POLITECHNIKA WARSZAWSKA

WYDZIAŁ MATEMATYKI
I NAUK INFORMACYJNYCH

PRACA MAGISTERSKA

KIERUNEK: INFORMATYKA STOSOWANA

„Zastosowanie agentów programowych w systemach e-commerce”

Autor:
Agnieszka Cieślik
nr indeksu: 177303
Promotor:
Prof. Marcin Paprzycki

WARSZAWA 05-2008

Spis treści.

1. Wstęp.

Celem pracy jest przedstawienie możliwości wykorzystania agentów programowych w systemach handlu elektronicznego (ang. e-commerce). Terminem handel elektroniczny określa się kupno i sprzedaż towarów lub usług przez Internet.

Agenci programowi mogą znaleźć zastosowanie na prawie każdym etapie procesu handlu elektronicznego. Począwszy od zbierania informacji, działań marketingowych oraz budowania relacji z klientem, poprzez wyszukiwanie najlepszej oferty, negocjacje warunków umowy, reprezentowanie użytkownika podczas licytacji i zakup w imieniu klienta ([7, 8])

W ramach pracy został zaprojektowany i zaimplementowany system *GS* (*Golf System*) służący do dostarczania informacji o polach golfowych. Jest on przykładem na wykorzystanie agentów w handlu elektronicznym. Poszukiwanym „towarem" jest możliwość gry na polu golfowym. W przyszłości *GS* będzie częścią powstającego Systemu Wspomagania podróży (*TSS – Travel Support System*).

TSS jest w zamierzeniu jego projektantów systemem mającym w sposób całościowy wspierać użytkownika przy planowaniu podróży. Został zaprojektowany tak, aby nie tylko służyć rozległą pulą usług: rezerwowanie miejsca w hotelu czy planowanie wycieczki, ale również żeby realizować funkcje informacyjne takie jak: dostarczanie programu kinowego lub sugerowanie restauracji z określonym rodzajem kuchni. *TSS* został zbudowany przy pomocy technologii agentowej. Wszystkie najważniejsze funkcje realizowane są przez agentów programowych. Tak więc to agenci wyszukują wartościowe, użyteczne i przede wszystkim spersonalizowane informacje. Dane w *TSS* są opisane za pomocą ontologii i przechowywane w centralnym repozytorium.

W systemie *GS,* podobnie jak w *TSS,* wszystkie ważniejsze funkcje są realizowane przez agentów programowych. Statyczne dane pól golfowych, czyli takie jak adres, typ powierzchni, architekt będą przechowywane w lokalnym repozytorium (jak w systemie *TSS*). Natomiast o informacje często się zmieniające (np. godziny w jakich możliwa jest gra) system *GS* będzie pytał źródła zewnętrzne. Wyszukiwanie informacji, komunikacja z użytkownikami i z zewnętrznymi systemami to zadania, które są realizowane przez agentów programowych.

Zakładam, że projektowany system będzie dostarczać informacje jak największej liczbie użytkowników i będzie mógł pobierać dane z jak największej liczby systemów zewnętrznych. Aby to było możliwe, system musi wysyłać i odbierać wiadomości w jednym z powszechnie używanych standardów. W chwili obecnej jednym z wykorzystywanych do wymiany informacji związanej z turystyką standardów są wiadomości OTA. Specyfikacja OTA (Open Travel Alliance) została opracowana przez liderów rynku turystyki i przedstawicieli rynku informatycznego, by dostarczyć: wspólny język terminologii związanej z podróżowaniem oraz mechanizm wymiany informacji między członkami organizacji związanych z turystyką. OTA określa zbiór wiadomości XML do wymiany danych pomiędzy różnymi systemami.

Praca magisterska składa się z dwóch części. Pierwsza część została poświęcona podstawom teoretycznym. Zawiera ona definicje agenta programowego, opisuje zastosowanie agentów i sposoby ich implementacji.

W drugiej części zawarty został opis, stworzonego w ramach tej pracy magisterskiej, systemu GS. Rozpoczyna się on od analizy wiadomości OTA. Następnie na podstawie uzyskanych wyników zostały zaprojektowane ontologie do opisu informacji o polach golfowych [1]. W dalszej części pracy zostali przedstawieni agenci programowi będący częścią projektu. Został także omówiony proces tłumaczenia wiadomości OTA na instancje zaprojektowanych ontologii i proces odwrotny, czyli tłumaczenie instancji ontologii na wiadomości OTA.

2. Pojęcie agenta programowego.

W codziennym życiu, prawie każdemu zdarza się korzystać z rad agentów przy podejmowaniu różnego rodzaju decyzji. Taki agent zazwyczaj posiada szeroką wiedzę w danej dziedzinie, np. agent ubezpieczeniowy powinien znać dostępne na rynku polisy ubezpieczeniowe, ich wady i zalety.

2.1. Definicja agenta programowego.

Pomimo, że pojęcie agenta programowego jest intuicyjnie dobrze rozumiane i często używane w literaturze, wydaje się, że brak jest jednej powszechnie przyjętej definicji. Poniżej przedstawię trzy popularne definicje.

Definicja 1.

Agenci programowi to jednostki podejmujące działania w imieniu użytkownika lub innych programów, w pewnym stopniu niezależne lub autonomiczne, które działając stosują pewną wiedzę lub reprezentację celów lub potrzeb użytkownika (IBM, 1997)

Definicja 2

Zamknięty system komputerowy znajdujący się w pewnym otoczeniu, posiadający umiejętność elastycznego działania w tymże otoczeniu, działania polegającego na wypełnieniu celów, dla jakich został stworzony (Wooldridge, 1997)

Definicja 3

Autonomiczny system znajdujący się w dynamicznym otoczeniu działający niezależnie od narzucanych przezeń ograniczeń i wypełniający w jego ramach zbiór celów lub poleceń, dla których został stworzony (Maes, 1998)

Pomimo braku ogólnie przyjętej definicji, istnieje jednak zbiór wspólnych cech, które najczęściej są wymieniane przy definicji agentów – i te cechy będą istotne w poniższej pracy:

- Zdolność do podejmowania samodzielnych decyzji,
- Umiejętność w komunikowaniu się z użytkownikiem i innymi agentami,
- Zdolność do postrzegania i reagowania na zmiany środowiska,
- Zdolność do wykorzystywania posiadanej wiedzy,
- Zdolność uczenia się.

3. Agenci w systemach e-commerce.

Agentów programowych odnaleźć można w bardzo szerokiej gamie projektów badawczych jak i wstępnych próbach implementacji praktycznych zastosowań. Celem pracy jest przedstawienie możliwości wykorzystania agentów w systemach e-commerce.

Zadania agentów w systemach e-commerce związane są z różnorodnymi funkcjami handlu Internetowego [27]:

- poszukiwanie interesującego towaru,
- negocjacja ceny zakupów,
- reprezentowanie sklepu, lub dostawcy usług

Agenci mogą działać w imieniu użytkownika, czyli klienta, osoby kupującej towar lub usługę lub w imieniu sklepu, dostawcy usług.

3.1 Agenci poszukujący danego towaru lub usługi.

Bardzo często, przed dokonaniem zakupu, użytkownik odwiedza kilka lub kilkanaście stron różnych sklepów internetowych. W sieci istnieje wiele produktów o podobnych cechach, różniących się np. producentem. Bardzo często ceny tego samego produktu w różnych sklepach internetowych, znacząco się różnią. Oczywiście istnieje już wiele stron porównujących ceny towarów, ale nadal użytkownik musi poświęcić sporo czasu na poszukiwanie i zakup on-line interesującego go produktu. Agenci programowi mogą okazać się bardzo dobrymi pomocnikami w poszukiwaniu jak i w negocjowaniu ceny towaru [27, 28]

Agenci w swoich poszukiwaniach mogą uwzględniać wiele cech produktów, nie tylko cenę. Przykładowe kryteria wyszukiwania to:
- jakość produktu,
- producent,
- koszty dostawy,
- warunki i czas gwarancji,
- promocje i prezenty,
- obsługa,
- reputacja sprzedającego.

Agent programowy może także, na podstawie obserwacji wyborów użytkownika, stworzyć jego profil [27]. Następnie, w oparciu o preferencje wynikające z profilu może ocenić ważność kryteriów wyszukiwania towaru dla danego użytkownika. Np. w sytuacji gdy dla użytkownika jest ważniejsza jakość produktu od ceny, agent może wybrać droższy produkt ale za to lepszej jakości.

Agent może uwzględniać nie tylko cenę, markę towaru, markę sklepu, ale także takie informacje jak, w których sklepach warto wyszukiwać danych towarów oraz jakie są koszty poszukiwań czyli jak długo będzie trwało wykonanie zapytania i zwrócenie rezultatów.

Agenci występujący w imieniu użytkownika z reguły rezydują na jego komputerze. Mogą go obserwować i uczyć się jego preferencji podczas robienia zakupów, a następnie informować go, gdy znajdą oferty podobnych towarów. Użytkownik może także bezpośrednio zlecić wyszukanie jakiegoś towaru i samodzielnie podać agentowi kryteria wyszukiwania, a także określić jak bardzo są dla niego ważne.

3.2 Agenci uczestniczący w negocjacjach.

Jedną z bardziej popularnych form handlu elektronicznego są, znane nam wszystkim, serwisy aukcyjne, np. bardzo popularny serwis Allegro. Korzystanie z takich serwisów wiąże się z koniecznością śledzenia aukcji, aktualnej ceny i pamiętaniem o terminie zakończenia licytacji.

Agenci programowi mogą, w imieniu klienta, brać udział w takiej licytacji, reprezentując klienta, kupca ale także sklep lub sprzedawcę. W negocjacjach agenci przygotowują i oceniają oferty w imieniu reprezentowanych klientów tak, aby zyskać jak najlepsze warunki. . Agenci mogą także monitorować strony aukcyjne i zbierać informacje, a następnie na ich podstawie określać „rozsądną" cenę za dany towar [27, 28].

3.3 Agenci rekomendujący dany produkt.

Bardzo często podczas zakupów w sklepach internetowych, po wyborze towaru, kupujący dostaje informacje z listą towarów które zostały wybrane przez innych klientów kupujących dany produkt.

Przygotowanie takiej listy może być realizowane przez agenta programowego. Zanim jeszcze potencjalny klient poczuje potrzebę nabycia jakiegoś towaru, już agent programowy może zacząć działać. Może przygotować ofertę dla danego użytkownika , powiadomić go o ciekawych dla niego produktach. Zadaniem tego typu agentów jest gromadzenie danych o użytkowniku lub grupie użytkowników, czyli budowanie ich profilu. Potem na podstawie utworzonego profilu agenci wybierają te towary lub usługi, które mogą zainteresować danego użytkownika [27, 28].

Agenci rekomendujący mogą wykorzystać technikę zwaną ”*collaborative* (lub *social*) *filtering*” [27]. Technika ta polega na identyfikowaniu podobnych użytkowników i posługiwaniu się ich wyborami lub opiniami przy rekomendowaniu produktów. Przykładem systemu agendowego był Ringo. Rekomendował on płyty CD i filmy wideo.

3.4 Agenci analizujący proces sprzedaży.

Agenci mogą być także wykorzystywani w procesie zbierania i analizowania informacji na temat sprzedaży, modelowania zachowań konsumenckich, analizowania historii zakupów. Na podstawie zgromadzonych informacji, agenci mogą utworzyć profile grup do których powinny być przeznaczone kampanie reklamowe [27].

4 Narzędzia wykorzystane w pracy.

4.1 Platforma JADE.

Dotychczasowy opis mówił o dość abstrakcyjnej idei agenta programowego. Aby była możliwa szybka implementacja systemów wieloagentowych potrzebna jest pewna platforma programistyczna, czyli struktura wspomagająca tworzenie i rozwój aplikacji agentowych.
W 2002 roku, stowarzyszenie FIPA (ang. *Foundation for Intelligent Physical Agents)* ([17]) opracowujące specyfikacje i wyznaczające standardy dla systemów agentowych, wydało zestaw dokumentów opisujących platformę do tworzenia takich systemów.

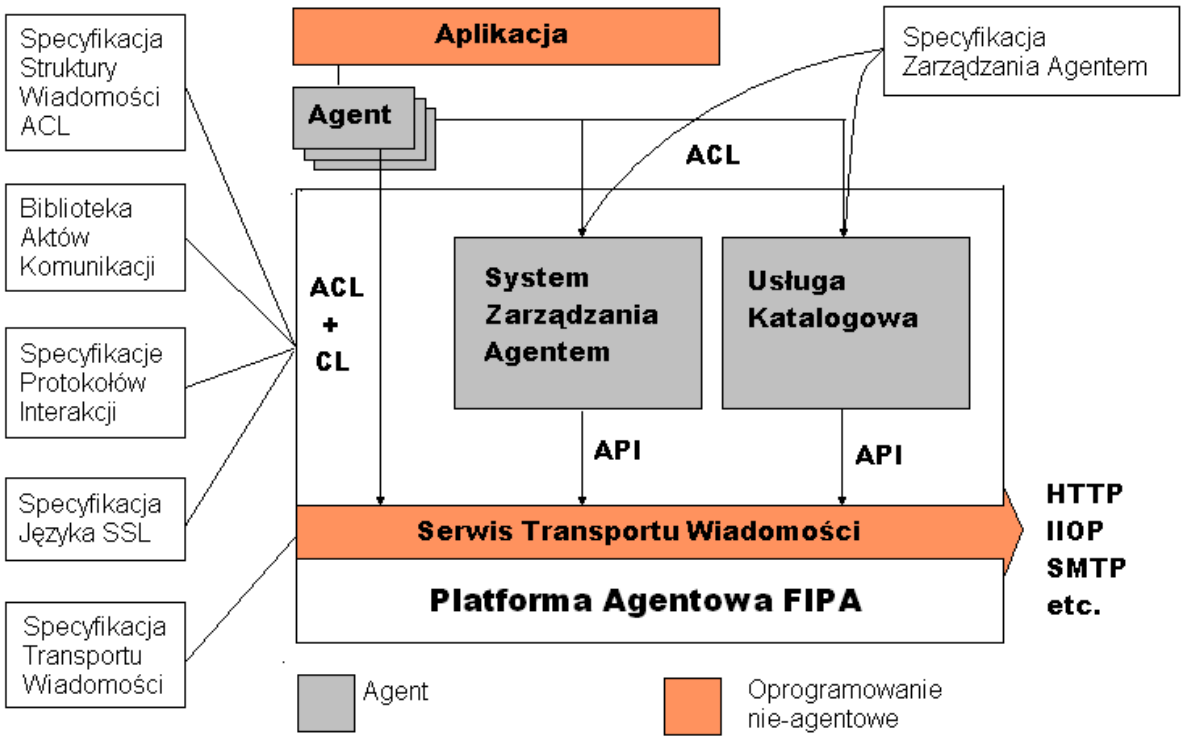
Standard FIPA wymienia komponenty, jakie powinna posiadać platforma i jakie usługi powinny być przez nie udostępniane. Opis dotyczy tego, jak powinny zachowywać się poszczególne składniki, ale nie definiuje ich wewnętrznej implementacji (standard FIPA pełni wiec podobna role jak specyfikacja J2EE dla serwerów aplikacji).

Usługi podzielone są na dwie grupy: normatywne i opcjonalne. Do wymaganych elementów platformy należą:

- System zarządzania agentami (ang. Agent Management System. AMS) zawierający mechanizmy do tworzenia, usuwania i nadzoru nad ich (agentów) działaniem, (czyli wstrzymywanie lub wznawianie działania, przenoszenie na inne platformy).
- *White Pages,* czyli usługa pozwalająca na podstawie nazwy agenta uzyskać jego identyfikator. Znajomość tego identyfikatora umożliwia innemu agentowi nawiązanie komunikacji.
- *Yellow Pages,* czyli usługa umożliwiająca agentom rejestrację zadań, jakie są w stanie wykonać w postaci nazwanych usług. Inni agenci mogą dzięki temu odnaleźć agenta, który potrafi zrealizować dany typ zadania i zlecić mu jego wykonanie.
- Usługa przesyłania wiadomości (ang., Messsage Transport Service), pełniąca rolę kanału komunikacyjnego, wykorzystywanego przez agentów do porozumiewania się między sobą.

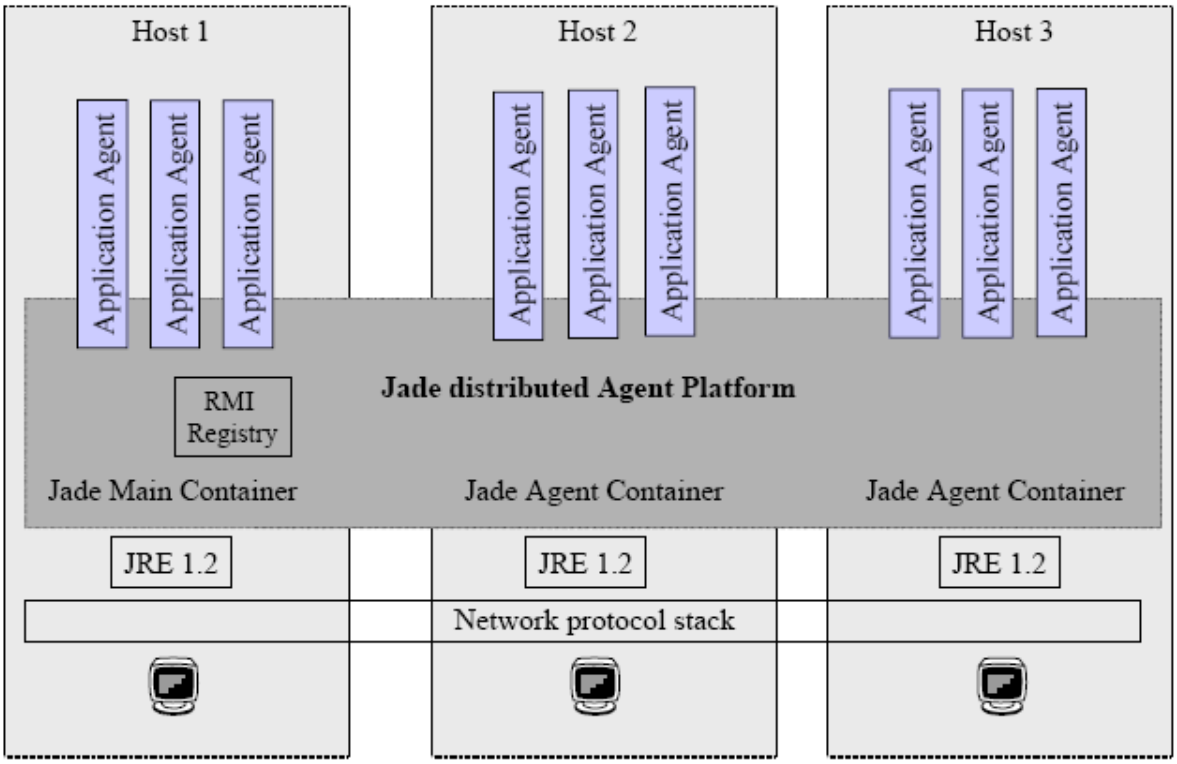
Do funkcji opcjonalnych platformy należą:

- Integracja agentów z innym oprogramowaniem,
- Usługi związane z ontologiami,
- Interakcja pomiędzy człowiekiem a agentem programowym.



Platforma Agentowa FIPA ([26])

W pracy korzystam z platformy **JADE** (ang. *Java Agent Development Framework*) ([18]). Jest to produkt typu open-source, szybko się rozwija i posiada mnóstwo dokumentacji. Środowisko to jest w pełni zgodne ze standardami FIPA. Językiem użytym do implementacji jest Java, co zapewnia temu systemowi bardzo dużą przenośność. Specyficzną cechą JADE jest kontener, czyli środowisko służące do uruchamiania agentów i zapewniające im usługi wymienione w specyfikacjach FIPA. Platforma opiera swoje działanie na współdziałaniu kilku maszyn wirtualnych Java (Java Virual Machine). Każda maszyna wirtualna jest kontenerem w obrębie platformy, w której mogą rezydować agenci. Komunikacja pomiędzy kontenerami odbywa sie przy pomocy zdalnych wywołań JAVA RMI, natomiast w obrębie jednego kontenera używa się mechanizmu sygnałów. Role kontenera dla agentów AMS oraz DF pełni kontener o nazwie *Main-Container* (Kontener Główny), który powstaje pierwszy po uruchomieniu platformy. Po starcie platformy i uruchomieniu kontenera *Main-Container*, istnieje możliwość przyłączenia pozostałych kontenerów. Kontenery te mogą być dołączane także z innym komputerów w sieci, w łatwy sposób tworząc rozproszona aplikacje.



Platforma Agentowa JADE([24])

Działający agent na platformie JADE jest osobnym wątkiem maszyny wirtualnej Javy.

Po utworzeniu agenta, platforma JADE:

- nadaje agentowi globalny identyfikator,
- rejestruje agenta w usłudze White Pages (usługa pozwalająca na podstawie nazwy agenta uzyskać jego identyfikator)
- inicjalizuje kolejkę do przechowywania nadchodzących wiadomości.

W JADE agent jest obiektem klasy dziedziczącej z jade.core.Agent, zawierającej implementację metod: setup() i takeDown(). Metoda setup() jest wywoływana przez kontener bezpośrednio po utworzeniu agenta, żeby zainicjalizować jego specyficzne właściwości. Metoda takeDown() wywoływana jest po zakończeniu pracy agenta w celu zwolnienia przydzielonych mu zasobów.

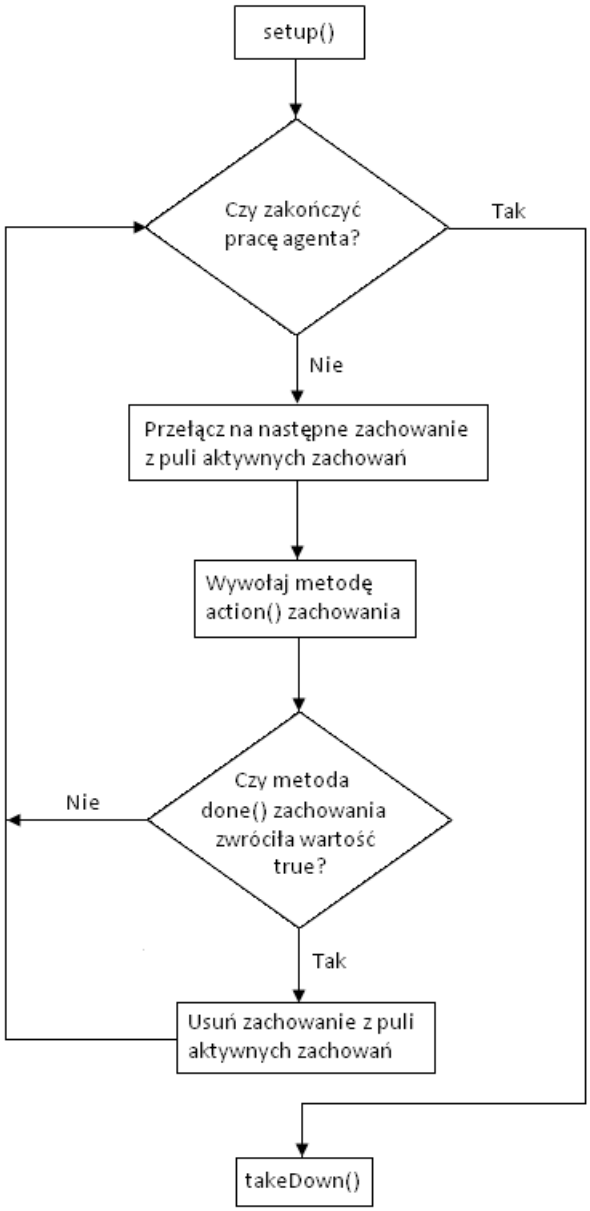
Każdy agent może wykonywać kilka niezależnych od siebie zadań, reprezentowanych przez obiekty klas dziedziczących z klasy jade.core.behaviours.Behaviour. Klasy opisujące zachowania powinny implementować dwie metody: done() i action(). Pierwsza z nich zwraca

wartość logiczną określającą, czy dane zachowanie się skończyło. Natomiast metoda action() jest wywoływana za każdym razem, gdy zachowanie jest wykonywane przez agenta.

Zadania agenta mogą być wykonywane quasi-równocześnie, tzn. w danej chwili może być realizowane przez wątek agenta tylko jedno zachowanie ale programista może spowodować przełączenie się na inne. Agent przełącza się automatycznie miedzy zachowaniami oznaczonymi jako aktywne, ale tylko wtedy, gdy obecnie wykonywane zachowanie zakończy swoje działanie lub odda kontrolę agentowi przez wywołanie metody block(). Zachowania nie podlegają wywłaszczeniu.

Cykl życia agenta składa się z faz aktywności, np. agent komunikuje się z innym agentem lub wykonuje zadane obliczenia, oraz faz pasywnych, np. agent czeka na wiadomość.

Przy nadejściu wiadomości do agenta musi zostać wyznaczone zachowanie, które powinno ją obsłużyć. Otrzymana wiadomość jest umieszczana w kolejce, w której oczekują nadesłane wiadomości. Następnie wszystkie zachowania są oznaczane jako aktywne. Każde, które spodziewa się nadejścia wiadomości sprawdza, czy jest ona przeznaczona dla niego.



4.2 Ontologia – dane zrozumiałe dla agenta.

Ontologia jest to dział filozofii zajmujący się ogólną teorią bytu, charakterem i strukturą rzeczywistości. Odpowiada na dwa podstawowe pytania:, „Co istnieje?” i „Jeśli to, co istnieje jest podzielone na części, to, jakie są te części i jakie są między nimi zależności.?”. Ontologia, zatem, definiuje obiekty, pojęcia i inne jednostki w rozważanym obszarze i określa zależności pomiędzy nimi ([11, 12, 13])

Ontologia w informatyce jest przyrównywana do hierarchii klas. Definiuje klasy i pokazuje relacje pomiędzy nimi ([13])

Przy projektowaniu ontologii wykorzystywane są metody kategoryzacji i hierarchizacji.
Pewnym pojęciom abstrakcyjnym i grupom obiektów posiadającym wspólne cechy przyporządkowywane są nazwy, tworzone są klasy obiektów (kategoryzacja). Tak utworzone klasy są następnie umieszczane w strukturze hierarchicznej. Przy tworzeniu ontologii można użyć różnych języków modelowania. Najczęściej wykorzystywany jest RDF/RDFS ([14, 15]).

RDF *(Resource Description Framework)* ([14]) jest podstawowym językiem używanym do zapisu informacji w Sieci Semantycznej i przeznaczonym do przetwarzania maszynowego. Każdy dokument RDF może być zakodowany w postaci dokumentu XML, istnieje więc możliwość łatwej jego wizualizacji.

W dosłownym tłumaczeniu, RDF jest środowiskiem do opisu zasobów. Jako zasób można rozumieć dowolny obiekt znajdujący się w sieci, pojęcia abstrakcyjne, relacje i obiekty fizyczne. Zasoby w RDF są reprezentowane przez zunifikowane identyfikatory zasobów (ang. *Uniform Resource Identifier,* URI).
URI jest pojęciem szerszym niż popularnie używane identyfikatory lokalizacji zasobów (ang. *Uniform Resource Locator,* URL). URL opisuje zasób, który musi być fizycznie dostępny w Internecie (jak strona WWW, plik na serwerze FTP) i jest szczególnym przypadkiem URI. URI może opisywać dowolny obiekt, niezależnie od tego, czy znajduje się on w Internecie czy nie. Przykładowo przy pomocy URI możemy reprezentować osobę przez nadanie jej identyfikatora URI będącego jej adresem e-mail, adresem jej domowej strony WWW lub innym identyfikatorem nadanym przez pewna organizacje -pozwalającym w sposób jednoznaczny zidentyfikować daną osobę.

Do opisu zasobów używa się tzw. stwierdzeń (ang. *statements*). Często zamiennie z terminami zdanie i stwierdzenie używane jest jeszcze określenie trójka (ang. *triple*).

To ostatnie określenie wynika z faktu, ze każde stwierdzenie w RDF ma strukturę podobną do prostego zdania w języku naturalnym i składa się z trzech elementów:

- Podmiotu *(ang. subject)*. Podmiotem jest opisywany zasób reprezentowany przez URI.

- Orzeczenia (predykatu) *(ang. predicate)*. Orzeczeniem jest nazwa cechy lub relacji dotyczącej opisywanego podmiotu i podobnie jak podmiot, orzeczenie jest reprezentowane przez URI.
- Obiektu *(ang. object)*. Obiektem może być inny zasób (reprezentowany przez URI) lub stała wartość określana mianem literału (ang. *literal).*

Najczęściej mamy do czynienia z jedna z dwóch sytuacji – opisujemy relacje miedzy dwoma zasobami (wtedy obiektem jest URI reprezentujące zasób) lub mówimy, ze podmiot ma jakąś cechę o zadanej wartości (wtedy używamy literału). Literał reprezentować może dowolną wartość o ile daje się ona zapisać w postaci ciągu znaków. W RDF literałami mogą być tylko obiekty, nie jest możliwe ich użycie w charakterze podmiotu czy orzeczenia.

Każde zdanie RDF można uważać za graf skierowany w którym podmiot i obiekt są węzłami a orzeczenie jest krawędzią skierowaną od węzła podmiotu do węzła obiektu i etykietowaną URI orzeczenia.

Podmiot: www.mojastrona.pl
Orzeczenie: http://purl.org/dc/elements/1.1/autor
Obiekt: Jan Kowalski (literał)
Zdanie: „Autorem strony internetowej www.mojastrona.pl jest Jan Kowalski.

Podmioty definiujemy znacznikiem „rdf:Description" z atrybutem „rdf:about" zawierającym URI zasobu. Zdania opisujące podmiot są definiowane jako jego węzły potomne. Obiekty, które są równocześnie zasobami zapisujemy w postaci węzła skróconego z URI tego zasobu umieszczonym w atrybucie rdf:resource.
Zdanie przedstawione w postaci grafu powyżej, zapiszemy jako:

```
<? xml version="1.0" ?>
<RDF xmls = „http://w3.org/TR/1999/PR-rdf-syntax-19990105#
```

```
      xmls: s="http://purl.org/dc/elements.1.1#>
<Description about=www.mojastrona.pl>
      <s:autor>Jan Kowalski</s:autor>
</Description>
```

.

Schematy RDF *(ang. RDF Schema, RDFS)* ([15, 16]) pełnią wobec RDF podobną rolę jak schematy XML wobec czystego XML. Sam RDF dostarcza jedynie podstawowych struktur do opisu abstrakcyjnych elementów zwanych zasobami.

RDFS umożliwia modelowanie pojęć bardzo zbliżone do modelowania obiektowego. Stanowi bazę dla języka OWL, który pozwala na jeszcze dokładniejszy zapis wiedzy. RDFSchema wprowadza takie pojęcia jak klasa i podklasa. Do zdefiniowania klas używany jest znacznik rdfs:Class, a do podklasy: rdfs:subClassOf.

RDFS posiada funkcjonalność pozwalającą na definiowanie właściwości (ang. *properties*) obiektów. Właściwości są w RDF instancjami standardowej klasy rdf:Property i mogą nie być przypisane do obiektów żadnej klasy, mogą również nie posiadać żadnego typu. Jest to jednak rzadko wykorzystywane, najczęściej dążymy do stworzenia pełnego opisu obiektu przez wyspecyfikowanie zbioru opisujących właściwości.
Przyporządkowywanie właściwości klasom jest realizowane przez znacznik rdf:domain, natomiast typ danych, jaki jest dopuszczalny dla danej właściwości definiujemy przy użyciu znacznika rdf:range.

```
<rdf:RDF xml:base="http://www.inria.fr/2007/04/17/humans.rdfs"
            xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
            xmlns="http://www.w3.org/2000/01/rdf-schema#">

  <Class rdf:ID="Man">
    <subClassOf rdf:resource="#Person"/>
    <label xml:lang="en">man</label>
    <comment xml:lang="en">an adult male person</comment>
  </Class>
  <rdf:Property rdf:ID="hasMother">
    <subPropertyOf rdf:resource="#hasParent"/>
    <range rdf:resource="#Female"/>
    <domain rdf:resource="#Person"/>
    <label xml:lang="en">has for mother</label>
    <comment xml:lang="en">to have for parent a female.</comment>
  </rdf:Property>
</rdf:RDF>
```

W przykładzie powyżej mamy zadeklarowaną klasę #Man (mężczyzna) będąca podklasą klasy #Person (osoba) i właściwość #hasMother (ma matkę). Właściwość #hasMother dziedziczy z właściwości #hasParent.
#hasMother jest właściwością klasy #Person: `<domain rdf:resource="#Person"/>`
Dopuszczalnym typem danych dla właściwości #hasMother są instancje klasy #Female (kobieta): `<range rdf:resource="#Female"/>`

Praca z informacjami zapisanymi za pomocą RDF

Jena ([19, 20]) jest jedną z najpopularniejszych bibliotek używanych przez aplikacje wykorzystujące dane zapisane w RDF. Prace nad systemem Jena rozpoczęły się w dziale badawczym firmy Hewlett-Packard, obecnie jest to oprogramowanie klasy OpenSource. Jene napisano w Javie i zapewnia ona w pełni obiektowy interfejs do dokonywania manipulacji na grafach RDF (nazywanych przez dokumentacje modelami). Modele mogą być nie tylko przechowywane
w pamięci podczas działania programu; mogą być również zapisywane do pliku. Jena umożliwia także tworzenie tzw. modeli trwałych (ang. *Persistent models*), w których trójki RDF są zapisywane w relacyjnej bazie danych na serwerze SQL takim jak MySQL, PostgreSQL czy Oracle. Ponadto Jena akceptuje ontologie stworzone przy użyciu RDFS, OWL i DAML+OIL (także ze wsparciem dla wnioskowania). Ma również klasy odpowiedzialne za wykonywanie zapytań RDQL *(ang. RDF Data Query Language),* czyli zapytań w języku podobnym do SQL, służącym do pracy z danymi opisanymi za pomocą RDF.

5. Wiadomości OTA.

Celem pracy jest stworzenie systemu GS (Golf System) służącego do dostarczania
i przechowywania informacji o polach golfowych. System taki powinien dostarczać informacje dla jak największej liczby użytkowników. Zakładamy zatem, że użytkownicy będą, podczas komunikacji z systemem, posługiwać się wiadomościami w standardzie opisanym przez, wspomnianą we wstępie, organizację OTA (Open Travell Alliance).

OTA opisuje trzy pary (request/response) wiadomości służących do wyszukiwania informacji o polach golfowych i dokonywania rezerwacji tych pól.

Wiadomości OTA:

- OTA_GolfCourseSearchRQ/RS – wiadomości o polach golfowych (adres, sposoby kontaktu, zbiór cech)
- OTA_GolfCourseAvailRQ/RS – wiadomości o dostępności pól golfowych, czyli o możliwej grze na danym polu.
- OTA_GolfCourseResRQ/RS – wiadomości o rezerwacji pola.

W pracy została wykorzystana specyfikacja OTA z roku 2006. Wszystkie, zaprezentowane w pracy, przykłady wiadomości OTA zostały pobrane
z OTA_MessageUserGuide2006V1.0 [22].

OTA_GolfCourseSearchRQ/RS

OTA_GolfCourseSearchRQ – wiadomość zawierająca zapytanie o informacje o polach golfowych. Wszystkie elementy i atrybuty (składniki wiadomości) są opcjonalne. Wiadomość zawiera listę kryteriów (cech) opisujących pole golfowe.

Każde kryterium składa się z nazwy (*Name*), wartości (*Value*) i atrybutu (*Required*) określającego, czy dane kryterium jest wymagane:

```
<Criterion Name="Singles Confirmed" Value="Yes" Required="true"/>
<Criterion Name="Architect" Value="Robert Jones" Required="false"/>
```

Czasami kryterium zawiera także atrybut „Operation”, opisujący operację porównania, na podstawie której określamy, czy kryterium jest spełnione. Np. jeśli gracz chce uzyskać informacje o polach golfowych o nachyleniu terenu mniejszym niż *(LessThan*) 110 stopni, wtedy kryterium ma postać:

```
<Criterion Name="Slope" Value="110" Required="true" Operation="LessThan"/>
```

Przykładowe kryteria:

- Architekt pola golfowego,
- Nachylenie terenu (slope),
- Informacja, czy jest dostępny caddy – pomocnik gracza, osoba nosząca torbę z kijami,
- Informacja, czy są dozwolone wózki golfowe (cart)
- Długość w jardach (metraż),
- Typ trawy na polu golfowym

OTA_GolfCourseSearchRS – wiadomość zawierająca informacje o polach golfowych spełniających dane kryteria (otrzymane w wiadomości OTA_GolfCourseSearchRQ). Jeżeli wartość atrybutu „Required" danego kryterium, ma wartość „Yes" wtedy w odpowiedzi dostaniemy tylko te pola golfowe, które spełniają dane kryterium. W przeciwnym przypadku w odpowiedzi mogą znaleźć się pola golfowe, które danego kryterium nie spełniają. Odpowiedzi są uporządkowane, według ilości spełnianych kryteriów.

W odpowiedzi, dla każdego pola golfowego zawarte są takie informacje jak:

- Id pola golfowego,
- Adres pola golfowego,
- Numer telefonu,
- Opis pola (lista kryteriów, wraz z wartościami).

Przykład (pobrany z OTA_MessageUserGuide2006V1.0):

Gracz w golfa chce uzyskać informacje o polach golfowych spełniających wybrane kryteria, czyli o polach, których nachylenie terenu (slope) jest mniejsze (Operation=LessThan) niż 110 stopni. Gracz wolałby również żeby pole było zaprojektowane przez Roberta Jonesa, ale nie jest to warunek konieczny.

Treść wiadomości OTA_GolfCourseSearchRQ:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseSearchRQ xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseSearchRQ.xsd"
EchoToken="54321" TimeStamp="2003-11-12T10:30:00" Target="Production" Version="1.001"
SequenceNmbr="2432" PrimaryLangID="en" ID="FL4902" DetailResponse="true">
<Criteria>
        <Criterion Name="Architect" Value="Robert Jones" Required="false"/>
        <Criterion Name="Slope" Value="110" Required="true" Operation="LessThan"/>
</Criteria>
</OTA_GolfCourseSearchRQ>
```

W odpowiedzi gracz dostał informacje o dwóch polach golfowych. Ponieważ kryterium „architekt" nie było kryterium wymaganym, to w odpowiedzi znajduje się też pole golfowe zaprojektowane przez innego architekta.

Kolejność pól golfowych w wiadomości OTA_GolfCourseSearchRS nie jest przypadkowa. Odpowiedzi są posortowane według ilości spełnionych, niewymaganych kryteriów. W naszym przykładzie takim kryterium był „architekt”, więc pierwsze w odpowiedzi jest pole golfowe o ID=”FL1234”, którego architektem jest Robert Jones (kryterium spełnione).

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseSearchRS xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseSearchRS.xsd"
EchoToken="54321" TimeStamp="2003-11-12T10:30:15" Target="Production" Version="1.002"
SequenceNmbr="2433" PrimaryLangID="en">
<Success/>
<GolfCourses>
<GolfCourse ID="FL1234" Name="Sea Grass Golf Resort">
        <Address>
                <CityName>Jupiter</CityName>
                <PostalCode>21921</PostalCode>
                <County>Palm Beach</County>
                <StateProv StateCode="FL"/>
                <CountryName Code="US"/>
        </Address>
        <Phone AreaCityCode="444" PhoneNumber="423-8954"/>
        <Traits>
                <Trait Name="Architect" Value="Robert Jones"/>
                <Trait Name="Singles Confirmed" Value="Yes"/>
                <Trait Name="ADA Challenged" Value="Wheelchair"/>
                <Trait Name="Slope" Value="110"/>
                <Trait Name="Metal Spikes" Value="No"/>
                <Trait Name="Caddies Available" Value="No"/>
                <Trait Name="Yardage" Value="6345"/>
                <Trait Name="Personal Carts Permitted" Value="No"/>
                <Trait Name="Fivesome" Value="No"/>
                <Trait Name="Grass Type" Value="Bermuda"/>
        </Traits>
</GolfCourse>
<GolfCourse ID="FL4321" Name="Beach Side Golf Resort">
        <Address>
                <CityName>Palm Beach Gardens</CityName>
                <PostalCode>21932</PostalCode>
                <County>Palm Beach</County>
                <StateProv StateCode="FL"/>
                <CountryName Code="US"/>
        </Address>
        <Phone AreaCityCode="444" PhoneNumber="423-2876"/>
        <Traits>
                <Trait Name="Architect" Value="Jack Nicklaus"/>
                <Trait Name="Singles Confirmed" Value="Yes"/>
                <Trait Name="ADA Challenged" Value="Wheelchair"/>
                <Trait Name="Slope" Value="112"/>
                <Trait Name="Metal Spikes" Value="Yes"/>
                <Trait Name="Caddies Available" Value="Yes"/>
                <Trait Name="Yardage" Value="7102"/>
                <Trait Name="Fivesome" Value="Yes"/>
                <Trait Name="Grass Type" Value="Rye"/>
        </Traits>
</GolfCourse>
</GolfCourses>
</OTA_GolfCourseSearchRS>
```

OTA_GolfCourseAvailRQ/RS

OTA_GolfCourseAvailRQ – pytanie o dostępność pola golfowego. Wszystkie elementy i atrybuty wiadomości są opcjonalne.

Informacje, jakie mogą być zawarte w OTA_GolfCourseAvailRQ:

- Data gry,
- Godzina rozpoczęcia gry (możliwość określenia, w jakich godzinach gra się rozpocznie np. pomiędzy 13.00 a 14.30)
- Identyfikator pola golfowego,
- Ilość dołków do gry,
- Maksymalna możliwa cena dla jednej osoby,
- Ilość graczy.

OTA_GolfCourseAvailRS – Zwraca zbiór wiadomości o dostępnych polach golfowych.

Informacje zawarte w OTA_GolfCourseAvailRS:

- Identyfikator pola golfowego,
- Data,
- Czas, w jakim gra się rozpocznie (np. pomiędzy 13.00 a 14.00)
- Ilość graczy,
- Ilość dołków,
- Maksymalna możliwa cena dla jednej osoby,
- Lista opłat: opłata za grę (green fee), opłata za wózek golfowy (cart fee) (każda informacja o opłacie składa się z nazwy opłaty np. „cartfee”, informacji czy w danej opłacie jest zawarty podatek, wysokości opłaty).

Przykład (pobrany z OTA_MessageUserGuide2006V1.0):

Użytkownik i troje jego przyjaciół chciałoby zagrać w golfa 31 października, rozpoczęcie gry pomiędzy 1.00 a 2.30. Są zainteresowani grą na polu z identyfikatorem FL1234. Maksymalna cena, jaką mogą zapłacić za 18 dołków, to 80.00$ na osobę.

Treść wiadomości z pytaniem o dostępności i kosztach wynajęcia pola golfowego OTA_GolfCourseAvailRQ:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseAvailRQ xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseAvailRQ.xsd"
EchoToken="12345" TimeStamp="1003-05-31T13:20:00-05:00" Target="Production" Version="1.001"
SequenceNmbr="123456">
<GolfCourseTeeTimes CourseID="FL1234">
        <GolfCourseTeeTime Start="2003-10-31T13:00:00" End="2003-10-31T14:30:00"
        NumberOfGolfers="4" NumberOfHoles="18" NumberOfTimes="1" MaxPrice="80.00" CurrencyCode="USD">
        </GolfCourseTeeTime>
</GolfCourseTeeTimes>
</OTA_GolfCourseAvailRQ>
```

Użytkownik otrzymał odpowiedź, że istnieje możliwość gry 31 października na polu golfowym o id FL1234. Opłata za grę wynosi 70.00$ na osobę, a opłata za wynajęcie wózka golfowego 11.00$.

Treść wiadomości OTA_GolfCourseAvailRS:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseAvailRS xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseAvailRS.xsd"
EchoToken="12345" TimeStamp="2003-05-31T13:20:00-05:00" Target="Production" Version="1.001"
SequenceNmbr="123456">
<Success/>
<UniqueID Type="4" ID="FL1234">
</UniqueID>
<GolfCourseTeeTimes>
        <GolfCourseTeeTime Start="2003-10-31T13:00:00" End="2003-10-31T14:30:00"
        NumberOfGolfers="4" NumberOfHoles="18" NumberOfTimes="1" MaxPrice="80.00" CurrencyCode="USD">
                <Fees>
                        <Fee TaxInclusive="true" Amount="70.00" CurrencyCode="USD">
                                <Description Name="GreensFee">
                                <Text Formatted="false" Language="EN"/>
                                </Description>
                        </Fee>
                        <Fee TaxInclusive="true" Amount="11.00" CurrencyCode="USD">
                                <Description Name="CartFee">
                                <Text Formatted="false" Language="EN"/>
                                </Description>
                        </Fee>
                </Fees>
</GolfCourseTeeTime>
</GolfCourseTeeTimes>
</OTA_GolfCourseAvailRS>
```

OTA_GolfCourseResRQ/RS

OTA_GolfCourseResRQ – wiadomość zawierająca prośbę o rezerwację.

Informacje zawarte w OTA_GolfCourseRQ:

- Informacje o użytkowniku (graczu, osobie dokonującej rezerwacji): imię i nazwisko, adres, data urodzenia, numer telefonu.
- Informacje formie płatności, np. o karcie kredytowej gracza,
- Data gry,
- Ilość graczy,
- Ilość zarezerwowanych pojazdów,
- Informacje o opłatach.

OTA_GolfCourseResRS – wiadomość z informacjami o rezerwacji i o sposobie (możliwości) rezygnacji z rezerwacji, a także czas, w jakim można to zrobić. W wiadomości użytkownik dostaje także informację o id rezerwacji.

Informacje o rezerwacji:

- Id rezerwacji.
- Informacje o użytkowniku (graczu, osobie dokonującej rezerwacji): imię i nazwisko, adres, data urodzenia, numer telefonu.
- Informacje o karcie kredytowej gracza,
- Data gry,
- Ilość graczy,
- Ilość zarezerwowanych pojazdów,
- Informacje o opłatach,
- Informacje o sposobie rezygnacji z rezerwacji (np. data, do której można dokonać rezygnacji i opłata będąca karą za rezygnację.

Przykład (pobrany z OTA_MessageUserGuide2006V1.0):

Użytkownik Boby Jones i jego przyjaciel, chcą zagrać w golfa 10 czerwca o godzinie 11.00 i zarezerwować jeden wózek golfowy. Użytkownik Bobby Jones wysyła prośbę o rezerwację pola golfowego o id FL1234. W wiadomości zawarte są dane użytkownika i informacje o jego karcie kredytowej.

Treść wiadomości OTA_GolfCourseResRQ:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseResRQ xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseResRQ.xsd"
EchoToken="12345" TimeStamp="2003-05-30T13:20:00Z" Target="Production" Version="1.002"
SequenceNmbr="123456" ID="FL1234" Notification="true">
<GolfCoursesRes>
<GolfCourseRes Status="New" StatusMessage="New Reservation Request"
ResponderResConfID="11223344" RequestorResID="22113344">
<Rounds>
        <Round RoundID="23456" PlayDateTime="2003-06-10T11:00:00" NumberOfGolfers="2"
        StartingTee="1" NumberOfCarts="1" Amount="140.00" CurrencyCode="USD">
        <Charges>
                <Charge TaxInclusive="true" Code="7" Amount="70.00" CurrencyCode="USD">
                <Description Name="Standard Rate - Tourist">
                <Text Formatted="false" Language="EN"/>
                </Description>
                </Charge>
        </Charges>
<Golfers>
<Golfer Gender="Male" BirthDate="1923-06-03">
        <Memberships>
                <Membership ProgramID="NGCOA" MembershipID="4123546"/>
        </Memberships>
        <PaymentForm>
                <PaymentCard CardType="1" CardCode="AX"
                CardNumber="3145234356876223" SeriesCode="1122334" EffectiveDate="0800"
                ExpireDate="0804">
                <CardHolderName>Mr. Bobby Jones</CardHolderName>
                <CardIssuerName BankID="1122334455"/>
                </PaymentCard>
        </PaymentForm>
        <PersonName>
                <NamePrefix>Mr.</NamePrefix>
                <GivenName>Bobby</GivenName>
                <Surname>Jones</Surname>
        </PersonName>
        <Address Type="1">
                <StreetNmbr>123 Augusta Lane</StreetNmbr>
                <CityName>Atlanta</CityName>
                <PostalCode>23456</PostalCode>
                <StateProv StateCode="GA">Georgia</StateProv>
                <CountryName Code="US">United States</CountryName>
        </Address>
        <Telephone PhoneTechType="1" CountryAccessCode="1" AreaCityCode="703"
        PhoneNumber="444-5555"/>
</Golfer>
</Golfers>
<RateQualifiers>
        <RateQualifier RateQualifier=" NGCOA">
        </RateQualifier>
</RateQualifiers>
</Round>
</Rounds>
<ResID Type="14" ID="123">
</ResID>
</GolfCourseRes>
</GolfCoursesRes>
</OTA_GolfCourseResRQ>
```

W odpowiedzi, użytkownik dostaje potwierdzenie rezerwacji, numer id rezerwacji oraz informacje o sposobie (możliwości) rezygnacji z rezerwacji.

Treść wiadomości OTA_GolfCourseResRS:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseResRS xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseResRS.xsd"
EchoToken="12345" TimeStamp="2003-05-30T13:20:00Z" Target="Production" Version="1.002"
SequenceNmbr="123456" ID="FL1234">
<Success/>
<GolfCoursesRes>
<GolfCourseRes Status="New" StatusMessage="New Reservation Request"
RequestorResID="AC4132" ResponderResConfID="11223344">
        <Rounds>
                <Round RoundID="23456" PlayDateTime="2003-06-10T11:06:00" NumberOfGolfers="2"
                StartingTee="1" NumberOfCarts="1" Amount="140.00" CurrencyCode="USD">
                <Charges>
                        <Charge TaxInclusive="true" Code="7" Amount="70.00" CurrencyCode="USD">
                        <Description Name="Standard Rate - Tourist">
                        <Text Formatted="false" Language="EN"/>
                        </Description>
                        </Charge>
                </Charges>
        <Cancel CancelByDate="2003-06-10T11:06:00" Amount="55.00" CurrencyCode="USD"/>
        <Golfers>
                <Golfer Gender="Male" BirthDate="1923-06-03">
                <Memberships>
                        <Membership ProgramID="NGCOA" MembershipID="4123546"/>
                </Memberships>
                <PaymentForm>
                        <PaymentCard CardType="1" CardCode="AX"
                        CardNumber="3145234356876223" SeriesCode="1122334" EffectiveDate="0800"
                        ExpireDate="0804">
                        <CardHolderName>Mr. Bobby Jones</CardHolderName>
                        <CardIssuerName BankID="1122334455"/>
                        </PaymentCard>
                </PaymentForm>
                <PersonName>
                        <NamePrefix>Mr.</NamePrefix>
                        <GivenName>Bobby</GivenName>
                        <Surname>Jones</Surname>
                </PersonName>
                <Address Type="1">
                        <StreetNmbr>123 Augusta Lane</StreetNmbr>
                        <CityName>Atlanta</CityName>
                        <PostalCode>23456</PostalCode>
                        <StateProv StateCode="GA">Georgia</StateProv>
                        <CountryName Code="US">United States</CountryName>
                </Address>
                <Telephone PhoneTechType="1" CountryAccessCode="1" AreaCityCode="703"
                PhoneNumber="444-5555"/>
        </Golfer>
        </Golfers>
<RateQualifiers>
        <RateQualifier RateQualifier=" NGCOA">
        </RateQualifier>
</RateQualifiers>
</Round>
</Rounds>
<ResID Type="14" ID="123">
</ResID>
</GolfCourseRes>
</GolfCoursesRes>
</OTA_GolfCourseResRS>
```

6. Opis projektu.

6.1 Założenia projektowe:

1. Informacje zawarte w systemie będą opisane za pomocą ontologii. Centralną częścią projektu jest repozytorium, w którym przechowywane są (przy pomocy technologii Jena), ontologicznie uporządkowane dane opisane w języku znacznikowym RDF.

2. W systemie będą przechowywane tylko „statyczne" dane. Czyli informacje o nazwie, adresie, kryteriach opisujących pole golfowe. Natomiast nie będzie w nim informacji o dostępności danego pola golfowego. Aby uzyskać takie informacje, system będzie musiał zapytać za pomocą wiadomości OTA ([21]), źródło zewnętrzne.

3. System został zaprojektowany w sposób umożliwiający jego późniejszą integrację z powstającym systemem wspomagania podróży TSS *(Travel Support System*) ([2])

4. Użytkownikiem projektu może być dowolna aplikacja lub inny agent programowy posługujący się wiadomościami w standardzie OTA.

5. System będzie posiadał także możliwość pobrania i aktualizacji danych znajdujących się w bazie (poprzez wczytanie danych o polach golfowych z wiadomości otrzymanych od innych systemów).

6.2 Przypadki użycia systemu:

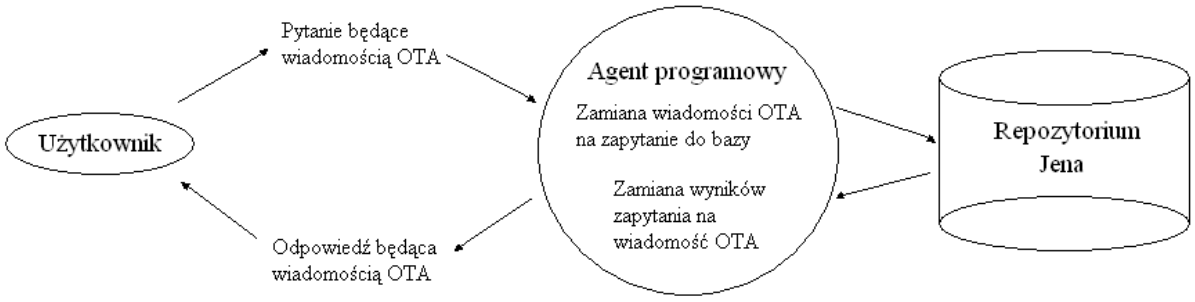
Scenariusz 1:

Zgodnie z pierwszym założeniem projektowym, system powinien dostarczać wiadomości dla jak największej liczby użytkowników. Czyli użytkownik, posługując się wiadomościami OTA, będzie mógł zapytać o informacje o polach golfowych w repozytorium.

Użytkownik wysyła do systemu pytanie, będące wiadomością XML, o pole golfowe. Następnie agenci programowi, na podstawie zadanego pytania, generują zapytanie do

repozytorium. Po otrzymaniu wyniku zapytania, zamienia otrzymane dane w wiadomość OTA i wysyła ją do użytkownika.

Schemat projektu:



Scenatiusz 2:

Zgodnie z założeniem nr 3, system musi zostać zaprojektowany w sposób umożliwiający jego późniejszą integrację z systemem TSS.

Celem systemu Wspomagania Podróży jest gromadzenie i dostarczania użytkownikowi spersonalizowanej informacji o hotelach, restauracjach, połączeniach komunikacyjnych. W przyszłości możliwe, że system będzie udzielał informacji także o polach golfowych. Aby agenci programowi systemu TSS mogli ocenić stopień ważności informacji dla danego użytkownika, informacja musi zostać opisana za pomocą ontologii ([2, 3, 4])

GolfCourse System musi zatem:

- na podstawie zbioru warunków (Map) opisujących dane do zapytania o dostępność pola golfowego, utworzyć zapytanie OTA (OTA_GolfCourseAvailRQ), zapytać źródło na zewnątrz, przetłumaczyć odpowiedź OTA (OTA_GolfCourseAvailRS) na zbiór ontologii i zwrócić go użytkownikowi (w przyszłości systemowi TSS),
- na podstawie zbioru warunków (Conditions) utworzyć zapytanie o pola golfowe, wykonać zapytanie i przekazać użytkownikowi wyniki zwrócone przez repozytorium (instancje ontologii GolfCourse).

Scenariusz 3:

W systemie będziemy przechowywać w bazie danych statyczne dane o polach golfowych, więc musi istnieć możliwość pobrania nowych danych i aktualizacja już przechowywanych. W tym celu zostały zaimplementowane dwa sposoby pobrania danych:

- Wczytanie danych z pliku RDF i zapisanie ich do bazy (podczas startu systemu, kiedy baza danych jest pusta)
- Wczytanie wiadomości OTA_GolfCourseSearchRS, przetłumaczenie jej na zbiór ontologii i zapisanie tego zbioru do bazy.

Pobranie danych umożliwia nie tylko wczytanie nowych, ale także aktualizację starych danych. Przed zapisem system sprawdza czy informacje o danym polu golfowym już istnieją w bazie, jeśli tak to dokonuje tylko ich aktualizacji, jeśli nie to dodaje nowe dane.

6.3. Projektowanie ontologii.

Ontologie, wykorzystywane w projekcie zostały zaprojektowane na podstawie odpowiednich, opisanych powyżej, wiadomości OTA. Analiza tych wiadomości pozwoliła określić, jakie informacje mają się znaleźć w projektowanych ontologiach. Celem pracy nie jest stworzenie ogólnej ontologii golfa, ale o ontologii, która powstanie na podstawie wiadomości OTA.

W następnym rozdziale zostaną wyodrębnione części wspólne pomiędzy ontologiami opisującymi pola golfowe, a ontologiami zawartymi w Systemie Wspomagania Podróży. A następnie zostaną przedstawione zaprojektowane ontologie.

6.3.1 Informacje zawarte w ontologiach.

Na podstawie wiadomości OTA, zostały określone informacje, jakie powinny zawierać ontologie opisujące pole golfowe, gracza i rezerwację pola golfowego [1].

Ontologia pola golfowego:

GolfCourse
Identyfikator pola golfowego. Użytkownik może uzyskać dany identyfikator odbierając wiadomość OTA_GolfCourseSearchRS. Użytkownik może posłużyć się danym id w celu uzyskania informacji o dostępności pola golfowego lub w celu dokonania jego rezerwacji.
Adres pola golfowego.
Traits - lista cech (kryteriów wyszukiwania) opisująca dane pole golfowe (np. rodzaj trawy)

Ontologia opisująca grę na polu golfowym:

GolfCourseTeeTime
Identyfikator pola golfowego. Użytkownik może uzyskać dany identyfikator odbierając wiadomość OTA_GolfCourseSearchRS. Użytkownik może posłużyć się danym id w celu uzyskania informacji o dostępności pola golfowego lub w celu dokonania jego rezerwacji.
Cena.
Czas i data rozpoczęcia gry.
Czas i data zakończenia gry.
Maksymalna cena za jedną osobę.
Ilość dółków na polu golfowym.
Ilość graczy.
Ilość rozegranych gier.
Opłaty za korzystanie z pola golfowego i z wypożyczonego sprzętu.

6.3.2 Części wspólne pomiędzy Systemem Wspomagania Podróży, a ontologiami opisującymi pole golfowe.

Ontologie, tworzone w ramach danego projektu, powinny zostać zaprojektowane w sposób umożliwiający integrację projektu z powstającym Systemem Wspomagania Podróży ([2, 5]) . W tym celu należy wyodrębnić części wspólne pomiędzy ontologiami opisującymi pola golfowe, a ontologiami zawartymi w tym systemie.

Celem Systemu Wspomagania Podróży jest gromadzenie i dostarczanie użytkownikowi danych o bazie hotelarskiej, bazie restauracji, połączeniach komunikacyjnych, geograficznym położeniu opisywanych obiektów.

Części wspólne pomiędzy ontologiami opisującymi pole golfowe, a Systemem Wspomagania Podróży ([5]) to:

- Opis lokalizacji (położenia geograficznego, adresu, charakterystyki okolicy) – miejsce zakwaterowania, położenie restauracji, lotniska i pola golfowego. W Systemie Wspomagania podróży istnieje klasa LokalizacjaZewnetrzna (org. *OutdoorLocation*) zawierająca atrybuty opisu lokalizacji. Klasy zakwaterowania, restauracji i lotniska zostały powiązane relacją *podklasy* (ang. sub-class) z klasą LokalizacjaZewnetrzna. Relacja *podklasy* w wykorzystywanym do budowy ontologii języku RDF(S) zapewnia dziedziczenie wszystkich atrybutów klasy nadrzędnej przez klasy potomne. Przeprowadzona w ten sposób generalizacja zapewnia ujednolicony sposób opisu dowolnego miejsca posiadającego adres i współrzędne geograficzne. Klasa GolfCourse, opisująca pole golfowe, również będzie podklasą klasy OutdoorLocation.

OutdoorLocation	
Atrybut	**Opis**
	Klasa główna reprezentująca koncept lokalizacji, budynek, plac, obiekt natury, itp.
adres	Dokładne informacje adresowe. Wartość: instancja *AddressRecord*.
atrakcje	Atrakcje znajdujące się w pobliżu lokacji. Wartość: instancja *AttractionCategory*.
odniesienie	Punkt odniesienia. Wartość: instancja *IndexPoint*
dystans	Odległość od punktu odniesienia. Wartość: naturalna.
kategoria	Kategoria lokacji, np. jezioro, teren miejski, plaża. Wartość: instancja *LocationCategory*.
okolica	Opis słowny okolicy danej lokacji. Wartość: ciąg znaków.
długość	Długość geograficzna. Wartość: rzeczywista.
szerokość	Szerokość geograficzna. Wartość: rzeczywista.

- Następną częścią wspólną ontologii w Systemie Wspomagania Podróży i ontologii opisujących pole golfowe jest koncept zniżki i taryfy.

Zniżki	
Atrybut	**Opis**
	Klasa główna reprezentująca koncept zniżki bądź taryfy.
Procent	Procent przysługującej ulgi. Najczęściej stosowany zamiennie z atrybutem *kwota.* Wartość: rzeczywista.
Kwota	Przysługująca ulga wyrażona konkretną kwotą. Najczęściej stosowany zamiennie z atrybutem *procent*. Wartość: rzeczywista.
Typ	Typ danej zniżki lub taryfy. Wartość: instancja *TypZniżki*
Nazwa	Opis słowny. Wartość: ciąg znaków.
Warunki	Słowny opis warunków uzyskania ulgi. Wartość: ciąg znaków.

W systemie, ontologia miejsca zakwaterowania powstała (podobnie jak ontologia pola golfowego) na podstawie specyfikacji OTA, a ontologia podróży lotniczej powstała na podstawie dokumentacji stworzonej przez IATA (Międzynarodowe Zrzeszenie Przewoźników Powietrznych ang. International Air Transport Association = IATA). Różnica pomiędzy zniżkami opisywanymi przez OTA, a opisywanymi przez IATA, polega na możliwych wartościach typu zniżki. Skutkiem tych różnic, w Systemie Wspomagania Podróży istnieją trzy klasy opisujące możliwe zniżki:

- TypyZnizekOTA (org. *OTADiscountTypes,)* Klasa nadrzędna wszystkich typów zniżek i taryf wyróżnionych w specyfikacji OTA, bez elementów wspólnych z IATA
- TypyZnizekIATA (org. *IATADiscountTypes)* – klasa nadrzędna wszystkich typów zniżek i taryf wyróżnionych w specyfikacji IATA, bez elementów wspólnych z OTA
- ZnizkiWspolneOtaIata (klasa podrzędna wobec TypyZnizekIATA i TypyZnizekOTA)
- klasa nadrzędna dla wszystkich typów zniżek.

- Do części wspólnych Systemu Wspomagania Podróży i ontologii opisujących pole golfowe należy także zaliczyć koncept form płatności. W systemie istnieje klasa MeanOfPayment opisująca możliwe formy płatności. Zostanie wykorzystana także w ontologii GolfCourseReservation i GolfCourse.

- AdressRecord – klasa Systemu Wspomagania Podróży opisująca adres. Zostanie użyta do opisu adresu gracza (ontologia Golfer)

AdressRecord	
fullAdress	Pełny adres.
street	Nazwa ulicy.
houseNumber	Numer domu
flatNumber	Numer mieszkania
region	Region
country	Kraj
city	Miasto

- Currency – ontologia opisująca walutę,
- FareTax – klasa Systemu Wspomagania Podróży opisująca informacje o podatkach.
- Contacts – klasa Systemu Wspomagania Podróży opisująca możliwe formy kontaktu.

6.4 Ontologie opisujące pole golfowe.

Po określeniu, jakie informacje zostaną zawarte w danych ontologiach i wyodrębnieniu części wspólnych pomiędzy projektowanymi ontologiami a ontologiami w Systemie Wspomagania Podróży, możemy zaprojektować ontologie opisujące pole golfowe.

Ontologie pomocnicze:

- Price – ontologia opisująca cenę (opłata, podatek, waluta),
- Fee – ontologia opisująca opłatę (np. za grę na polu golfowym, za korzystanie ze sprzętu golfowego).
- Tax – ontologia opisująca podatek,
- Description – ontologia do przedstawienia opisu,

Ontologia opisująca cenę – Price:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
# W ontologii Price, wykorzystane zostaną ontologie opisujące podatek
# (FareTax) i walutę (CurrencyOntology).

@prefix tax: <Payment/FareTax#>.
@prefix cur: <http://protege.stanford.edu/CurrencyOntology#>.

base:Price a rdfs:Class;
        rdfs:comment "Description of a price".

base:amount a rdf:Property;
                        rdfs:domain base:Price;
                        rdfs:range xsd:float.

base:taxInclusive a rdf:Property;
                  rdfs:domain base:Price;
                  rdfs:range xsd:boolean.

base:totalAmount a rdf:Property;
                     rdfs:domain base:Price;
                     rdfs:range xsd:double;
                  rdfs:comment "Total amount (taxes incl.).".

# Wartość netto do zapłaty (bez podatku).

base:fareAmount a rdf:Property;
                     rdfs:domain base:Price;
                     rdfs:range xsd:double;
                  rdfs:comment "Fare amount (taxes excluded).".

# Informacje o podatku - ontologia FareTax (będąca częścią Systemu
Wspomagania Podróży)

base:tax a  rdf:Property;
                  rdfs:domain base:Price;
                  rdfs:range tax:FareTax;
                  rdfs:comment "Information about taxes.".

# Informacja o walucie.

base:curr a       rdf:Property;
                        rdfs:domain base:Price;
                        rdfs:range cur:Currency;
                  rdfs:comment "The currency information.".
```

Ontologia opisująca opłaty na polu golfowym – Fee:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix desc: <file:///D:/ontologies/domain/Golf/Description#>.
@prefix price: <file:///D:/ontologies/domain/Golf/Price#>.
@prefix disc: <file:///D:/ontologies/domain/Payment/Discounts#>.
@prefix base: <file:///D:/ontologies/domain/Golf/Fee#>.

base:Fee a rdfs:Class;
        rdfs:comment "Description of a fee".

base:description a rdf:Property;
```

```
                        rdfs:domain base:Fee;
                        rdfs:range desc:Description.

base:price a rdf:Property;
                  rdfs:domain base:Fee;
                  rdfs:range price:Price.

base:effectiveDate a rdf:Property;
                  rdfs:domain base:Fee;
                  rdfs:range xsd:date.

base:expireDate a rdf:Property;
                  rdfs:domain base:Fee;
                  rdfs:range xsd:date.

base:discounts a rdf:Property;
                  rdfs:domain base:Fee;
                  rdfs:range disc:Discounts.
```

Ontologia “Description”:

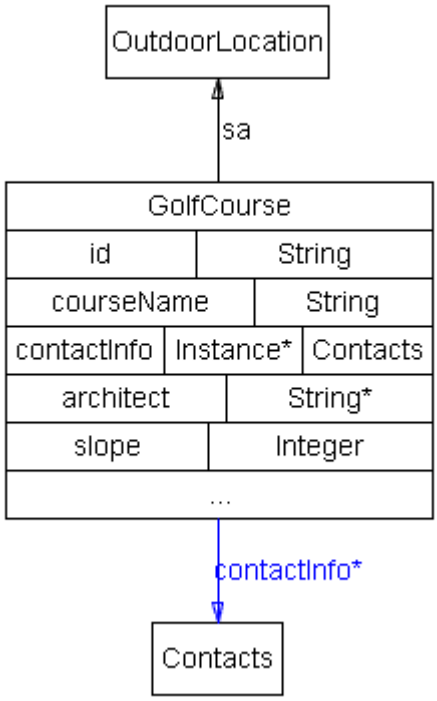
```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix base: <file:///D:/ontologies/domain/Golf/Description#>.

base:Description a rdfs:Class;
        rdfs:comment "Description".

base:name a rdf:Property;
                  rdfs:domain base:Description;
                rdfs:range xsd:string.

base:createDate a rdf:Property;
            rdfs:domain base:Description;
            rdfs:range xsd:date.
```

Ontologia pola golfowego (GolfCourse):

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix base: <Golf/GolfCourse#>.

# W ontologii pola golfowego będziemy korzystać z ontologii (będących
# częścią Systemu Wspomagania Podróży) opisujących lokalizację zewnętrzną
# (OutdoorLocation), sposób płatności (MeanOfPayment) i kontakty (Contacts)

@prefix loc: <Location/OutdoorLocation#>.
@prefix pmt: <Payment/MeanOfPayment#>.
@prefix phc: <Contacts/Contacts#>.
@prefix rsv: <HotelInfrastructureCodes/ReservationCodes#>.

# W ontologii pola golfowego będziemy korzystać z pomocniczej ontologii
# opisującej rodzaj trawy (GrassType)
```

```
@prefix grs: <Golf/GrassType#>.


# Zgodnie z założeniami klasa GolfCourse jest podklasą klasy
# Lokalizacja Zewnętrzna (OutdoorLocation)

base:GolfCourse   a rdfs:Class;
                  rdfs:subClassOf   loc:OutdoorLocation;
                  rdfs:comment "Used for golf course's city and
                  geographical location description".

# id pola golfowego – ciąg znaków jednoznacznie identyfikujący dane pole
# golfowe (każde pole ma swoje unikalne id)

base:id       a rdf:Property;
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:string.
# nazwa pola golfowego – ciąg znaków

base:name  a rdf:Property;
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:string.

# Informacje o sposobach kontaktu

base:contactInfo a rdf:Property;
                  rdfs:comment "Contact information.";
                  rdfs:domain base:GolfCourse;
                  rdfs:range phc:Contacts.


# Informacje o polu golfowym - lista cech pola golfowego, kryteria
# wyszukiwania pól golfowych (traits) – na podstawie OTA_Messages:

base:architect a  rdf:Property;
                  rdfs:comment "Information about architect, who design
                  this golf course";
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:string.

base:slope a      rdf:Property;
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:integer.

base:availCaddy a rdf:Property;
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:boolean.

base:permCart a rdf:Property;
                  rdfs:comment "Information if personal carts are
                  permitted"
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:boolean.

base:yardage a    rdf:Property;
                  rdfs:domain base:GolfCourse;
                  rdfs:range xsd:float.

base:singlesConfirmed a rdf:Property;
                  rdfs:domain base:GolfCourse;
```

```
                    rdfs:range xsd:boolean.

base:metalSpikes a      rdf:Property;
                    rdfs:domain base:GolfCourse;
                    rdfs:range xsd:boolean.

base:grass  a       rdf:Property;
                    rdfs:domain base:GolfCourse;
                    rdfs:range xsd:string;
```

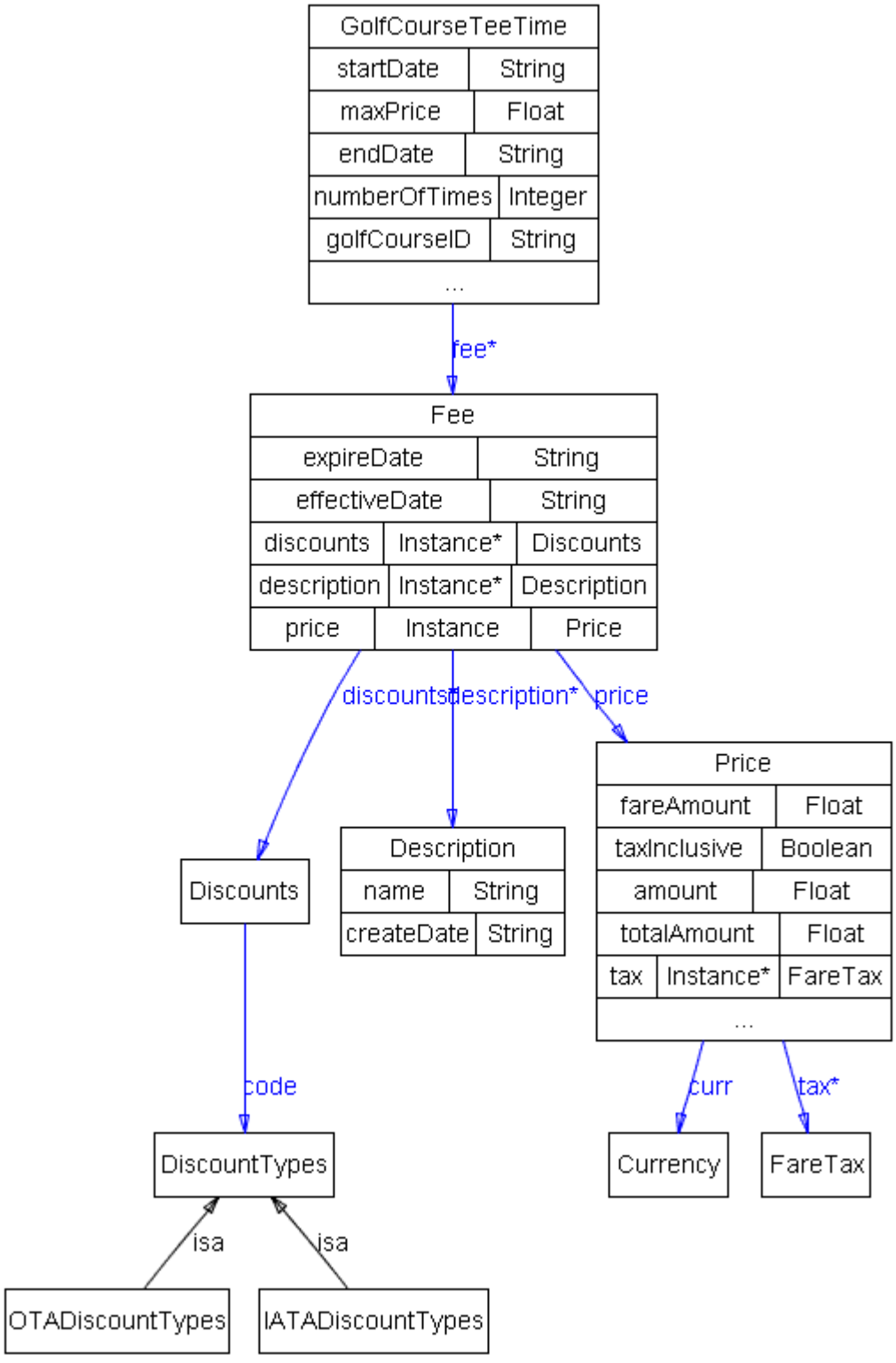


Ontologia pola golfowego GolfCourse.

Ontologia gry na polu golfowym – GolfCourseTeeTime

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix fee: <file:///D:/ontologies/domain/Golf/Fee#>.
@prefix base: <file:///D:/ontologies/domain/Golf/GolfCourseTeeTime#>.

base:GolfCourseTeeTime a rdfs:Class;
        rdfs:comment "Description of golf tee time”.

base:golfCourseID a rdf:Property;
                        rdfs:domain base:GolfCourseTeeTime;
                        rdfs:range xsd:string.

base:amount a rdf:Property;
                    rdfs:domain base:GolfCourseTeeTime;
                    rdfs:range xsd:float.
```

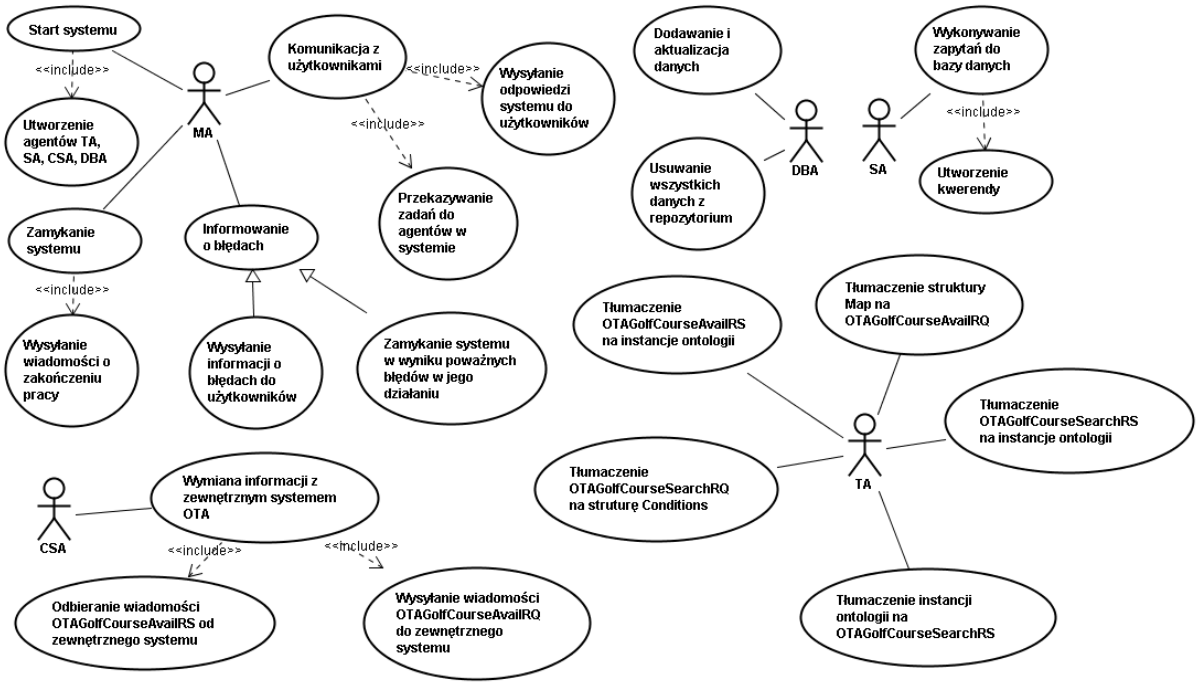
```
base:currencyCode a rdf:Property;
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:string.

base:startDate a rdf:Property;
                  rdfs:comment "Information about date and time in format
                  yyyy:MM:dd'T'HH:mm:ss";
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:string.

base:endDate a rdf:Property;
                  rdfs:comment "Information about date and time in format
                  yyyy:MM:dd'T'HH:mm:ss";
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:string.

base:maxPrice a rdf:Property;
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:float.

base:numberOfHoles a rdf:Property;
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:integer.

base:numberOfTimes a rdf:Property;
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range xsd:integer.

base:fee a rdf:Property;
                  rdfs:domain base:GolfCourseTeeTime;
                  rdfs:range fee:Fee.
```



Ontologia gry na polu golfowym „GolfCourseTeeTime”

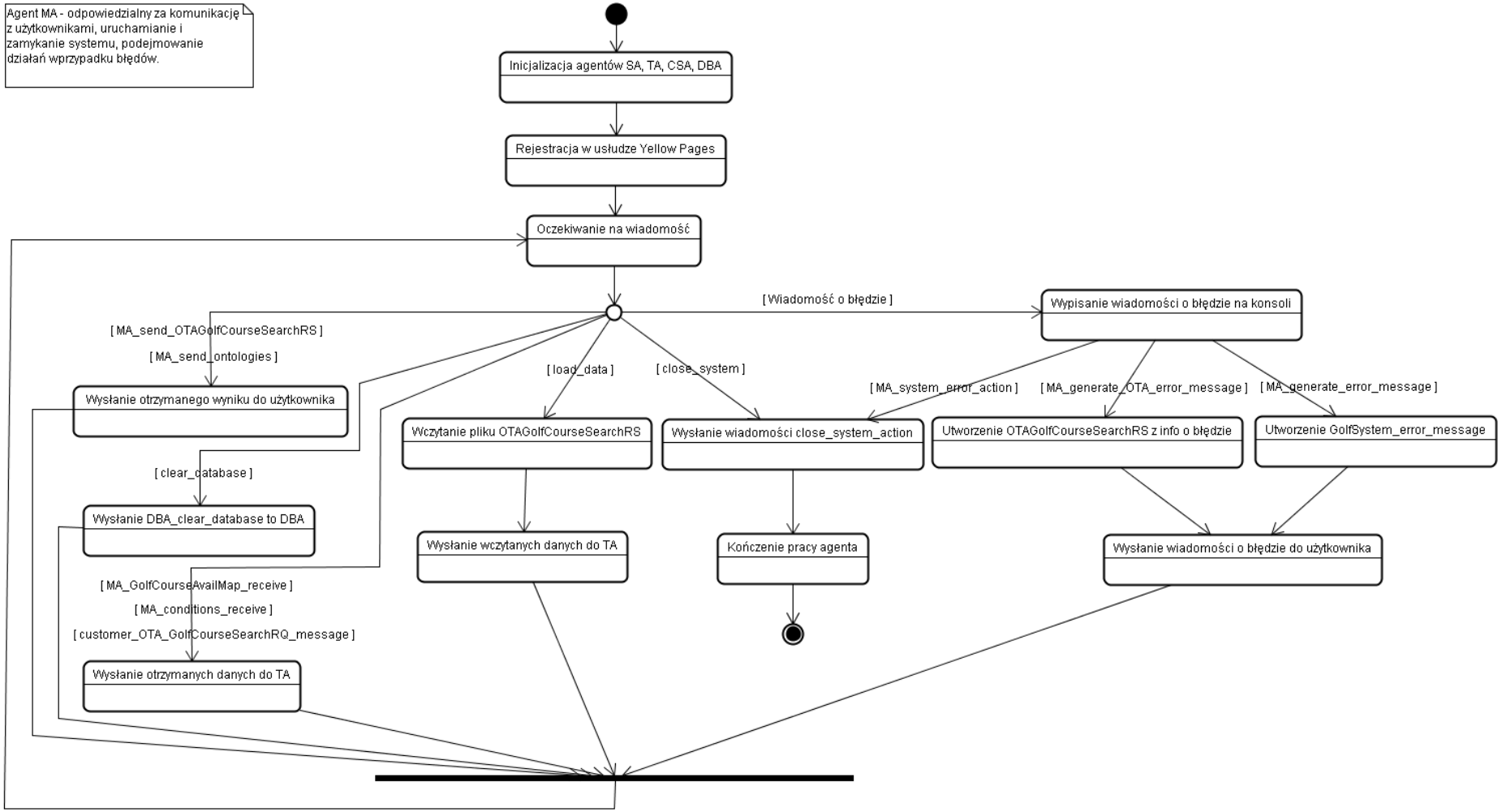
6.5. Agenci programowi w systemie GolfSystem:

Agenci będący częścią systemu:



Agenci pomocniczy:

Główny agent nadzorca (Manager Agent – MA) – jest agentem nadzorującym pracę całego systemu. To z nim łączą się klienci (MA odbiera wiadomości OTA). MA przy uruchamianiu systemu inicjuje pracę wszystkich pozostałych agentów, a także rozsyła żądania zakończenia pracy w momencie, gdy system jest zamykany. MA tworzy pozostałych agentów poza agentami CA (*Customer Agent*) i PA (*Personal Agent*), ponieważ oni nie są częścią systemu) i dodaje ich do kontenera JADE.


Agent MA - odpowiedzialny za komunikację z użytkownikami, uruchamianie i zamykanie systemu, podejmowanie działań wprzypadku błędów.
Inicjalizacja agentów SA, TA, CSA, DBA
Rejestracja w usłudze Yellow Pages
Oczekiwanie na wiadomość
[Wiadomość o błędzie]
Wypisanie wiadomości o błędzie na konsoli
[MA_send_OTAGolfCourseSearchRS]
[MA_send_ontologies]
Wysłanie otrzymanego wyniku do użytkownika
[load_data]
[close_system]
[MA_system_error_action]
[MA_generate_OTA_error_message]
[MA_generate_error_message]
Wczytanie pliku OTAGolfCourseSearchRS
Wysłanie wiadomości close_system_action
Utworzenie OTAGolfCourseSearchRS z info o błędzie
Utworzenie GolfSystem_error_message
[clear_database]
Wysłanie DBA_clear_database to DBA
Wysłanie wczytanych danych do TA
Kończenie pracy agenta
Wysłanie wiadomości o błędzie do użytkownika
[MA_GolfCourseAvailMap_receive]
[MA_conditions_receive]
[customer_OTA_GolfCourseSearchRQ_message]
Wysłanie otrzymanych danych do TA

Wiadomości odbierane przez MA:

Agent MA jako jedyny agent w systemie może otrzymywać informacje od użytkownika (innych agentów z poza systemu, użytkownika korzystającego z GUI oferowanego przez platformę jade). Dlatego wiadomości odbierane przez MA zostały podzielone na wiadomości od agentów będących częścią systemu i na wiadomości od pozostałych użytkowników).

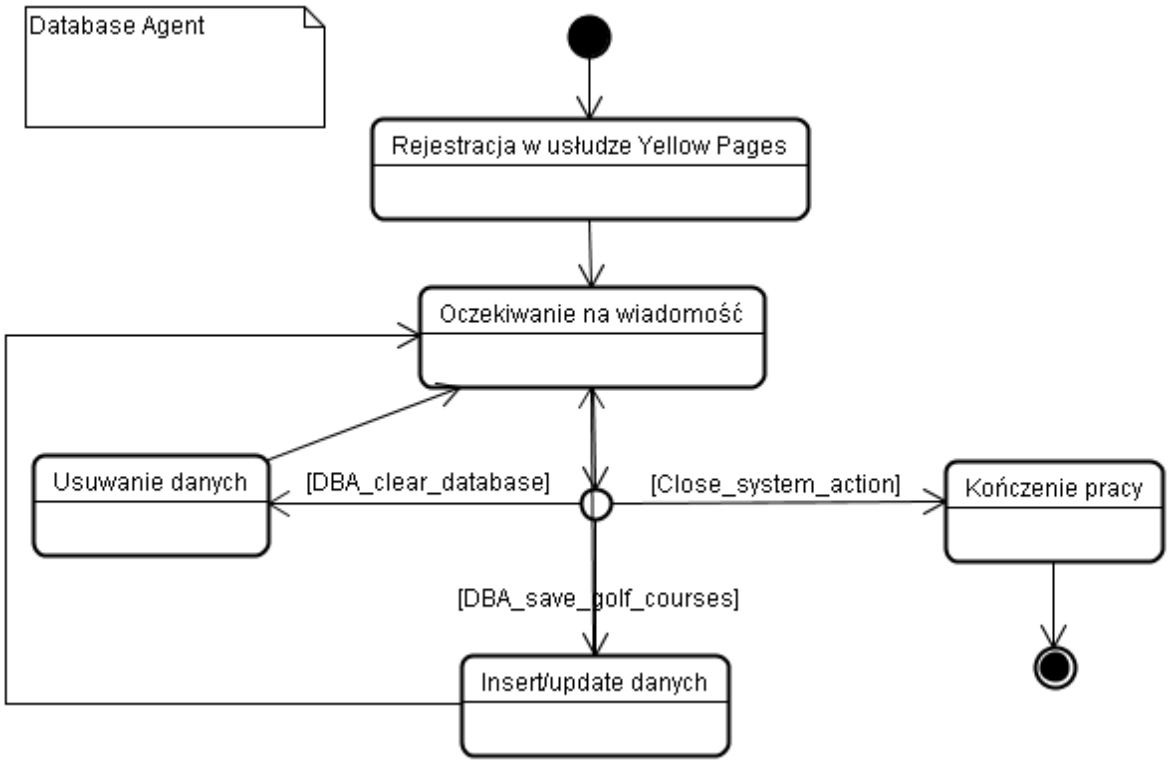
Wiadomości od systemu.	
action	Działanie agenta MA:
MA_send_OTAGolfCourseSearchRS	Agent MA wysyła otrzymaną wiadomość OTAGolfCourseSearchRS do użytkownika (np. agenta CA), który wysłał wcześniej do systemu pytanie o pole golfowe – OTAGolfCourseSearchRQ (część scenariusza 1)
MA_send_ontologies	Wiadomość ta zawiera instancje ontologii (OWLData). Agent MA może otrzymać taką wiadomość od agenta TA lub SA. Po otrzymaniu tego typu wiadomości agent MA wysyła odpowiedź z ontologiami do oczekującego jej użytkownika. (część scenariusza 2)
MA_generate_OTA_error_message	Agent MA może otrzymać taką wiadomość od dowolnego agenta w systemie w sytuacji gdy nastąpi wyjątek podczas generowania przez system odpowiedzi na wiadomość OTAGolfCourseSearchRQ i wyjątek ten nie będzie spowodowany błędem w pracy systemu a np niepoprawnymi danymi (*TranslateException*) W wiadomości tej zawarty jest opis błędu. Po otrzymaniu takiej wiadomości agent MA generuje odpowiedź OTAGolfCourseSearchRS z informacją o błędzie (bez informacji o polach golfowych) i wysyła ją do odpowiedniego użytkownika (np

	agenta CA). (wynik błędów podczas wykonywania akcji scenariusza 1).
MA_system_error_action	Agent MA może otrzymać taką wiadomość od dowolnego agenta w systemie w sytuacji gdy nastąpi błąd działania systemu (np w systemie nie będzie żadnego agenta TA). Po otrzymaniu takiej wiadomości agent MA wyświetla informację o błędzie na konsoli a następnie zamyka system.
MA_generate_error_message	Agent MA może otrzymać taką wiadomość od dowolnego agenta w systemie w sytuacji gdy nastąpi wyjątek podczas generowania przez system odpowiedzi na pytanie użytkownika i oczekiwaną odpowiedzią nie będzie wiadomość OTA a np zbiór instancji ontologii. Wyjątek ten nie będzie spowodowany błędem w pracy systemu a np niepoprawnymi danymi (*TranslateException*). Po otrzymaniu takiej wiadomości agent MA wypisuje informacje o błędzie na konsoli, generuje i wysyła do odpowiedniego użytkownika wiadomość o błędzie. (Wynik błędów podczas wykonywania akcji scenariusza 2).
MA_show_error_message_action	Agent MA może otrzymać taką wiadomość od dowolnego agenta w systemie w sytuacji gdy nastąpi wyjątek podczas działania systemu, wyjątek ten nie będzie spowodowany błędem w systemie, a akcja podczas której on nastąpi nie będzie miała na celu udzielenia użytkownikowi odpowiedzi (np akcja wczytywania danych albo usuwania danych z repozytorium). Po otrzymaniu takiej wiadomości agent MA wypisuje na konsoli infomację o błędzie. (Wynik błędów podczas wykonywania akcji scenariusza 3).

Wiadomości od użytkownika	
action	Działanie agenta MA:
MA_GolfCourseAvailMap_receive	Agent MA w wiadomości otrzymuje obiekt Map – pytanie o dostępność pola golfowego. Agent MA wysyła wiadomość do jednego z agentów TA (*translation agent*). Wiadomość zawiera akcję TA_translate_to_OTAGolfCourseAvailRQ (część scenariusza 2)
MA_conditions_receive	Wiadomość ta zawiera Conditions – czyli zbiór warunków jakie powinno spełniać dane pole golfowe. Po otrzymaniu takiej wiadomości agent MA wysyła wiadomość (z Conditions) do agenta SA z akcją SA_search_golf_courses. (część scenariusza 2)
„close system”	Agent MA może otrzymać taką wiadomość od dowolnego użytkownika systemu (np osoby korzystającej z GUI oferowanego przez platformę jade). Po otrzymaniu takiej wiadomości agent MA wysyła wiadomość o zamknięciu systemu do wszystkich pozostałych agentów w systemie a następnie sam kończy swoją działalność.
„load data”	Agent MA może otrzymać taką wiadomość od dowolnego użytkownika systemu (np osoby korzystającej z GUI oferowanego przez platformę jade). Po takiej wiadomości system nie generuje odpowiedzi. Agent MA wczytuje podany plik OTAGolfCourseSearchRS.xml z informacjami o polach tekstowych a następnie wysyła wczytane informacje do TA (*translation agent*). (część scenariusza 3)
„clear database”	Agent MA może otrzymać taką wiadomość od

	dowolnego użytkownika systemu (np osoby korzystającej z GUI oferowanego przez platformę jade). Po otrzymaniu takiej wiadomości agent MA wysyła wiadomość do agenta DBA (*database agent*) z akcją DBA_clear_database. (część scenariusza 3)
customer_message – wiadomość OTA	Agent MA może otrzymać taką wiadomość od dowolnego użytkownika systemu, np osoby korzystającej z GUI oferowanego przez platformę jade lub agenta CA (*customer agent*). Wiadomość ta zawiera (w postaci stringa) OTAGolfCourseSearchRQ. Po otrzymaniu takiej wiadomości agent MA wysyła wiadomość do agenta TA z akcją: TA_translate_from_OTAGolfCourseSearchRQ (część scenariusza 1)

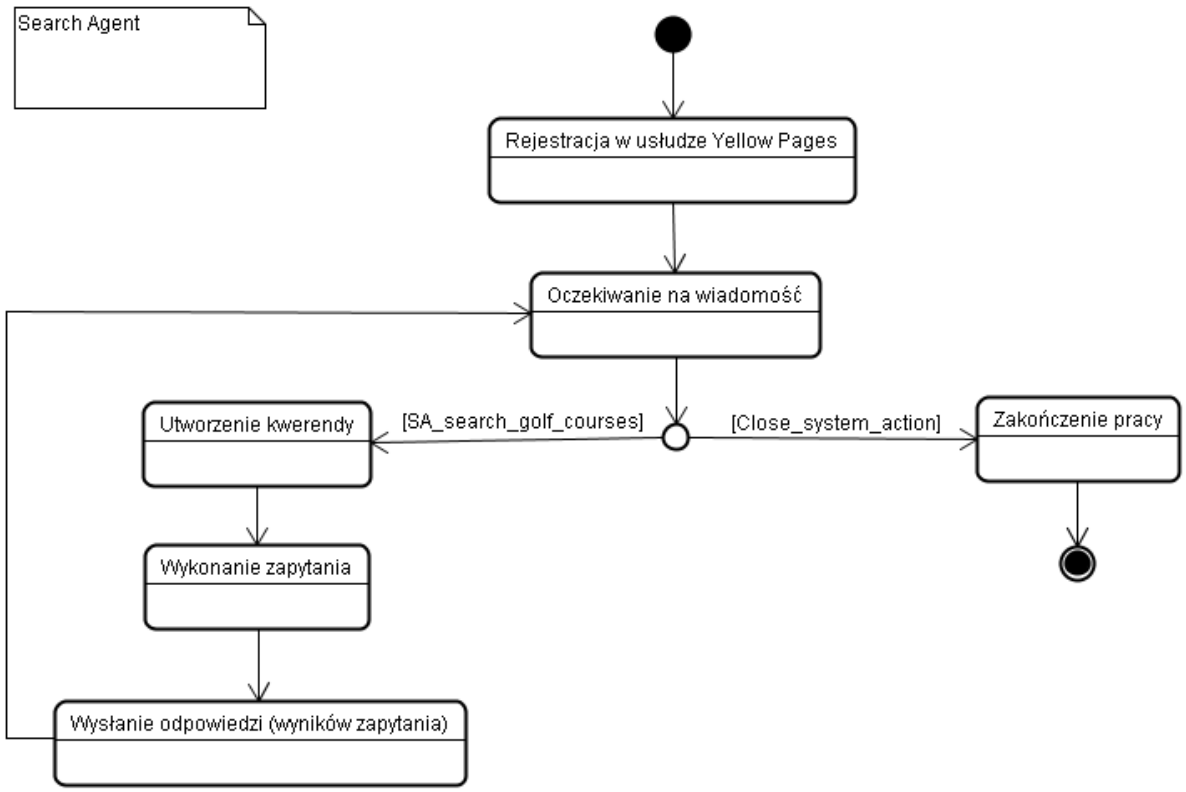
Agent bazodanowy (DataBase Agents – DBA) - wykonuje operacje zapisu na modelu Jena. W systemie jest tylko jeden agent tego typu i tylko on ma możliwość zapisywania do bazy danych.



Wiadomości odbierane przez DBA:

message	Działanie agenta DBA.
DBA_save_golf_courses	Wiadomość ta zawiera zbiór instancji ontologii (OWLData). Agent DBA zapisuje otrzymane instancje do repozytorium. Jeśli dane pole istniało już w bazie – agent DBA uaktualnie dane o nim. Jeśli dane pole nie istniało w bazie zostaje dodane. (Część scenariusza 3)
DBA_clear_database	Agent DBA usuwa wszystkie dane z repozytorium. (Część scenariusza 3)
Close_system_action	Agent DBA kończy swoją działalność.

Agent wykonujący zapytania RDF (Search Agent - SA) – agent tworzy, na podstawie otrzymanych danych do repozytorium z ontologiami i wykonuje je. Następnie przesyła wynik do agenta TA (Translation Agent). Agenci QA i agent DBA są jedynymi agentami mogącymi pracować z modelem z danymi. Reszta agentów jest odizolowana od fizycznej formy zapisu RDF.



Wiadomości odbierane przez SA:

action	Działanie agenta SA.
SA_search_golf_courses	Wiadomość ta zawiera Conditions – listę warunków jakie powinny spełniać szukane pola golfowe. Na podstawie tych warunków agent SA tworzy kwerendę SPARQL i ją wykonuje. Po uzyskaniu wyników, agent SA sortuje pola golfowe według ilości spełnianych, niewymaganych kryteriów. Następnie agent SA wysyła otrzymane wyniki do agenta MA (scenariusz 2) lub do agenta TA (scenariusz 1).

SA_return_golf_courses	
Close_system_action	Agent SA kończy swoją działalność.

Agenci tłumaczący wiadomości OTA (Translation Agents – TA) –

Translation Agents korzystają z dwóch struktur pomocniczych przy tłumaczeniu wiadomości:

- Conditions – zawiera listę obiektów klasy Condition.

```
class Condition implements Concept{
   String name_;     //nazwa kryterium (np. „Architect”)
   boolean required_; //informacja, czy dane kryterium jest wymagane
   String valueString;     // wartość kryterium (np. „Jan Kowalski”)
   String operation_;      //rodzaj operacji (np. EQUAL lub LESS_THEN)
}
```

Klasa Condition służy do opisu warunku jaki powinno spełniać pole golfowe (warunki z wiadomości OTA_GolfCourseSearchRQ). Agent SA, na podstawie listy obiektów klasy Condition, generuje zapytanie do repozytorium Jena. Powodem dla którego używamy struktury pomocniczej Condition, a nie tłumaczymy wiadomość OTA_GolfCourseSearchRQ od razu na zapytanie SPARQL, jest założenie że w przyszłości system zostanie zintegrowany z systemem TSS. Wtedy TSS nie będzie generowało wiadomości OTA_GolfCourseSearchRQ żeby wyszukać interesujące użytkownika pole golfowe, a jedynie wyśle do systemu listę obiektów klasy Condition. Zatem możliwe są dwie sytuacje. W pierwszej użytkownik wysyła do systemu wiadomość OTA_GolfCourseSearchRQ i agent TA tłumaczy ją na zbiór Condition, w drugiej system od razu dostaje listę Conditions.

- Map – struktura z systemu TSS, do przekazywania parametrów. Zostanie wykorzystana do przekazywania parametrów zapytania o dostępność pola golfowego. Na podstawie tej struktury generowana jest wiadomość OTA_GolfCourseAvailRQ w celu uzyskania informacji o dostępności pola golfowego od zewnętrznego systemu.

Struktura Map składa się z listy obiektów klasy MapEntry:

```
public class MapEntry implements Concept {
   private String key; //nazwa parametru np. golfCourseId lub numberOfHoles
   private String value; // wartość parametru np. „1232” lub „9”
}
```

Klasy Conditions, Condition, Map i MapEntry dziedziczą z klasy Concept i są częścią ontologii GolfCourse służącej do wymiany informacji pomiędzy agentami.

Zadaniem agentów TA jest tłumaczenie:

- wiadomości OTAGolfCourseSearchRQ na strukturę pomocniczą Conditions,
- zbioru instancji ontologii GolfCourse na wiadomość OTAGolfCourseSearchRS,
- wiadomości OTAGolfCourseSearchRS na zbiór instancji ontologii GolfCourse,
- struktury Map na wiadomość OTAGolfCourseAvailRQ,
- wiadomości OTAGolfCourseAvailRS na zbiór instancji ontologii GolfCourseTeeTime

Agent TA podczas tłumaczenia wiadomości korzysta z klas wygenerowanych przez generatory Jastor ([23]) i Castor ([24]).

Praca z generatorem Castor:

Castor jest produktem Open Source firmy Exolab Group. Generuje klasy Javy na podstawie plików xsd do odczytywania i zapisywania plików xml.

W projekcie zostały wygenerowane klasy dla każdej z wiadomości OTA, czyli OTA_GolfCourseSearchRQ, OTA_GolfCourseSearchRS, OTA_GolfCourseAvailRQ, OTA_GolfCourseAvailRS, OTA_GolfCourseResRQ, OTA_GolfCourseResRS. Castor generuje wiele klas, nie tylko dla wiadomości ale także dla każdego typu ich atrybutów.

Przykład generowania klas na podstawie XMLSchema:

XMLSchema dla wiadomości OTA_GolfCourseSearchRQ:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.opentravel.org/OTA/2003/05"
targetNamespace="http://www.opentravel.org/OTA/2003/05"
elementFormDefault="qualified" version="1.005" id="OTA2006A">
<xs:include schemaLocation="OTA_GolfCourseTypes.xsd"/>
<xs:element name="OTA_GolfCourseSearchRQ">
      <xs:complexType>
            <xs:sequence>
            <xs:element name="Criteria">
                  <xs:complexType>
                  <xs:sequence>
                        <xs:element name="Criterion" maxOccurs="99">
                        <xs:complexType>
                              <xs:attributeGroup ref="CriteriaGroup"/>
                        </xs:complexType>
                        </xs:element>
```

```
                </xs:sequence>
                </xs:complexType>
            </xs:element>
            </xs:sequence>
      <…>
      </xs:complexType>
      </xs:element>
</xs:schema>
```

Z pliku XMLSchema wynika, że każda wiadomość OTA_GolfCourseSearchRQ ma listę elementów *Criterion. Criterion* jest atrybutem posiadającym referencje do grupy atrybutów *CriteriaGroup.*

Część pliku XMLSchema dla *CriteriaGroup:*

```
 <xs:attributeGroup name="CriteriaGroup">
<…>
<xs:attribute name="Name" type="StringLength1to32" use="required">
</xs:attribute>

<xs:attribute name="Value" type="StringLength1to16" use="required">
</xs:attribute>

<xs:attribute name="Required" type="xs:boolean" use="required">
</xs:attribute>

<xs:attribute name="Operation" type="StringLength1to16" use="optional">
</xs:attribute>
</xs:attributeGroup>
```

Z tej części pliku XMLSchema wynika, że *Criterion* ma atrybuty:

- *Name*, typu String, atrybut wymagany,
- *Value*, typu String, atrybut wymagany,
- *Reqiured*, typu boolean, atrybut wymagany,
- *Operation*, typu String, atrybut opcjonalny.

Za pomocą generatora Castor została wygenerowana klasa dla elementu *Criterion*, z metodami „get" i „set" dla wymienionych wyżej atrybutów:

```
public class Criterion implements java.io.Serializable {
/**
* A code representing the criterion on which to filter
*/
private java.lang.String _name;

/**
* The value of the criterion
*/
private java.lang.String _value;
```

```
/**
* A flag establishing if this criterion must be met
*/
private boolean _required;

/**
* keeps track of state for field: _required
*/
private boolean _has_required;

/**
* Other operations to be used as the filter (e.g. GT, LT, etc.).
*/
private java.lang.String _operation;

//----------------/
//- Constructors -/
//----------------/
public Criterion() {
      super();
} //-- golfCourse.translations.castor.Criterion()

//-----------/
//- Methods -/
//-----------/
/**
* @return the value of field 'name'.
*/
public java.lang.String getName()
{
      return this._name;
} //-- java.lang.String getName()

/**
* @return the value of field 'operation'.
*/
public java.lang.String getOperation()
{
      return this._operation;
} //-- java.lang.String getOperation()

/**
* @return the value of field 'required'.
*/
public boolean getRequired()
{
      return this._required;
} //-- boolean getRequired()

/**
* @return the value of field 'value'.
*/
public java.lang.String getValue()
{
      return this._value;
} //-- java.lang.String getValue()

/**
* Method hasRequired
*/
public boolean hasRequired()
```

```
{
      return this._has_required;
} //-- boolean hasRequired()
}
```

W klasie wygenerowanej dla wiadomości OTA_GolfCourseSearchRQ znajdują się metody „get” i „set” dla listy elementów *Criteria.*

```
public class OTA_GolfCourseSearchRQ implements java.io.Serializable {
 ...
    /**
      * Field _criteria
      */
     private golfCourse.translations.castor.Criteria _criteria;

 ...

    /**
      * Returns the value of field 'criteria'.
      *
      * @return the value of field 'criteria'.
      */
     public golfCourse.translations.castor.Criteria getCriteria()
     {
         return this._criteria;
     } //-- golfCourse.translations.castor.Criteria getCriteria()

 ...

    /**
      * Sets the value of field 'criteria'.
      *
      * @param criteria the value of field 'criteria'.
      */
     public void setCriteria(golfCourse.translations.castor.Criteria
criteria)
     {
         this._criteria = criteria;
     } //-- void setCriteria(golfCourse.translations.castor.Criteria)
}
```

Wszystkie klasy wygenerowane przez Castor mają metodę „marshal” i statyczną metodę „unmarshal”. Metody te służą do przekształcania instancji klas na tekst XML i zamiany tekstu XML na instancje odpowiednich klas Javy.

W celu utworzenia / zapisania wiadomości OTA należy utworzyć obiekt odpowiedniej, odpowiadającej wiadomości klasy, wypełnić go wartościami i wywołać jego metodę „marshal”.

Przykład utworzenia wiadomości OTA_GolfCourseSearchRQ za pomocą klas wygenerowanych przez Castor:

Chcemy utworzyć wiadomość xml OTA_GolfCourseSearchRQ. Castor wygenerował klasy *Criterion* i *OTA_GolfCourseSearchRQ*.
W celu utworzenia wiadomości XML należy stworzyć obiekty tych klas, wypełnić je danymi a następnie za pomocą Castor zamienić instancje tych klas na wiadomość XML.

```
\\ create instance of OTA_GolfCourseSearchRQ class
 OTA_GolfCourseSearchRQ ota = new OTA_GolfCourseSearchRQ();

 \\ set data to this instance
 ...

 \\ create instance of Criteria
 Criteria criteria = new Criteria();

\\ create Criterion for architect
Criterion architectCriterion = new Criterion();
architectCriterion.setName("Architect");
architectCriterion.setValue("Robert Jones");
architectCriterion.setRequired(false);

criteria.add(architectCriterion);

\\ create criterion for Slope
Criterion slopeCriterion = new Criterion();
slopeCriterion.setName("Slope");
slopeCriterion.setValue("110");
slopeCriterion.setRequired(true);
slopeCriterion.setOperation("LessThan");

criteria.add(slopeCriterion);

\\ put instance of Criteria to instance of class OTA_GolfCourseSearchRQ;
ota.setCriteria(criteria);
}
```

Zamiana instancji klasy na wiadomość XML:

```
...
Writer writer = new StringWriter();
try {

ota.marshal(writer);// operacja zamiany obiektu na tekt wiadomości xml

} catch(MarshalException e){
                  ...
} catch(ValidationException e2) {
                  ...
}
```

Wynikiem tej operacji będzie wiadomość XML:

```
<OTA_GolfCourseSearchRQ xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
   "http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseSearchRQ.xsd"
      EchoToken="54321"
      TimeStamp="2003-11-12T10:30:00"
      Target="Production" Version="1.001"
      SequenceNmbr="2432"
      PrimaryLangID="en" ID="FL4902"
      DetailResponse="true">
<Criteria>
<Criterion Name="Architect" Value="Robert Jones" Required="false"/>
<Criterion Name="Slope" Value="110" Required="true" Operation="LessThan"/>
</Criteria>
</OTA_GolfCourseSearchRQ>
```

W celu odczytania wiadomości OTA (zamiany tekstu wiadomości na obiekt odpowiedniej klasy) należy wywołać statyczną metodę „unmarshal" odpowiedniej dla wiadomości klasy z parametrem będącym tekstem XML. Metoda „unmarshal" zamienia tekst XML na instancje odpowiednich klas.

Przykład odczytania wiadomości XML (OTA_GolfCourseSearchRQ):

```
Reader reader = new StringReader(text); // text – tekst wiadomości xml
try {
OTA_GolfCourseSearchRQ
ota=(OTA_GolfCourseSearchRQ)OTA_GolfCourseSearchRQ.unmarshal(reader);
}catch(MarshalException e){
                  ...
} catch(ValidationException e2) {
                  ...
}
```

Po wykonaniu metody „unmarshal" mamy instancje klasy OTA_GolfCourseSearchRQ i możemy, korzystając z metod „get", pobierać dane z tej instancji.

Praca z generatorem Jastor:

Do pracy z ontologiami został wykorzystany generator Jastor. Jastor (podobnie jak Castor dla XMLSchema) generuje klasy dla ontologii i umożliwia zamianę instancji wygenerowanych klas na instancje ontologii, a także zamianę instancji ontologii na instancje wygenerowanych klas.

Jastor generuje interfejsy, ich implementacje i fabryki klas.

Np. dla ontologii GolfCourse Jastor wygenerował:

```
interface GolfCourse extends com.ibm.adtech.jastor.Thing
interface GolfCourseListener extends com.ibm.adtech.jastor.ThingListener
class GolfCourseImpl extends com.ibm.adtech.jastor.ThingImpl implements
golfCourse.ontologies.jastor.golfCourse.GolfCourse
class GolfCourseFactory extends com.ibm.adtech.jastor.ThingFactory
```

Jastor został wykorzystany do wygenerowania klas dla ontologii: GolfCourse, GolfCourseTeeTime, Contacts, Description, Price, Fee, AddressRecord, OutdoorLocation.

Przykład wygenerowania klasy za pomocą generatora Jastor:

Ontologia GolfCourseTeeTime ma parametry: golfCourseId (String), amount (float), currencyCode (String), startDate (String), endDate (String), maxPrice (float), numberOfHoles (integer), numberOfTimes (integer), lista fee (fee jest typu Fee).

Dla tej ontologii Jastor wygenerował interface *GolfCourseTeeTime* i jego implementację *GolfCourseTeeTimeImpl* z metodami get/set dla wymienionych wyżej parametrów.

```
public interface GolfCourseTeeTime extends com.ibm.adtech.jastor.Thing {

...
/**
* Gets the 'golfCourseID' property value
* @return   {@link java.lang.String}
* @see      #golfCourseIDProperty
*/
public java.lang.String getGolfCourseID()
      throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'golfCourseID' property value
* @param    {@link java.lang.String}
* @see      #golfCourseIDProperty
*/
public void setGolfCourseID(java.lang.String golfCourseID)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'numberOfTimes' property value
* @return   {@link java.math.BigInteger}
* @see      #numberOfTimesProperty
*/
public java.math.BigInteger getNumberOfTimes()
      throws com.ibm.adtech.jastor.JastorException;
```

```
/**
* Sets the 'numberOfTimes' property value
* @param    {@link java.math.BigInteger}
* @see      #numberOfTimesProperty
*/
public void setNumberOfTimes(java.math.BigInteger numberOfTimes)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'currencyCode' property value
* @return   {@link java.lang.String}
* @see      #currencyCodeProperty
*/
public java.lang.String getCurrencyCode()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'currencyCode' property value
* @param    {@link java.lang.String}
* @see      #currencyCodeProperty
*/
public void setCurrencyCode(java.lang.String currencyCode)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'amount' property value
* @return   {@link java.lang.Float}
* @see      #amountProperty
*/
public java.lang.Float getAmount()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'amount' property value
* @param    {@link java.lang.Float}
* @see      #amountProperty
*/
public void setAmount(java.lang.Float amount)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'numberOfHoles' property value
* @return   {@link java.math.BigInteger}
* @see      #numberOfHolesProperty
*/
public java.math.BigInteger getNumberOfHoles()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'numberOfHoles' property value
* @param    {@link java.math.BigInteger}
* @see      #numberOfHolesProperty
*/
public void setNumberOfHoles(java.math.BigInteger numberOfHoles)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'numberOfGolfers' property value
* @return   {@link java.math.BigInteger}
* @see      #numberOfGolfersProperty
```

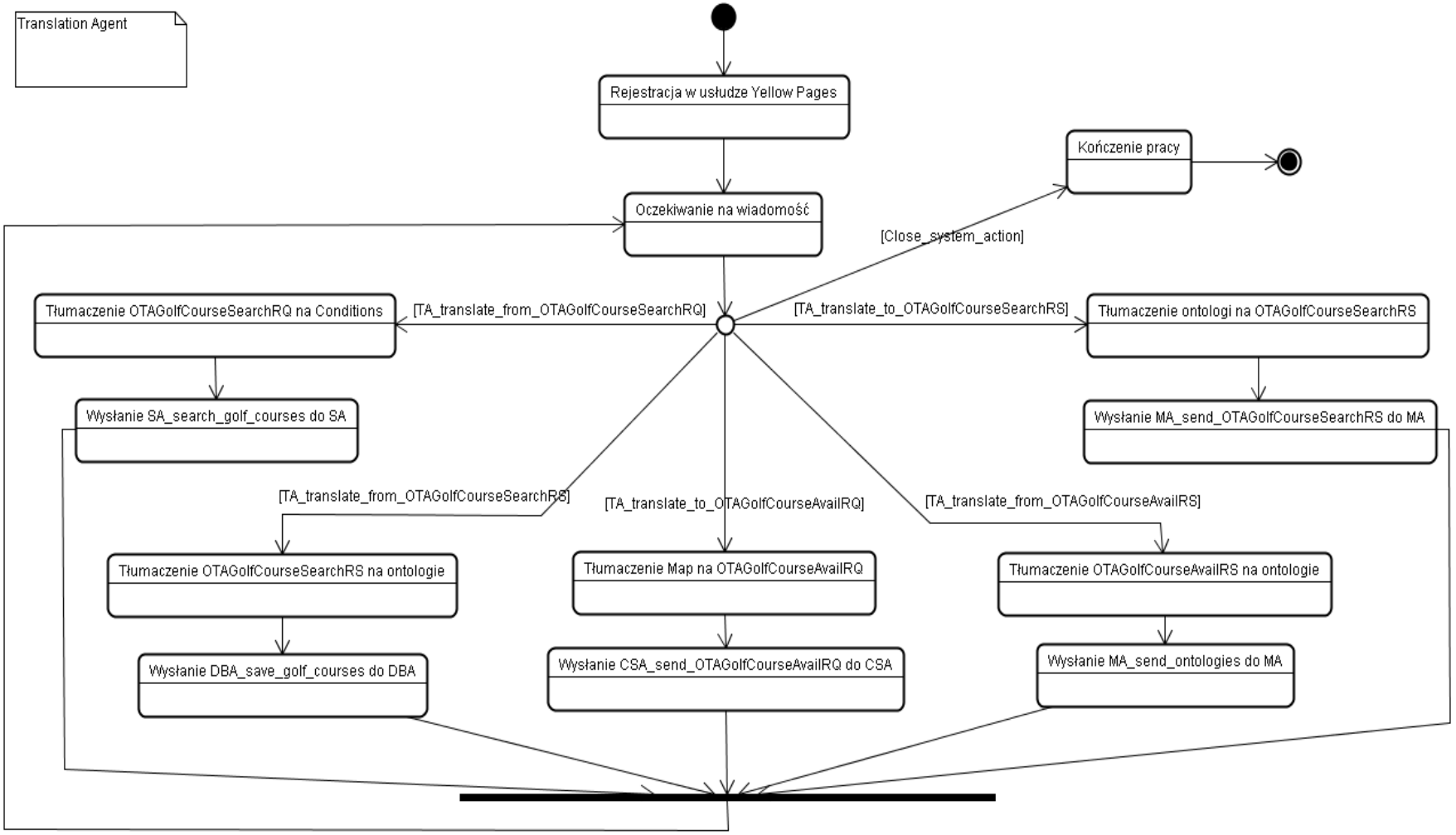
```
*/
public java.math.BigInteger getNumberOfGolfers()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'numberOfGolfers' property value
* @param    {@link java.math.BigInteger}
* @see      #numberOfGolfersProperty
*/
public void setNumberOfGolfers(java.math.BigInteger numberOfGolfers)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'maxPrice' property value
* @return   {@link java.lang.Float}
* @see      #maxPriceProperty
*/
public java.lang.Float getMaxPrice()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'maxPrice' property value
* @param    {@link java.lang.Float}
* @see      #maxPriceProperty
*/
public void setMaxPrice(java.lang.Float maxPrice)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'startDate' property value
* @return   {@link java.lang.String}
* @see      #startDateProperty
*/
public java.lang.String getStartDate()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'startDate' property value
* @param    {@link java.lang.String}
* @see      #startDateProperty
*/
public void setStartDate(java.lang.String startDate)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Gets the 'endDate' property value
* @return   {@link java.lang.String}
* @see      #endDateProperty
*/
public java.lang.String getEndDate()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Sets the 'endDate' property value
* @param    {@link java.lang.String}
* @see      #endDateProperty
*/
public void setEndDate(java.lang.String endDate)
            throws com.ibm.adtech.jastor.JastorException;

/**
```

```
* Get an Iterator the 'fee' property values.  This Iteartor
* may be used to remove all such values.
* @return   {@link java.util.Iterator} of {@link
* com.ibm.adtech.jastor.Thing}
* @see      #feeProperty
*/
public java.util.Iterator getFee()
            throws com.ibm.adtech.jastor.JastorException;

/**
* Adds a value for the 'fee' property
* @param    The {@link com.ibm.adtech.jastor.Thing} to add
* @see      #feeProperty
*/
void addFee(com.ibm.adtech.jastor.Thing fee)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Adds an anonymous value for the 'fee' property
* @return   The anoymous {@link com.ibm.adtech.jastor.Thing} created
* @see      #feeProperty
*/
public com.ibm.adtech.jastor.Thing addFee()
            throws com.ibm.adtech.jastor.JastorException;


/**
* Adds a value for the 'fee' property.  This
* method is equivalent constructing a new instance of
* {@link com.ibm.adtech.jastor.Thing} with the factory
* and calling addFee(com.ibm.adtech.jastor.Thing fee)
* The resource argument have rdf:type http://www.w3.org/2000/01/rdf-
schema#Resource.  That is, this method
* should not be used as a shortcut for creating new objects in the model.
* @param    The {@link om.hp.hpl.jena.rdf.model.Resource} to add
* @see      #feeProperty
*/
public com.ibm.adtech.jastor.Thing
addFee(com.hp.hpl.jena.rdf.model.Resource resource)
            throws com.ibm.adtech.jastor.JastorException;

/**
* Removes a value for the 'fee' property.  This method should not
* be invoked while iterator through values.  In that case, the remove()
method of the Iterator
* itself should be used.
* @param    The {@link com.ibm.adtech.jastor.Thing} to remove
* @see      #feeProperty
*/
public void removeFee(com.ibm.adtech.jastor.Thing fee)
            throws com.ibm.adtech.jastor.JastorException;

...
}
```

Interfejsy wygenerowane przez Jastor zawsze dziedziczą z interfejsu `com.ibm.adtech.jastor.Thing`. Klasy wygenerowane przez Jastor dziedziczą z

klasy `com.ibm.adtech.jastor.ThingImpl`, która jest implementacją interfejsu `com.ibm.jastor.Thing`.

Praca z Jastorem jest bardzo podobna do pracy z Castorem. Najpierw Jastor generuje klasy dla ontologi (tak jak Castor dla XMLSchema), potem pracujemy z instancjami tych klas.
Możemy zamienić instancje klas wygenerowanych przez Jastor na instancje ontologii (tak jak instancje klas wygenerowanych przez Castor możemy zamienić na tekst XML), a także możemy dokonać konwersji instancji ontologi na instancje wygenerowanych klas Javy (tak jak tekst XML na instancje klas wygenerowanych przez Castor).
Agent TA podczas tłumaczenia wiadomości korzysta z wygenerowanych przez Castor i Jastor klas. Zatem praca agenta TA polega na „przepisywaniu" wartości z obiektów jednych klas do obiektów innych klas.


Translation Agent
Rejestracja w usłudze Yellow Pages
Oczekiwanie na wiadomość
Kończenie pracy
[Close_system_action]
Tłumaczenie OTAGolfCourseSearchRQ na Conditions
[TA_translate_from_OTAGolfCourseSearchRQ]
[TA_translate_to_OTAGolfCourseSearchRS]
Tłumaczenie ontologi na OTAGolfCourseSearchRS
Wysłanie SA_search_golf_courses do SA
Wysłanie MA_send_OTAGolfCourseSearchRS do MA
[TA_translate_from_OTAGolfCourseSearchRS]
[TA_translate_to_OTAGolfCourseAvailRQ]
[TA_translate_from_OTAGolfCourseAvailRS]
Tłumaczenie OTAGolfCourseSearchRS na ontologie
Tłumaczenie Map na OTAGolfCourseAvailRQ
Tłumaczenie OTAGolfCourseAvailRS na ontologie
Wysłanie DBA_save_golf_courses do DBA
Wysłanie CSA_send_OTAGolfCourseAvailRQ do CSA
Wysłanie MA_send_ontologies do MA

Wiadomości odbierane przez TA:

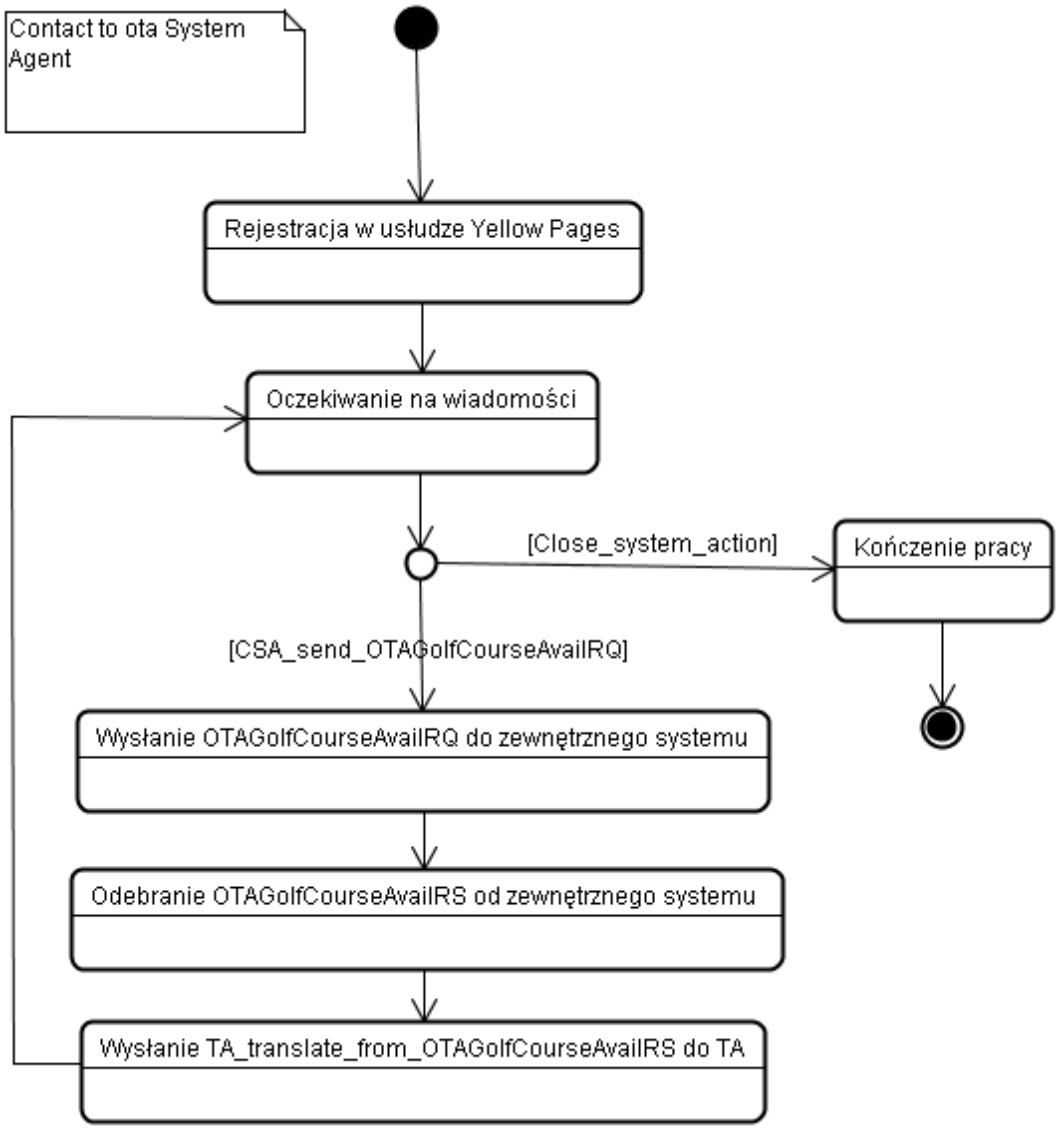
message	Działanie agenta TA.
TA_translate_from_OTAGolfCourseSearchRQ	Agent TA tłumaczy OTAGolfCourseSearchRQ na Conditions - prostą strukturę pomocniczą do zapisu informacji o kryteriach pola golfowego. Po wykonaniu tłumaczenia wyniki są wysyłane do agenta SA (scenariusz 1). Czyli agent TA zamienia obiekt klasy OTA_GolfCourseSearchRQ (klasa wygenerowana przez Castor) na obiekt klasy pomocniczej Conditions.
TA_translate_from_OTAGolfCourseSearchRS	Agent TA tłumaczy OTAGolfCourseSearchRS na listę instancji ontologii, a następnie wysyła wyniki do agenta DBA (scenariusz 3) lub do agenta MA (scenariusz 2). Czyli agent TA pobiera dane z obiektu OTA_GolfCourseSearchRS (klasa wygenerowana przez Castor) i zamienia je na listę obiektów klasy GolfCourse (klasa wygenerowana przez Jastor).
TA_translate_from_OTAGolfCourseAvailRS	Agent TA tłumaczy OTAGolfCourseAvailRS na listę instancji ontologii, a następnie wysyła wyniki do agenta MA. (scenariusz 2). Czyli agent TA pobiera dane z obiektu klasy OTA_GolfCourseAvailRS (klasa wygenerowana przez Castor) i zamienia je na listę obiektów klasy GolfCourseTeeTime (klasa wygenerowana przez Jastor).

TA_translate_to_OTAGolfCourseSearchRS	Wiadomość zawiera instancje ontologii opisujące pole golfowe. Agent TA tłumaczy otrzymane ontologie na OTAGolfCourseSearchRS i wysyła wynik do MA. (scenariusz 1). Czyli agent TA zamienia listę obiektów klasy GolfCourse (klasa wygenerowana przez Jastor) na obiekt klasy OTA_GolfCourseSearchRS (klasa wygenerowana przez Castor)
TA_translate_to_OTAGolfCourseAvailRQ	Wiadomość zawiera Map – pytanie o dostępność pola golfowego. Agent TA tłumaczy Map na wiadomość OTAGolfCourseAvailRQ a następnie wysyła wynik do agenta CSA (scenariusz 2). Czyli agent TA zamienia obiekt klasy pomocniczej Map na obiekt klasy OTA_GolfCourseAvailRQ (wygenerowanej przez Castor)
Close_system_action	Agent TA kończy swoją działalność.

W systemie będzie jeden agent MA, jeden agent DBA i kilku agentów TA, SA i CSA.

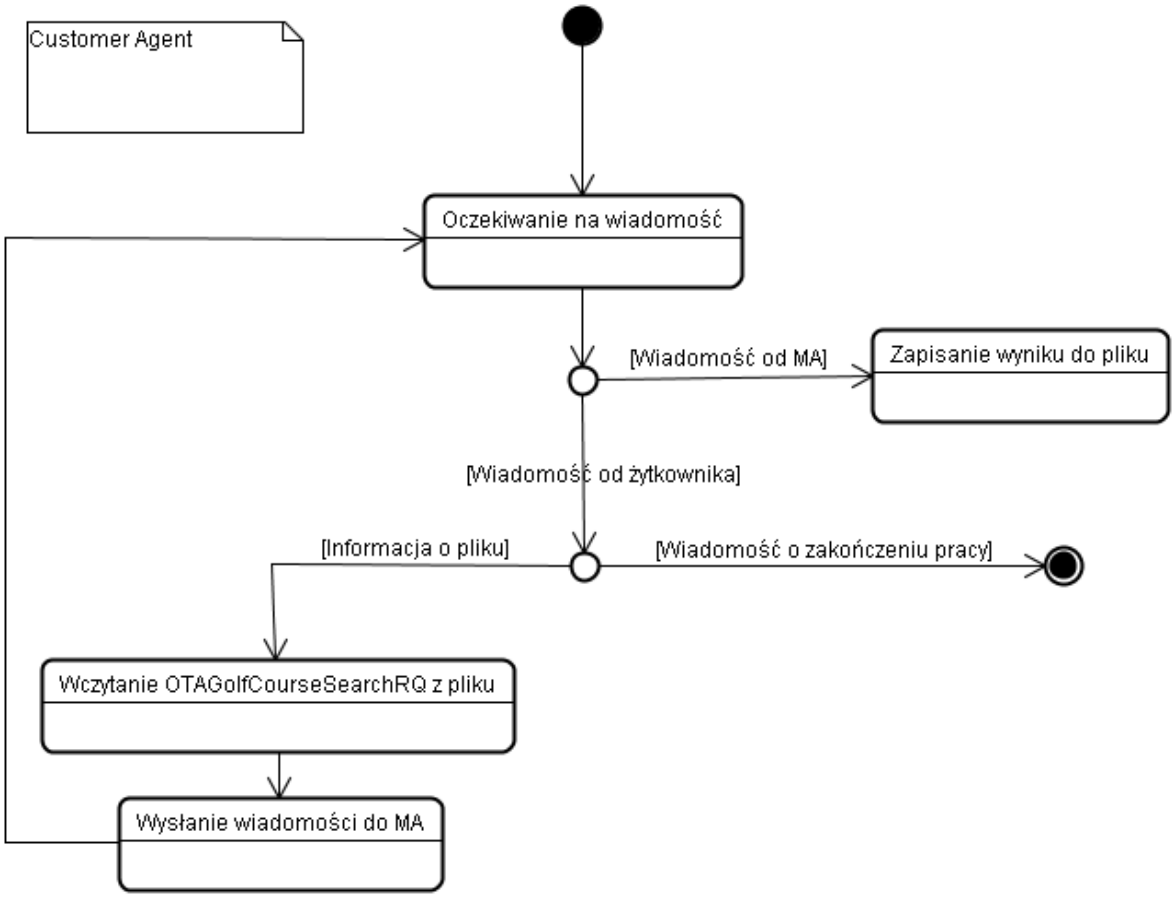
Agenci odpowiedzialni za wymianę informacji z innymi systemami posługującym się wiadomościami OTA – CSA (*Contact to external System Agent*)

W chwili obecnej w systemie jest tylko jeden taki agent lub raczej agent zastępujący agenta CSA. Agent taki po otrzymaniu wiadomości OTAGolfCourseAvailRQ tylko wypisuje ją na konsoli a następnie wczytuje wiadomość OTAGolfCourseAvailRS z pliku i wysyła ją, jako odpowiedź od zewnętrznego źródła.

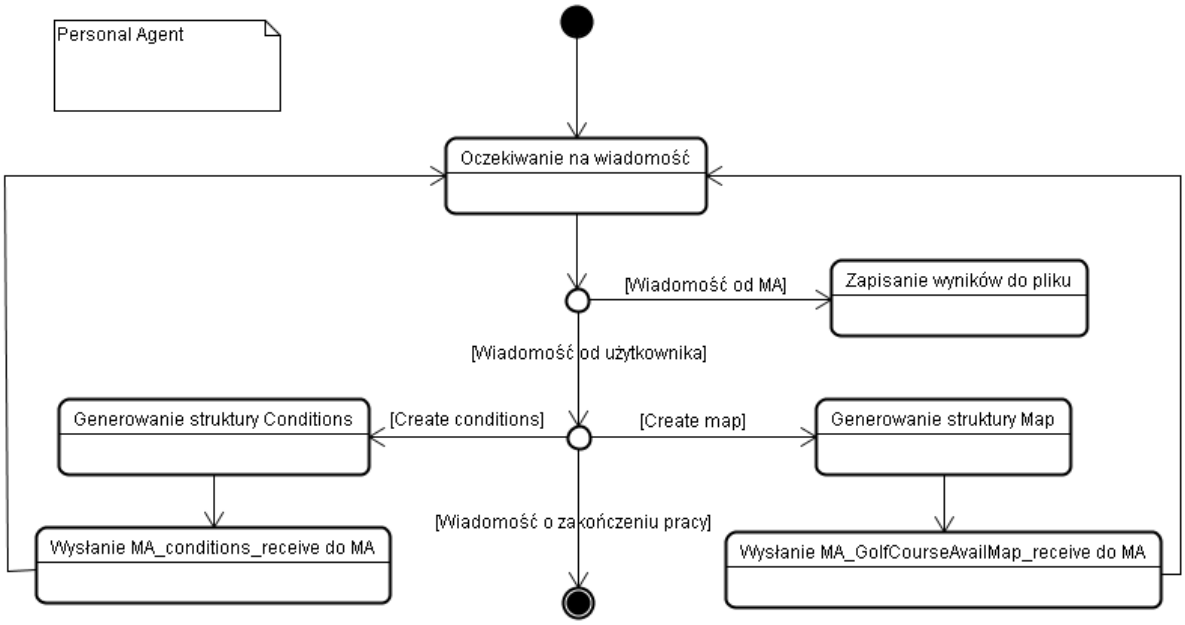


Agenci – klienci:

(Customer Agent - CA) – agenci wysyłający wiadomości OTA do głównego agenta MA. Nie są częścią tworzonego systemu, mają za zadanie „udawać" klientów w celu pokazania działania systemu. Do agenta CA zostaje wysłana wiadomość (za pośrednictwem GUI będącego częścią JADE) ze ścieżką dostępu do pliku XML, zawierającego wiadomość OTA. CA odczytuje treść wiadomości z pliku i wysyła ją do MA. Po otrzymaniu odpowiedzi, zapisuje wynik (wiadomość OTA) do pliku result.xml.

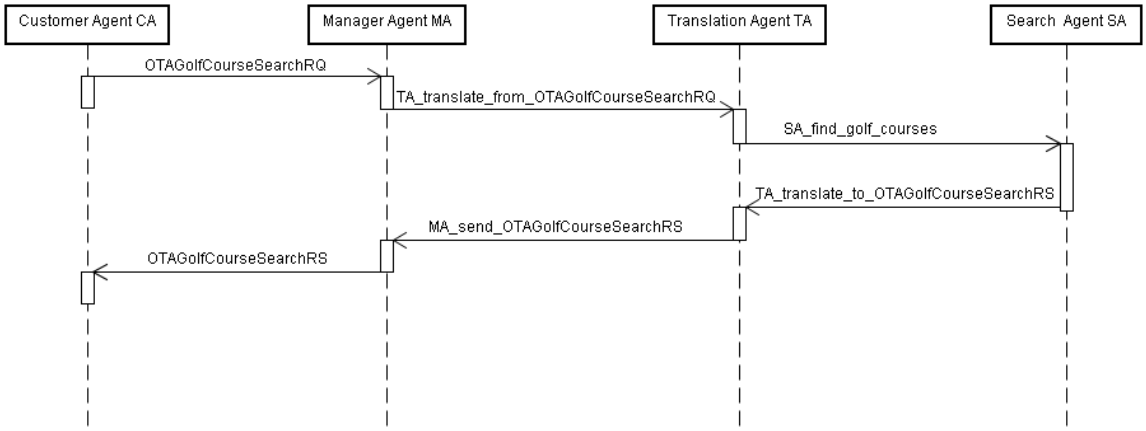


(Personal Agent – PA) – agent naśladujący inny zewnętrzny system korzystający z ontologii, np. system TSS. Celem stworzenia tego agenta jest pokazanie działania systemu podczas wykonywania scenariusza 2. Agent PA wysyła do systemu (agenta MA) wiadomość z Conditions (kryteriami opisującymi dane pole golfowe) lub z Map (pytanie o dostępność). System wysyła do agenta PA wiadomości zawierające zbiór instancji ontologii odpowiednio GolfCourse i GolfCourseTeeTime. Agent PA zapisuje otrzymane wyniki do pliku.



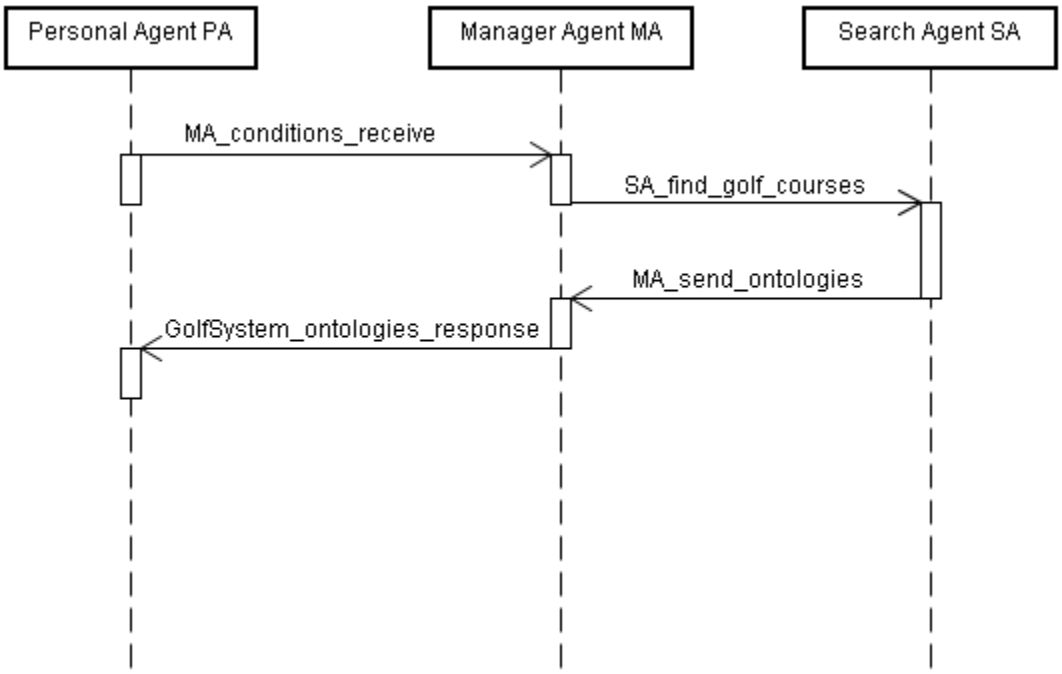
Schemat działania systemu:

- Scenariusz 1:

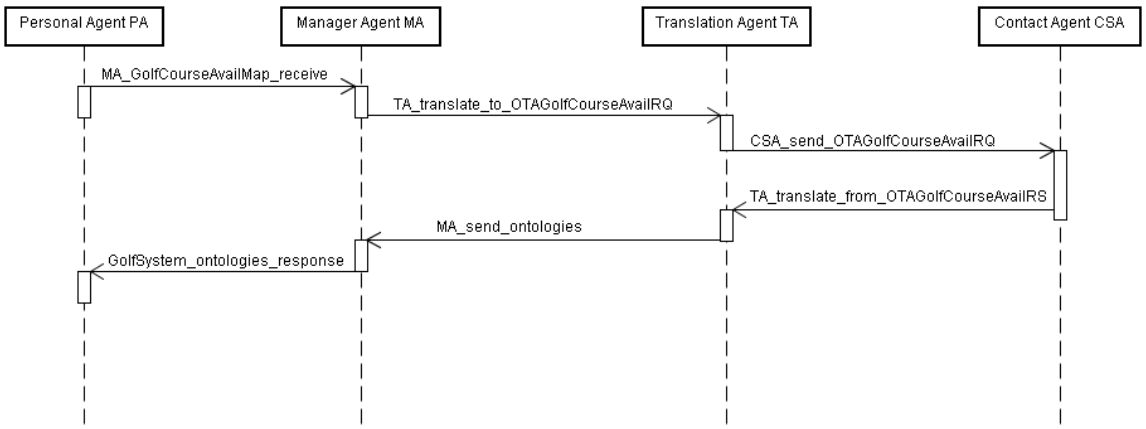


- Scenariusz 2:

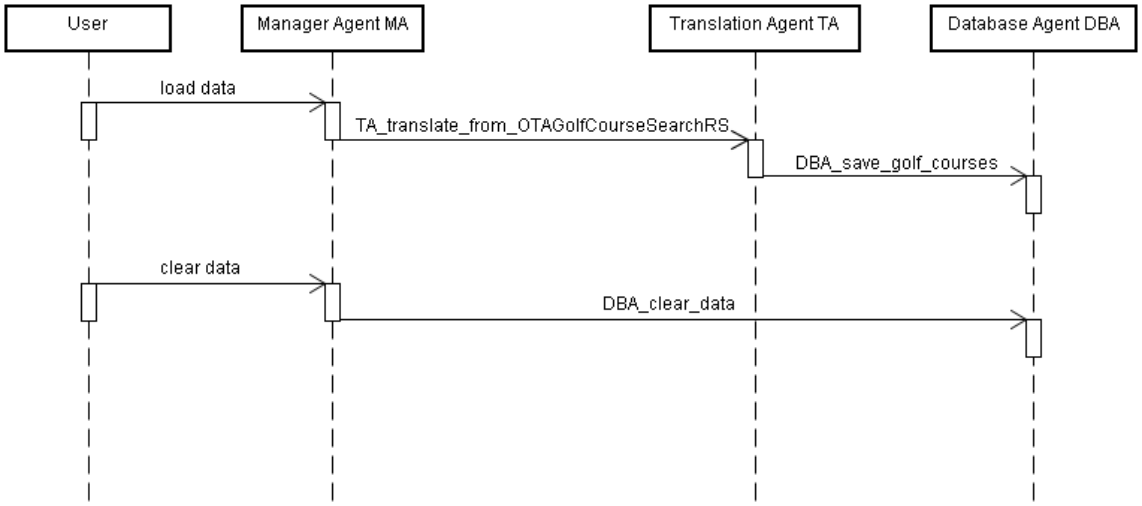
  - pytanie o pole golfowe



  - pytanie o dostępność pola golfowego

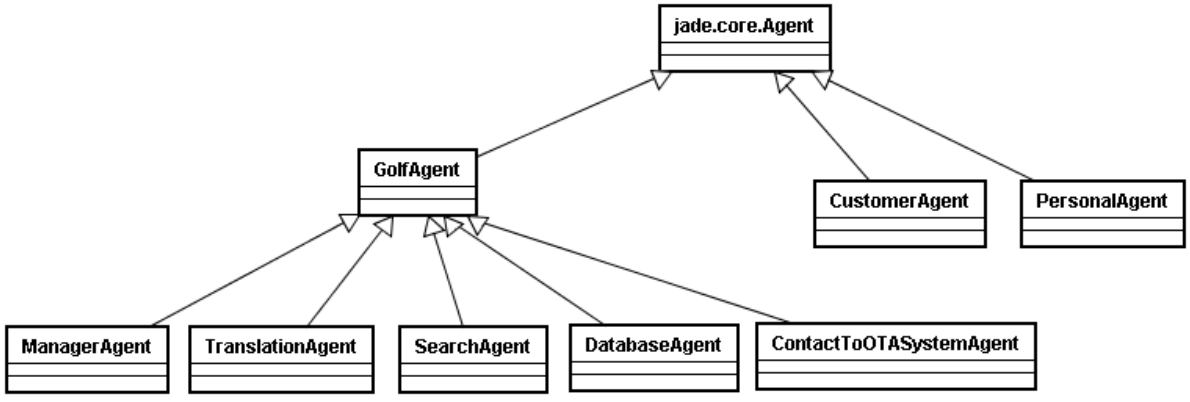


- Scenariusz 3:



6.6 Implementacja agentów w systemie.

W projekcie została wykorzystana, opisana wcześniej, platforma JADE ([18]).
Wszyscy agenci systemu GolfSystem dziedziczą z klasy GolfAgent, dziedziczącej z klasy jade.core.Agent.
Agenci pomocniczy - agenci klienci są obiektami klas dziedziczących z jade.core.Agent



Agenci MA, TA, SA, CSA i DBA rejestrują się także w usłudze Yellow Pages, czyli rejestrują zadania, jakie są w stanie wykonać w postaci nazwanych usług. Dzięki tej usłudze

agenci odnajdują agentów, którzy potrafią zrealizować dany typ zadania. Np. agent MA dzięki tej usłudze odnajduje agentów TA potrafiących przetłumaczyć daną wiadomość OTA.

W projekcie agenci MA, TA, SA, CSA i DBA po wykonaniu zadań początkowych (rejestracja w usłudze Yellow Pages, wczytanie modeli do pamięci, w przypadku agenta MA także utworzenie pozostałych agentów), przechodzą w stan uśpienia i oczekują na nadejście wiadomości. Po otrzymaniu wiadomości i wykonaniu zadań z nią związanych (np. przetłumaczenie wiadomości), o ile nie nadejdzie kolejna, ponownie przechodzą w stan uśpienia.

Agenci porozumiewają się pomiędzy sobą w języku ACL. Komunikacja polega na wysyłaniu wiadomości (komunikatów) ACL.

Do komunikacji pomiędzy agentami na platformie JADE niezbędna jest znajomość identyfikatora AID agenta, do którego ma być wysłana wiadomość. Identyfikator taki można uzyskać dzięki usługom White Pages i Yellow Pages. W projekcie wszyscy agenci są zarejestrowani w usłudze Yellow Pages i dzięki niej uzyskują identyfikatory pozostałych agentów.

6.7. Wyjątki i błędy w pracy systemu.

W systemie wyróżniamy trzy rodzaje błędów:

1. Błędy w pracy systemu niezwiązane z parsowaniem XML i z pracą z instancjami ontologii (np. brak agentów TA w systemie).
2. Błędy podczas parsowania plików XML lub podczas pracy z instancjami ontologii, w sytuacji gdy użytkownik oczekuje od systemu odpowiedzi (np. użytkownik wysłał do systemu zapytanie o pole golfowe)
3. Błędy podczas parsowania plików XML lub podczas pracy z instancjami ontologii, w sytuacji gdy użytkownik nie oczekuje odpowiedzi (np. użytkownik wysłał do systemu polecenie „load data”).

W przypadku wystąpienia dowolnego z opisanych powyżej błędów, agent systemu wysyła wiadomość o błędzie do agenta MA i agent MA wypisuje na konsoli informację o błędzie.

Po otrzymaniu (i wyświetleniu na konsoli) informacji o błędzie pierwszego typu, agent MA podejmuje działania mające na celu zakończenie pracy systemu.

Po otrzymaniu (i wyświetleniu na konsoli) informacji o błędzie drugiego typu, agent MA generuje odpowiedź o błędzie i wysyła ją do użytkownika. Jeśli użytkownik wysłał do systemu wiadomość OTAGolfCourseSearchRQ i podczas udzielania odpowiedzi nastąpił błąd, Manager Agent generuje odpowiedź w formacie XML OTAGolfCourseSearchRS z informacją o błędzie. Jeśli użytkownik wysłał do systemu zapytanie nie w formacie XML (np. Conditions lub Map) wtedy Manager Agent generuje odpowiedź GolfSystem_error_message. Błędy typu drugiego i trzeciego nie powodują zamknięcia systemu.

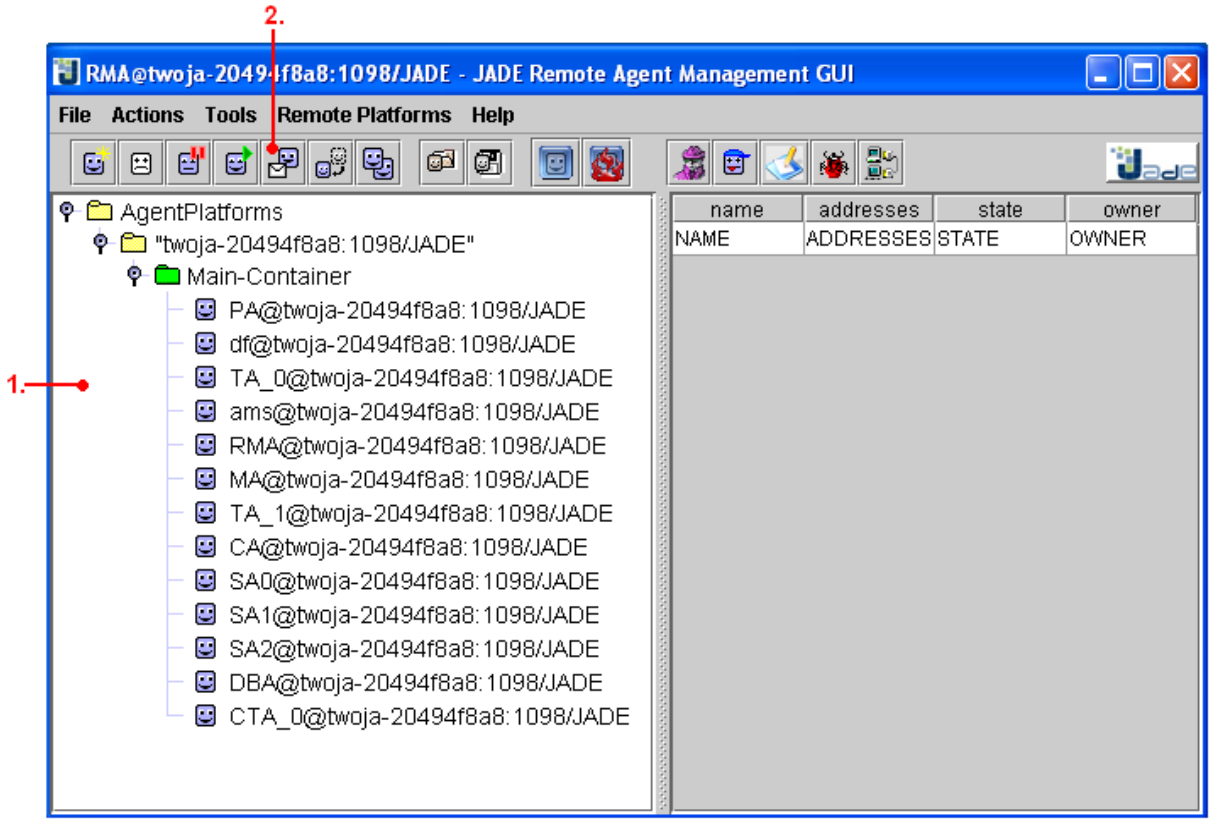
6.8 Dane.

Dane są opisane za pomocą, omówionych w poprzednich rozdziałach, ontologii. Część modeli (ontologie opisujące koncept pola golfowego, opłaty, ceny, gry itd.) są przechowywane w pamięci. Dane opisujące konkretne pola (wartości) są również opisane za pomocą ontologii i przechowywane w bazie danych postgreSQL.

Przy pierwszym starcie systemu dane o polach golfowych zostaną wczytane do bazy danych. Użytkownik musi jedynie utworzyć bazę danych i wypełnić prawidłowo plik konfiguracyjny (nazwa bazy, hasło, użytkownik).

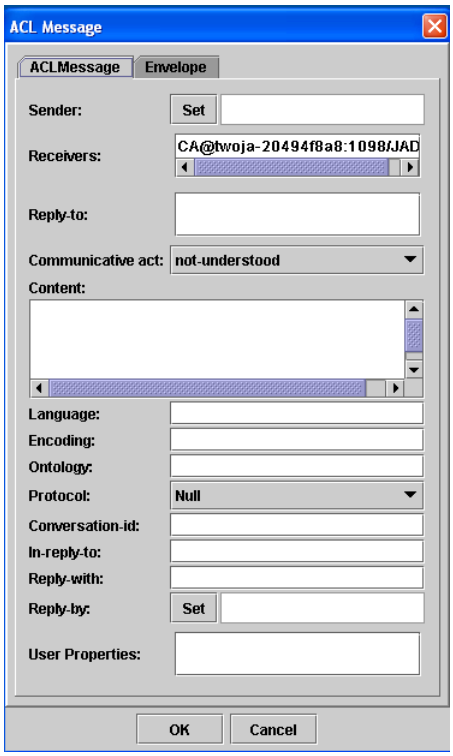
6.9 Projekt – opis dla użytkownika.

System nie posiada interfejsu użytkownika, należy korzystać z GUI dostarczonego przez platformę JADE:



1. Lista działających agentów
2. Przycisk do wysyłania wiadomości

GUI pokazuje listę wszystkich działających agentów na danej platformie. Aby wysłać wiadomość do agenta, należy wybrać go z listy (zaznaczyć pojedynczym kliknięciem myszy) i nacisnąć przycisk (2) do wysyłania wiadomości.



Okienko do wysyłania wiadomości ACL.

Aby wysłać wiadomość do agenta w systemie *GS* należy podać „Conversation-id" i „Content" (czyli treść wiadomości).

Użytkownik systemu *GS* może wysyłać wiadomości do agenta MA (*Manager Agent*) i agentów pomocniczych: CA (*Customer Agent* – agent zastępujący użytkownika, wczytuje a następnie wysyła do systemu *GS* wiadomość OTA), PA (*Personal Agent* – agent naśladujący inny zewnętrzny system np. TSS).

Wiadomości (treść i id), jakie użytkownik może wysłać do agentów systemu, są zdefiniowane w plikach konfiguracyjnych.

Fragment pliku dla agenta MA:

```
# Agent-based Golf System
# Agents MA configuration file
############################################################

CONVERSATION_ID=1
LOAD_DATA=load data
CLOSE_SYSTEM=close system
REMOVE_DATA=clear database
```

Wysyłanie wiadomości – przykłady:

Przykład 1:

Jak zakończyć pracę systemu?

- użytkownik wybiera agenta MA z listy agentów,
- naciska przycisk wysyłania wiadomości,
- w pole „Conversation-id” wpisuje „1”, a w pole „Content”: „close system”.

Przykład 2:

Usuwanie danych z repozytorium.

- użytkownik wybiera agenta MA z listy agentów,
- naciska przycisk wysyłania wiadomości,
- w pole „Conversation-id” wpisuje „1”, a w pole „Content”: „clear database”.

Przy uruchamianiu systemu zostanie uruchomiona również konsola. Na konsoli będą wypisywane podstawowe informacje o działaniu agentów (np. informacje o rozpoczęciu pracy, o otrzymywanych wiadomościach) a także informacje o błędach systemu.
Wynik pracy systemu zostanie zapisany w pliku CA_results.xml

Po uruchomieniu systemu należy poczekać, aż agent MA utworzy pozostałych agentów. Informacje o utworzeniu agentów pojawią się na konsoli.

Przykład działania systemu:

Użytkownik (agent CA) wysyła do systemu pytanie, będące wiadomością XML, o pole golfowe

1. Przygotowanie pliku z wiadomością OTA_GolfCourseSearchRQ.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseSearchRQ xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseSearchRQ.xsd"
EchoToken="54321" TimeStamp="2003-11-12T10:30:00" Target="Production" Version="1.001"
SequenceNmbr="2432" PrimaryLangID="en" ID="FL4902" DetailResponse="true">
<Criteria>
        <Criterion Name="Architect" Value="Robert Jones" Required="true"/>
</Criteria>
</OTA_GolfCourseSearchRQ>
```

Poszukiwane są pole golfowe, których architektem jest „Robert Jones”.

2. Wysłanie wiadomości do agenta CA ze ścieżką do pliku OTA_GolfCourseSearchRQ.xml.

Rozpoczyna się działanie systemu.

Na konsoli pojawiają się komunikaty o pracy systemu, np.:

```
CA received message from user.
CA is reading file OTA_GolfCourseSearchRQ.xml.
CA sent message to MA.
MA received message from customer CA.
MA sent customer message to TA.
TA_0 received OTA_GolfCourseSearchRQ message.
TA_0 translated OTA_GolfCourseSearchRQ message to set of conditions.
TA_0 sent conditions to SA.
SA_2 received conditions.
SA_2 build query.
SA_2 executed query.
SA_2 found 2 golf courses.
SA_2 created response message with ontologies – query results.
SA_2 sent results to TA.
TA_1 received ontologies from SA.
TA_1 translated ontologies to OTA_GolfCourseSearchRS message.
TA_1 sent OTA_GolfCourseSearchRS message to MA.
MA sent response to customer CA.
CA received message from MA.
CA has written the answer to file result.xml
```

Zawartość pliku CA_result.xml po zapisaniu wyniku przez agenta CA:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_GolfCourseSearchRS xmlns="http://www.opentravel.org/OTA/2003/05"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opentravel.org/OTA/2003/05 OTA_GolfCourseSearchRS.xsd"
EchoToken="54321" TimeStamp="2003-11-12T10:30:15" Target="Production" Version="1.002"
SequenceNmbr="2433" PrimaryLangID="en">
<Success/>
<GolfCourses>
<GolfCourse ID="FL1234" Name="Sea Grass Golf Resort">
        <Address>
                <CityName>Jupiter</CityName>
                <PostalCode>21921</PostalCode>
                <County>Palm Beach</County>
                <StateProv StateCode="FL"/>
                <CountryName Code="US"/>
        </Address>
        <Phone AreaCityCode="444" PhoneNumber="423-8954"/>
        <Traits>
                <Trait Name="Architect" Value="Robert Jones"/>
                <Trait Name="Singles Confirmed" Value="Yes"/>
                <Trait Name="ADA Challenged" Value="Wheelchair"/>
                <Trait Name="Slope" Value="110"/>
                <Trait Name="Metal Spikes" Value="No"/>
                <Trait Name="Caddies Available" Value="No"/>
                <Trait Name="Yardage" Value="6345"/>
                <Trait Name="Personal Carts Permitted" Value="No"/>
                <Trait Name="Fivesome" Value="No"/>
                <Trait Name="Grass Type" Value="Bermuda"/>
```

```
        </Traits>
</GolfCourse>
<GolfCourse ID="FL4321" Name="Beach Side Golf Resort">
        <Address>
                <CityName>Palm Beach Gardens</CityName>
                <PostalCode>21932</PostalCode>
                <County>Palm Beach</County>
                <StateProv StateCode="FL"/>
                <CountryName Code="US"/>
        </Address>
        <Phone AreaCityCode="444" PhoneNumber="423-2876"/>
        <Traits>
                <Trait Name="Architect" Value=" Robert Jones "/>
                <Trait Name="Singles Confirmed" Value="Yes"/>
                <Trait Name="ADA Challenged" Value="Wheelchair"/>
                <Trait Name="Slope" Value="112"/>
                <Trait Name="Metal Spikes" Value="Yes"/>
                <Trait Name="Caddies Available" Value="Yes"/>
                <Trait Name="Yardage" Value="7102"/>
                <Trait Name="Fivesome" Value="Yes"/>
                <Trait Name="Grass Type" Value="Rye"/>
        </Traits>
</GolfCourse>
</GolfCourses>
</OTA_GolfCourseSearchRS>
```

7. Podsumowanie

W pracy zostały omówione zastosowania agentów programowych i systemów agentowych w handlu elektronicznym. Jako przykład implementacji takiego systemu, został stworzony system GS (*Golf System*) do dostarczania informacji o polach golfowych. Poszukiwanym „towarem” jest możliwość gry w golfa.

System został zaprojektowany tak by dostarczać informacje jak największej liczbie użytkowników i pobierać dane z jak największej liczby systemów zewnętrznych. *GS* wysyła i odbiera wiadomości w jednym z powszechnie używanych standardów OTA. Dane w systemie są zapisane za pomocą ontologii. Ontologie zostały zaprojektowane na podstawie wyników uzyskanych z analizy wiadomości OTA.

W przyszłości *GS* będzie częścią powstającego Systemu Wspomagania podróży (*TSS – Travel Support System*). *TSS* jest systemem mającym w sposób całościowy wspierać użytkownika przy planowaniu podróży. Został zaprojektowany tak, aby służyć rozległą pulą usług: rezerwowanie miejsca w hotelu, planowanie trasy podróży i wybór środków transportu. *TSS* będzie również realizować funkcje informacyjne takie jak: dostarczanie programu kinowego lub sugerowanie restauracji z określonym rodzajem kuchni. Gdy w przyszłości *GS* stanie się częścią systemu *TSS* to system ten będzie mógł także zaplanować grę na polu golfowym.

8. Literatura:

[1] Agnieszka Cieślik, Maria Ganzha, Marcin Paprzycki, Developing Open Travel Alliance-based Ontology of Golf, in: J. Cordeiro (et. al.), Proceedings of the 2008 WEBIST Conference, INSTICC Press, Setubal, Portugal, 2008, 62-69

[2] Minor Gordon, Marcin Paprzycki (2005) „Designing Agent Based Travel Support System"

[3] Maciej Gawinecki, Zygmunt Vetulani, Minor Gordon, Marcin Paprzycki (2005) „Representing Users in a Travel Support System"

[4] Maciej Gawinecki, Mateusz Kruszyk, Marcin Paprzycki (2005) „Ontology-based Stereotyping in a Travel Support System"

[5] Michał Szymczak, Maciej Gawinecki, Mladenka Vukmirovic, Marcin Paprzycki „Ontological reusability In state-of-the-art. Semantic languages"

[6] „Utilizing Semantic Web and Software Agents in a Travel Support System" – Maria Ganzha, Maciej Gawineski, Marcin Paprzycki,

[7] „Agenci programowi jako metodologia tworzenia oprogramowania" - Marcin Paprzycki Computer Science Department, Oklahoma State University, Tulsa, OK 74106 USA

[8] "Zastosowania systemów agentowych" – Michał Nowakowski

[9] "Agent nie tylko u Twoich drzwi" – Violetta Galant Wrocław, Marcin Paprzycki Poznań, 29 grudnia, 2001

[10] „Modelowanie użytkownika na podstawie interakcji z systemem opartym o technologie WWW" – praca magisterska Macieja Gawineckiego,

[11] Tom Gruber: „What is an Ontology?” http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[12] Marek Łabuzek, Wydziałowy Zakład Informatyki, Politechnika Wrocławska: „Wykorzystanie metamodelowania do specyfikacji ontologii znaczenia opisów rzeczywistości”

[13] Andrzej Bassara: „Ontology Engineering”, "I weź tu dogadaj się – Ontologie”

[14] RDF: www.w3.org/RDF

[15] RDFS: http://www.w3.org/TR/rdf-schema/

[16] RDFS tutorial: http://www-sop.inria.fr/acacia/soft/corese/tutorial.php

[17] FIPA: http://www.fipa.org/

[18] JADE: http://jade.tilab.com/

[19] JENA: http://jena.sourceforge.net

[20] JENA: http://www.hpl.hp.com/semweb/jena2.htm

[21] OpenTravel Alliance: http://www.opentravel.org/

[22] OTA_MessageUserGuide2006V1.0

[23] Jastor: http://jastor.sourceforge.net/

[24] Castor: http://www.castor.org/

[25] JADE Programmer’s Guide: http://jade.tilab.com/doc/programmersguide.pdf

[26] Paweł Kaczmarek, Wydział Matematyki i Informatyki, Uniwersytet Adama Mickiewicza: „Multimodalna komunikacja agentów programowych z użytkownikiem”

[27] Anna Borkowska, asystent w Katedrze Informatyki Gospodarczej SGH, artykuł: „Inteligentni agenci w handlu elektronicznym”.

[28] M.V. Sinmao 24 November 1999: „Intelligent Agents & E-Commerce” http://www.cis.udel.edu/~wchen/Sinmaom.htm