami zarządzania treścią (CMS), w związku z zastosowaniem niszowych technologii (Raccoon, agenci); wydaje się to szczególnie istotne, kiedy treść prezentowana użytkownikowi miałaby być skomponowana z fragmentów dostarczanych nie tylko przez agentów;
- konieczne jest opanowanie nowego nietradycyjnego sposobu projektowania stron przez dewelopera systemu;
- wprowadzone zostało rozwiązanie, które może być mniej stabilne i/lub wydajne niż całościowo przetestowane rozwiązanie komercjalne lub open source.

Uogólniając, możemy stwierdzić, że wymodelowaliśmy system za pomocą abstrakcji wyższej niż naturalnie wymagana, co zwykle skutkuje trudnościami w weryfikowaniu i rozważaniu takiego podejścia (innymi słowy: im prostszy model, tym łatwiej o nim myślimy, łatwiej weryfikować jego poprawność i usuwać błędy).

9.5.3. Rozwiązywanie konfliktów przy wiązaniu funkcjonalności

Tworzenie zbyt wielu agentów, z których każdy realizuje odmienną funkcjonalność, jest kolejną pułapką czyhającą na twórców aplikacji agentowych [39]. Zasadniczy problem, który może powstać w wyniku takich decyzji, jest potencjalny nadmiar komunikacji, spowodowany wymianą wiadomości między fizycznie odseparowanymi funkcjonalnościami [36]. Wytyczne pomocne przy rozwiązywaniu takich problemów proponuje metodologia Prometheus. Zgodnie z nią należy przy określaniu poszczególnych agentów przeanalizować, w podanej kolejności, następujące czynniki: (1) powiązania danych i wiedzy, (2) relacje między agentami i (3) częstotliwość interakcji. Funkcjonalności, które używają tych samych danych, lub współpracują ze sobą często, powinny zostać umieszczone w obrębie tego samego agenta [31]. Z drugiej strony, takie podejście kłóci się z sytuacją, kiedy zasoby lub dane, związane ze sobą jakąś funkcjonalnością, która z nich korzysta, muszą być umieszczone na oddzielnych komputerach (np. dla poprawy wydajności lub jeśli należą do różnych właścicieli). Rozwiązaniem w takim przypadku mogą być agenci mobilni, którzy pozwolą ograniczyć komunikację. Przyjrzyjmy się temu rozwiązaniu na przykładzie z naszego systemu.



Rysunek 9.4. Zależności rozpoznane w Systemie Wspomagania Podróży

Źródło: opracowanie własne.

Pierwszym krokiem w jego realizacji było zastosowanie metodologii Prometheus. Jego rezultaty czytelnik znajdzie na rysunku 9.4. Linie kropkowane oznaczają dostęp do poszczególnych baz danych i ontologii (groty strzałek wskazują dostęp na zasadzie odczytu i/lub zapisu). Linie ciągłe opisują zależności wskazanych agentów w ramach wskazanych funkcjonalności. Na przykład, AZP czyta dane z bazy danych Stereotypów i jednocześnie zapisuje i czyta dane z baz danych Profili oraz Statystyki. Co więcej, jego proces uczenia wymaga dostępu do (1) do bazy danych Historia (i w związku z tym komunikowanie się z AOS) oraz bazą danych Restauracji (opakowanych przez *ASR*). Spójrzmy na ten problem szerzej. *Filtrowanie oparte na treści* (*content-based filtering*) wymaga dużych ilości informacji: zarówno opisów rekomendowanych obiektów (tu restauracji), jak i historii reakcji_użytkownika na te obiekty (baza danych *Historia*). Postępując zgodnie z regułą integracji powinniśmy zintegrować związane ze sobą funkcjonalności *AZP*, *AOS* i *ASR* razem z dostępem do źródeł danych w ramach jednego agenta. Jednakże poczyniliśmy wyjątek dla tej reguły z następujących powodów: (1) dane restauracji mogą być udostępnione przez firmę zewnętrzną, nie należącą bezpośrednio do systemu, (2) *AZP* i *AOS* powinny funkcjonować na oddzielnych kom-

puterach, ponieważ proces uczenia wymaga ogromnych ilości zasobów procesora, podczas gdy od *AOS* oczekuje się szybkiej reakcji na żądania użytkowników. Alternatywnym rozwiązaniem byłoby wprowadzenie agenta, który uczy się profili użytkowników i przemieszcza do tej maszyny, na której przechowywane są dane wymagane w danej fazie uczenia. W takim podejściu mobilność agenta daje nam dwie zalety: (1) wzrost wydajności – przez ograniczenie nadmiaru komunikacji przy dostępie do odległych źródeł danych i (2) pewną abstrakcję na poziomie projektowania, która inżynierowi pozwala przyporządkować ustalone obliczenia do jednego punktu kontrolnego, który może być przemieszczany w obrębie systemu, zamiast narzucać mu potrzebę przekazywania tej kontroli między różnymi jednostkami rozproszonymi (jak ma to miejsce w przypadku klasycznego zdalnego wywoływania procedur).

9.5.4. Umieszczenie Agenta Osobistego w środowisku rzeczywistym

Patrząc ogólnie, zautomatyzowani osobiści asystenci są jednym ze sposobów rozumienia pojęcia agenta [26]. Perspektywa ta mówi, że każdy użytkownik jest reprezentowany w systemie przez agenta osobistego. Zauważyliśmy, że podejście takie jest usprawiedliwione jedynie, wtedy gdy agent taki funkcjonuje na maszynie klienta. Oto przyczyny, które doprowadziły nas do takiego wniosku.

W naszym systemie zdecydowaliśmy się na użycie *Agenta Osobistego* jako jednostki odpowiedzialnej za filtrowanie i personalizację danych dostarczanych użytkownikowi [29]. Pierwotnie agent taki (również zwany inteligentnym agentem interfejsu) miał funkcjonować na maszynie klienckiej lub w telefonie komórkowym. Przemawiały za tym następujące czynniki:

- bezpieczeństwo profili użytkowników, które nie są bezpośrednio dostępne dla systemu,
- naturalna mobilność – agent może towarzyszyć użytkownikowi w jego podróży, pod warunkiem, że ma ze sobą swoje urządzenie mobilne (np. telefon komórkowy),
- oddzielenie zasobów, tzn. agent zasadniczo wykorzystuje zasoby należące do użytkownika, a nie do systemu.

Jednakże, jak wspomniano, w naszym systemie zdecydowaliśmy, że urządzenia klienckie mają być lekkie. Decyzja ta jednak automatycznie wyparła oczekiwane zalety, ponieważ agent *AO* znalazł się po stronie systemu. W szczególności, profile użytkownika są przechowywane po stronie systemu. Dalej, dostęp do *AO* jest możliwy jedynie za pośrednictwem *PrA*, co oczywi-

ście ogranicza sens mobilności agenta *AO*. Ostatecznie więc wymaga to od systemu dostarczenia poszczególnym agentom *AO* dodatkowych zasobów (czasu procesora, itp.). Celem ograniczenia negatywnego wpływu ostatniego problemu zdecydowaliśmy się na powoływanie pojedynczego *AO* tylko na czas sesji z jego użytkownikiem. Dzięki temu w systemie możemy mieć zarejestrowanych ogółem 1000 użytkowników, ale w czasie, gdy tylko 100. spośród nich aktywnie komunikuje się z systemem, wymagane jest powołanie jedynie 100. agentów *AO*.

W tym miejscu rodzi się interesująca kwestia, związana z filtrowaniem kolaboratywnym, które w swej klasycznej postaci zakładało, że wszyscy zaufani agenci osobiści powinni być dostępni za każdym razem, gdy inny agent osobisty chce zapytać ich o ich opinie na dany temat (w naszym przypadku o restaurację). Podejście takie gwarantowało najwyższą trafność rekomendacji dla użytkownika. Dlatego też, użycie filtrowania kolaboratywnego lub nawet użycie agentów osobistych musi być ponownie przeanalizowane.

9.6. W jaki sposób SWP powinien zostać przeprojektowany

Przeanalizowawszy problemy, które powstały w trakcie projektowania i implementacji pierwotnej wersji systemu, zaobserwowaliśmy, że użycie tradycyjnych technologii dla niektórych jego części mogłoby być korzystne dla architektury całości. Podsumujmy więc, które funkcjonalności systemu powinny być zrealizowane przez agentów, a które za pomocą tradycyjnych technologii; oraz w jaki sposób mogłyby one ze sobą współpracować.

Agenci mogą zostać umieszczeni zgodnie z jednym ze scenariuszy:

- uczenie profilu – ponieważ wykazują się mobilnością; podejście to (1) pozwala również na organizacje agentów w drużyny agentów, gdzie każda drużyna realizuje odmienny algorytm uczenia, oraz (2) wprowadza możliwość wyboru agentów do realizacji poszczególnych zadań, ale wyboru opartego na negocjacjach, biorącego pod uwagę aktualną pozycję agenta i jego dostęp do zasobów;
- filtrowanie treści – tworzenie pojedynczego *Agenta Osobistego* dla każdego użytkownika powinno zostać zastąpione wprowadzeniem agentów reprezentujących grupy użytkowników, oraz używających odmiennych algorytmów filtrowania (tj. eksploatacji profilu).

Tradycyjne technologie powinny być wykorzystane w następujących przypadkach:

- środowisko oparte na podejściu MWK, dla przykładu Spring, które byłoby używane do dostarczania treści, realizując tym samym dotychczasową

funkcjonalność *AOS* i *ATW* (włącznie z funkcjonalnością serwera Raccoon),

- PrA opakowujący przygotowany *ad hoc* serwer HTTP zostałby zamieniony na tradycyjny kontener serwletów taki, jak Apache Tomcat, na którym rezydowałby Spring
- dostęp do zdalnych źródeł danych powinien zostać odpakowany i udostępniony w ramach prostego połączenia bazodanowego; również równoległy rozproszony dostęp do danych powinien w tym miejscu zostać rozważony (np. przez użycie polityki wielokrotnego czytania, ale pojedynczego zapisu).

9.7. Integracja technologii agentowej i tradycyjnej

Pokazano, że w Systemie Wspomagania Podróży istnieje potrzeba współistnienia obok siebie technologii tradycyjnej i agentowej. Dlatego należy przygotować odpowiednie oprogramowanie pośrednie (tzw. *Middleware*), pozwalające heterogenicznym częściom systemu na wzajemną komunikację.

Serwlety wykonywane przez *System Zarządzania Treścią* powinny mieć możliwość przekazywania komend do agentów JADE'a, na przykład za pomocą, tzw. *bramki* (*gateway*), którą może odgrywać klasa *GatewayAgent* z pakietu *jade.wrapper.gateway*.

Agenci wymagający dostępu do funkcjonalności określonego Web Serwisu (mówiącego w języku WSDL) mogą wysyłać swoje żądania do *GatewayAgent* (z dodatku Web Serices Integration Gateway add-on [21]), który będzie tłumaczył wiadomości ACL na komunikaty SOAP w obie strony i przekazywał je między Web Serwisem a agentem składającym żądanie.

Konkluzja

W rozdziale tym podsumowaliśmy wiedzę zgromadzoną przy projektowania i wstępnej implementacji agentowego Systemu Wspomagania Podróży. Pokazaliśmy, że początkowe założenie – że wszystkie funkcje systemu powinny być zaimplementowane w postaci agentów – ma wartość głównie edukacyjną. Zgromadzone doświadczenia mówią jednoznacznie, że rozwiązania agentowe należy zastosować tylko i wyłącznie tam, gdzie jest ich miejsce. Pozwoli to ominąć pułapki, które wyszczególniliśmy i przeanalizowaliśmy w niniejszym rozdziale, a w ostateczności pozwoli uczynić system bardziej realistycznym, elastycznym i wydajnym.

Podziękowania

Autorzy chcieliby podziękować za owocną dyskusję: Minorowi Gordonowi z Laboratorium Komputerowego na Uniwersytecie Cambridge w Wielkiej Brytanii, Pawłowi Kobzdejowi z Insytutu Badań Systemowych Polskiej Akademii Nauk, Mateuszowi Kruszykowi oraz Pawłowi Kaczmarkowi. Podziękowania należą się również Juanowi A. Bota Blaya z Wydziału Informacji i Inżynierii Telekomunikacji Uniwersytetu Murcia w Hiszpanii, za sugestie na temat przypadku przeciążenia komunikacji semantyką.

Bibliografia

[1] Abramowicz W., Kalczyński P.: *Building and Taking Advantages of the Digital Library for the Organizational Data Warehouse*, Southern Conference on Computing 2000, Hattiesburg, Mississipi, USA.

[2] *Agent-based Travel Support System*, http://e-travel.sorceforge.net.

[3] Ali D., Cobb M. Fiedorowicz I., Angryk Rafał, Kołodziej K., Paprzycki M. i Rahimi S.: 2001. *Development of a Travel Support System Based on Intelligent Agent Technology*. str. 243–255 z: Proceedings of the PIONIER 2001 Conference. Poznań: Wydawnictwo Politechniki Poznańskiej.

[4] Angryk R., Galant V., Gordon M. i Paprzycki M. 2002. *Travel Support System – an Agent-Based Framework*. str. 719–725 z: Proceedings of the International Conference on Internet Computing IC'2002. Las Vegas, NV, USA: CSREA Press.

[5] Bond A.H., Gasser L., eds.: *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA (1988) 12.

[6] Booch G.: *Object-oriented analysis and design with applications* (2nd ed.). Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA (1994).

[7] Bussmann S., Schild K.: *An Agent-based Approach to the Control of Flexible Production Systems. In: Procedding of the 8th IEEE International onference of Emergent Technologies and Factory Automation* (EFTA 2001), Abtibes Juan-les-pins, France (2001) 481–488.

[8] *FIPA: Foundation for Intelligent Physical Agents*. http://www.fipa.org (2007).

[9] Galant V., Gordon M. i Paprzycki M. 2002. *Knowledge Management in an Internet Travel Support System*. str. 97–104 z: Proceedings of ECON2002. Wejcherowo: ACTEN.

[10] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Massachusetts, January 1995.

[11] Ganzha M., Gawinecki M., Paprzycki M., Gasiorowski R., Pisarek S., Hyska W.: (2006) *Utilizing Semantic Web and Software Agents in a Travel Support System*. In: A. F. Salam and Jason Stevens (eds.) *Semantic Web Technologies and eBusiness: Virtual Organization and Business Process Automation*, Idea Publishing Group, Hershey, USA, 325–359.

[12] Ganzha M., Gawinecki M., Paprzycki M., Gsiorowski R., Pisarek S., Hyska W.: *Utilizing Semantic Web and Software Agents in a Travel Support System*. In: *Semantic Web Technologies and eBusiness*: *Virtual Organization and Business Process Automation*. Idea Publishing Group (2006).

[13] Gawinecki M., Gordon M., Kaczmarek P., Paprzycki M.: *The Problem of Agent-Client Communication on the Internet*. Parallel and Distributed Computing Practices 6(1) (2003) 111–123.

[14] Gawinecki M., Gordon M., Nguyen, Ngoc Thanh, Paprzycki M. i Szymczak M.: 2005b. *RDF Demarcated Resources in an Agent Based Travel Support System*. str. 271–288 z: Proceedings of the 17th Mountain Conference of the Polish Information Society.

[15] Gawinecki M., Gordon M., Paprzycki M., Szymczak M., Vetulani Z. i Wright J.: 2005a. *Enabling Semantic Referencing of Selected Travel Related Resources*. str. 271–288 z: Abramowicz W. (redaktor), BIS 2005: Proceedings of the 8th International Conference on Business Information Systems. Poznań: Poznań University of Economics Press.

[16] Gawinecki M.: *User modelling on a base of interaction with WWW system. Master's thesis, Deparment of Mathematics and Computer Science*, Adam Mickiewicz University, Poznań (2005).

[17] Genesereth M.R., Ketchpe, S.: *Software agents*. Communications of the ACM 37(7) (1994) 48–53.

[18] Gordon M. i Paprzycki M. 2005. *Designing Agent Based Travel Support System*. str. 207–214 z: Proceedings of the ISPDC 2005 Conference. Los Alamitos, CA, USA: IEEE Computer Society Press.

[19] Guessoum Z., Briot J.P.P.: *From active objects to autonomous agents.* IEEE Concurrency 7(3) (1999) 68–76.

[20] Interface21: *Spring Application Framework.* http://www.springframework.org (2006).

[21] *JADE Board, Whitestein Technologies AG: JADE Web Services Integration Gateway Guide*. http://jade.tilab.com (2006).

[22] Jena A *Semantic Web Framework for Java*. http://jena.sourceforge.net/ (2005).

[23] Jennings N.R., Wooldridge M.: *Applications of intelligent agents. In: Agent Technology: Foundations*, *Applications and Markets*. Springer, Berlin 1998.

[24] Jennings, N.R.: *Agent-oriented software engineering*. In: Proceedings of the 12th International Conference on Industrial and engineering applications of artificial intelligence and expert systems : multiple approaches to intelligent systems, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (1999) 4–10.

[25] Kalczyński P., J., Paprzycki M., Fiedorowicz I., Abramowicz W., Cobb M.: (2001) *Personalized Traveler Information System*. In: B. F. Kubiak and A. Korowicki (eds.), Proceedings of the 5th International Conference Human-Computer Interaction, Akwila Press, Gdańsk, Poland, 445–456.

[26] Laufmann S.C.: *Agent software for near-term success in distributed applications*. (1998) 49–69.

[27] Liminal Systems: Raccoon. http://rx4rdf.liminalzone.org/Racoon (2005).

[28] Ndumu D., Collins J., Nwana H. (1998) "*Towards Desktop Personal Travel Agents*," BT Technological Journal, 16 (3), pp. 69–78.

[29] Nesbitt S.: *Collaborative Filtering on the Web:* An agent-based Approach (Literature Review) (1997).

[30] Nwana H.S., Ndumu D.T.: *A perspective on software agents research*. The Knowledge Engineering Review 14(2) (1999) 1–18.

[31] Padgham L., Winikoff M.: *Prometheus: a methodology for developing intelligent agents*. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2002) 37–38.

[32] Paprzycki's M.: *Research, Software Agents in Travel Support System*, http://agentlab.swps.edu.pl/agents_TSS.html.

[33] Ramachandran V.: *Design Patterns for Building Flexible and Maintainable J2EE Applications*. http://java.sun.com/developer/technicalArticles/J2EE/despat/ (2002).

[34] Rao B.R.: *Making the most of middleware*. 12(24) (1995) 89–06.

[35] Schmidt-Belz B., Laukkanen M., Laamanen H., Veríssimo M., Zipf A., Aras H., Poslad S., *CRUMPET, Creation of User Friendly Mobile Services Personalised for Tourism*, Report at 26/03/2003., http://www.elec.qmul.ac.uk/crumpet/docs/deliverables/d4.4-final-v2.pdf.

[36] Tusiewicz M.: *System wieloagentowy: teoria, projekt, implementacja oraz przyklady zastosowań*. Master's thesis, Department of Mathematics, Physics and Computer Science, Jaggielonian University, Kraków (2003).

[37] W3C: *Resource Description Framework* (RDF). http://www.w3.org/RDF/ (2005).

[38] W3C: *Semantic Web Activity Statement*. http://www.w3.org/2001/sw/Activity (2001).

[39] Wooldridge M., Jennings N.R.: *Pitfalls of Agent-Oriented Development*. In Sycara, K.P., Wooldridge, M., eds.: *Proceedings of the 2nd International Conference on Autonomous Agents* (Agents'98), New York, ACM Press (1998) 385–391.

[40] Wooldridge M.: *An introduction to multiagent systems*. JohnWiley & Sons (2002).