POLITECHNIKA WARSZAWSKA
Wydział Matematyki i Nauk Informacyjnych

P R A C A M A G I S T E R S K A

Szymon Pisarek

„Ontologicznie zorientowane przeszukiwanie Internetu”

Opiekun pracy
Prof. Marcin Paprzycki

WARSZAWA 2005

Autor pragnie podziękować
Panu prof. Marcinowi Paprzyckiemu
oraz Pani dr Marii Ganzha
za pomoc w realizacji pracy magisterskiej

Spis Treści:

1. Wstęp

Ciągły przyrost ilości informacji dostępnej w Internecie ma zarówno pozytywne jak i negatywne skutki. Z jednej strony znajdują się tam „prawie wszystkie" istotne informacje z prawie każdej dziedziny, z drugiej zaś strony ich ilość jest tak wielka, iż bardzo często mamy problem ze znalezieniem tych danych, które są nam „naprawdę" potrzebne. Wielu naukowców, w tym P. Maes [Maes 1994], twierdzi, że inteligentni agenci programowi mogą być odpowiedzią na problem zalewu informacją. Jeśli rzeczywiście ma się tak stać to koniecznym jest przekształcenie informacji dostępnej w Internecie w postać, która będzie czytelna dla tychże agentów. Jedną z możliwych dróg osiągnięcia tego celu jest *Semantic Web* (Rozdział 2), czyli Internet Semantyczny, w którym informacja jest opisana ontologicznie. W tym celu możemy posłużyć się językami opisu zasobów, takimi jak: *Resource Description Framework* (RDF) (podrozdział 2.3.1) czy *Ontology Web Language* (OWL) (podrozdział 2.3.3). Niestety, w chwili obecnej praktycznie nie ma dostępnych danych w formacie RDF/OWL. Jedno z niewielu większych repozytoriów to ChefMoz, jednakże korzystanie z tychże danych bywa bardzo kłopotliwe [Gawinecki, 2005a].

Aby przetestować hipotezę P. Maes, o skuteczności agentów programowych postanowiliśmy stworzyć agentowy *System Wspomagania Podróży*, w którym dane opisane będą językiem RDF. Celem niniejszej pracy jest omówienie rozwiązań zastosowanych przy gromadzeniu danych dla systemu. W następnym rozdziale przedstawiona zostanie koncepcja *Sieci Semantycznej* (*Internetu Semantycznego*), omówione zostaną główne technologie nowej generacji Internetu. Następnie w Rozdziale 3 omówiony zostanie całościowy obraz systemu oraz przedstawieni zostaną jego agenci. Rozdział 4 całkowicie poświęcony jest podsystemowi odpowiedzialnemu za gromadzenie danych (*Content Collection Subsystem, CCS*). Przedstawione zostaną szczegółowe informacje na temat komunikacji agentów, przyjrzymy się bliżej agentom podsystemu *CCS* oraz zaproponowane zostaną możliwości usprawnienia tegoż podsystemu. Rozdział 4 zostanie zakończony przykładem działania podsystemu odpowiedzialnego za gromadzenie danych.

W opisach elementów *Systemu Wspomagania Podróży* oraz przy opisach wykorzystywanych technologii pozostaliśmy przy nazwach angielskich, aby utrzymać zgodność z nazwami występującymi w bibliografii.

2. Sieć Semantyczna

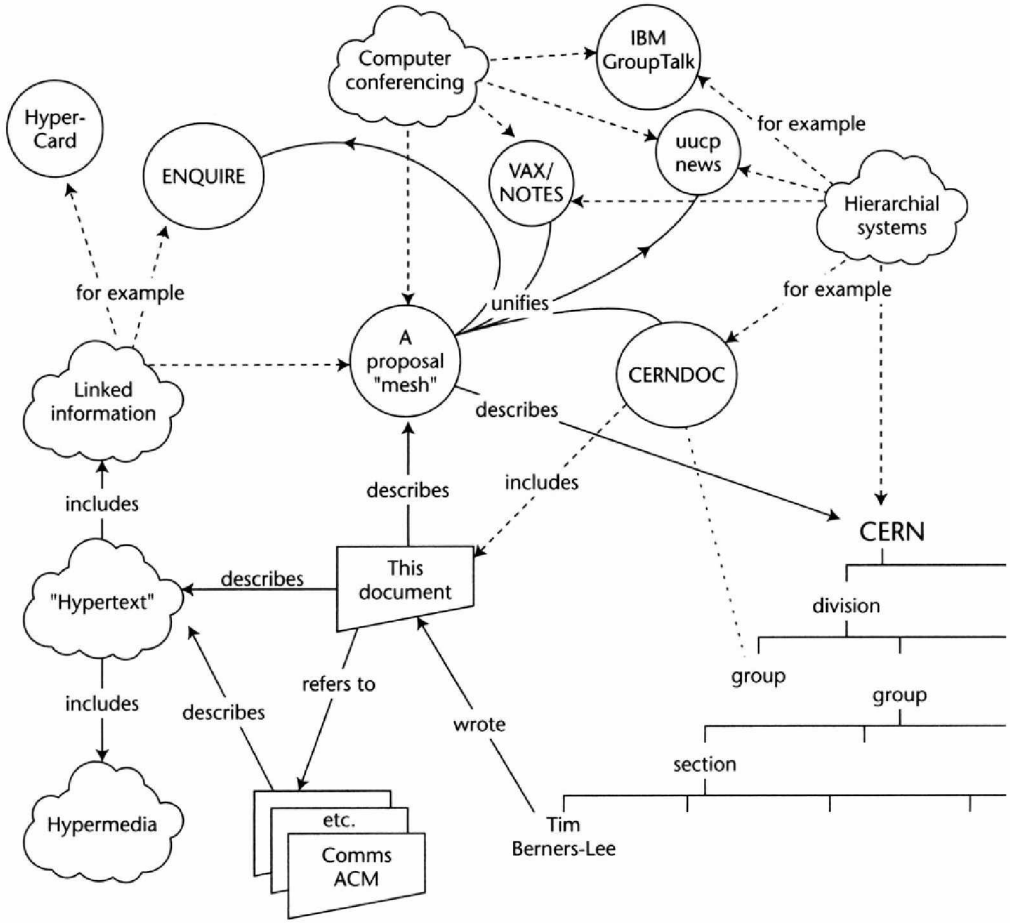
2.1 Definicja

Jaka jest różnica pomiędzy zwykłą *Siecią* (*Web*) a *Siecią Semantyczną* [*Semantic Web*]. Zwykła *sieć* złożona jest z olbrzymiej liczby węzłów, które powiązane są różnego typu relacjami. My chcielibyśmy umieć w jasny i przejrzysty sposób opisać te relacje oraz potrafić łatwo nimi zarządzać. Strony Internetowe, które teraz tworzymy przeznaczone są dla ludzi, my potrzebujemy natomiast stron, które są rozumiane również przez maszyny. To właśnie jest idea *Sieci Semantycznej* (*SS*).

Jako pewna wizja, Sieć Semantyczna jest kolekcją dokumentów XML [XML], semi-strukturalnych baz danych oraz milionów obiektów, które s a odpowiednio opisane. Co więcej, na podstawie informacji z *Internetu Semantycznego*, maszyny, agenci programowi, serwisy internetowe wykonują akcje i podejmują decyzje.

Idea SS została zapoczątkowana prze Tim'a Berners'a Lee, twórcę sieci Web. Jego zdaniem rozwój sieci powinien iść równocześnie w dwóch kierunkach. Po pierwsze chcielibyśmy, aby elementy sieci (np.: serwisy internetowe) współpracowały ze sobą w większym stopniu. Z drugiej zaś strony bardzo ważne jest, aby informacje prezentowane w sieci, były rozumiane przez maszyny. Na rysunku 1 przedstawiony jest oryginalny diagram ukazujący wizję *SS* według Tim'a Berners-Lee.

Początkowa wizja sieci według Tim'a Berners-Lee ewoluowała ze zwykłego pobierania stron HTML [HTML] z serwerów internetowych. Na wyżej przedstawionym rysunku łatwo zauważyć relacje między obiektami, m.in. „includes" (zawiera), „describes" (opisuje) czy „wrote" (napisał). Niestety zapis tego typu relacji praktycznie nie istnieje obecnie w sieci Web. Technologie pozwalające na tworzenie, utrzymywanie, zarządzanie takimi relacjami to m.in. Resource Description Framework (RDF) czy Ontologie opisane w dalszej części tego rozdziału.



Rys. 1. Wizja Sieci Semantycznej według Tim'a Berners-Lee. Prawa autorskie Tim Berners Lee.

Ważnym wnioskiem z powyższego diagramu jest, iż do oryginalnej wizji sieci Web zostały dołożone metadane (dane o danych) dla informacji prezentowanych w zwykłej sieci Web. Te dodatkowe metadane potrzebne są, aby informacje prezentowane w sieci były rozumiane przez maszyny.

Powyżej chciałem przedstawić podstawową różnicę między *zwykłą siecią Web* a *Siecią Semantyczną*. O zwykłej sieci czasami mówi się, że jest systemem, gdzie ludzie sami czytają strony internetowe, podczas gdy *Sieć Semantyczna* jest systemem, gdzie to maszyny czytają, rozumieją i przetwarzają informacje prezentowane w sieci. Ciekawy artykuł na temat *SS* można przeczytać w [Berners-Lee, 2001].

Cały czas zbliżamy się do stron internetowych rozumianych przez maszyny, ale jeszcze długa droga aby to osiągnąć.

2.2 Potrzeba Sieci Semantycznej

Sieć Semantyczna jest nie tylko do użytku *World Wide Web* [*WWW*]. *SS* reprezentowana jest przez zbiór technologii, które pomogą rozwiązać kluczowe problemy związane z obecną architekturą systemów IT. Poniżej przedstawiam dwa przykłady.

Przeładowanie informacji (Information overload)
Przeładowanie informacji jest jednym z poważnych problemów, które potrzebują szybkiego rozwiązania. Problem ten staje się coraz większy razem ze zwiększającym się dostępem do Internetu. Niestety nasza przewaga tworzenia informacji nad ponownym wykorzystaniem powstałej wcześniej wiedzy pozostawia ten problem nierozwiązanym.

Słaba agregacja informacji (Poor Content Aggregation)
Zarządzanie informacjami pochodzącymi z różnych źródeł jest problemem pojawiającym się w wielu obszarach, za przykład niech posłużą: agregacja portali (portal aggregation) oraz „przeglądanie zawartości" (content mining). Niestety większość technik mających na celu rozwiązanie tych problemów to „zeskrobywanie" danych ze strony WWW (Screen Scraping). Dużym minusem tej metody jest, iż „zeskrobuje" ona informacje zapisane w języku HTML, który opisuje format (czcionka, rozmiar, itp.), ale nie prezentuje znaczenia dokumentu. Programista, który „zeskrobuje" informacje z danej strony musi wiedzieć, iż interesujące go dane znajdują się w pewnej jej części. Problem szybko pojawia się przy zmianie wyglądu strony. Metoda ta wymaga ciągłego zarządzania.

2.3 Technologie Sieci Semantycznej

Pomimo tego, iż koncepcja Sieci Semantycznej wciąż ewoluuje, wiele zostało już zrobione: RDF – język służący do opisu zasobów, ontologie czy agenci programowi.

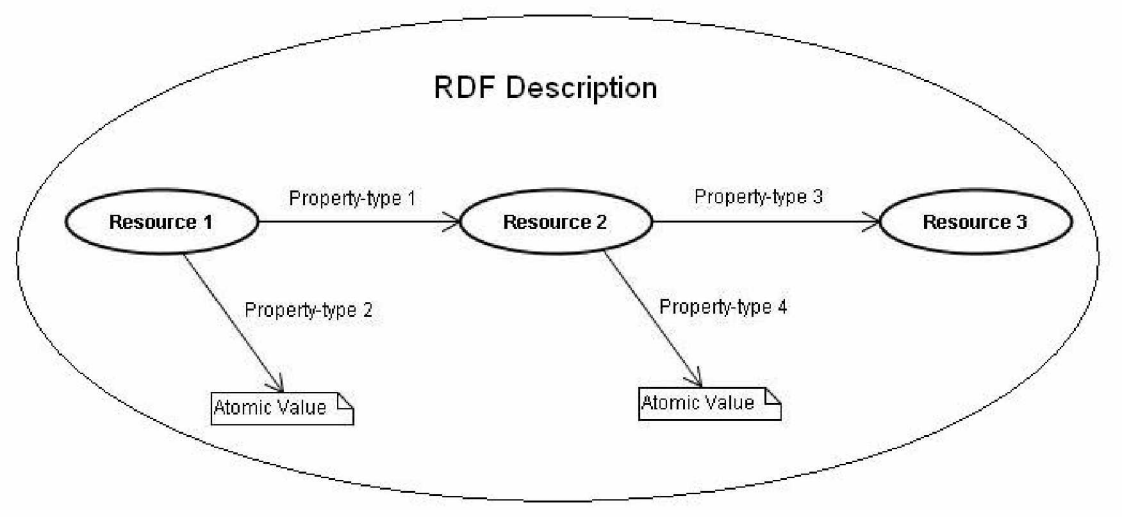
2.3.1 RDF – Resource Description Framework

Zgodnie ze organizacją *W3C* (*World Wide Web Consortium*), *Resoucre Description Framework* [*RDF*] jest językiem zaprojektowanym, aby wspomóc Sieć Semantyczną w podobny sposób jak HTML pomógł w zapoczątkowaniu oryginalnej Sieci. Interesujący tutorial RDF znajduje się na stronie: www.w3.org/TR/REC-rdf-syntax/.

Możliwości języka XML w dostarczanie dokumentów rozumianych przez maszyny są ograniczone, z pomocą przychodzi RDF, który to dostarcza znaczne większe wsparcie dla przeszukiwania czy katalogowania. RDF jest jedną z technologii umożliwiających opis zasobów czy metadanych (danych o danych). RDF dostarcza wspólne struktury, dzięki czemu może być on wykorzystywany dla wymiany danych XML. W szcególności, zgodnie ze stwierdzeniem zawartym w wyżej wymienionym tutorialu, RDF jest uzupełnieniem XML. XML dostarcza składnie oraz definiuje notację, natomiast RDF dodaje semantyczne znaczenie w ustandaryzowany sposób.

Przyjrzyjmy się bliżej modelowi danych języka RDF. Język RDF dostarcza model służący do opisywania danych, w którym zasoby (*resources*) posiadają właściwości (*properties*). Zasobem jest dowolny obiekt (*object*), który jest identyfikowany poprzez Unikalny Identyfikator Zasobu (*Uniform Resource Identifier* (*URI*)) [URI1], [URI2]

Właściwości powiązane z zasobami identyfikowane są poprzez typy właściwości (*property-types*), każda właściwość ma odpowiadającą jej wartość. Wartością (*value*) może być stała wartość (*literal*) (taki jak ciąg znaków czy liczba) bądź inny zasób, który z kolei może mieć swoje właściwości. Kolekcja właściwości, które odnoszą się do jednego zasobu nazywa się opisem (*description*). Poniższy graf przedstawia podstawowy opis RDF (*RDF description*).



Rys. 2. Opis RDF.

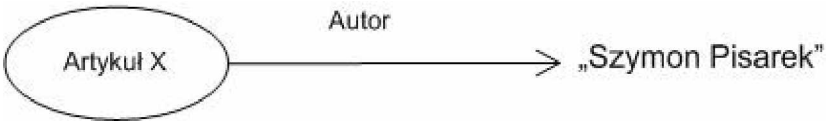
Użycie języka RDF do opisu zasobu najprościej zilustrować na konkretnym przykładzie. Weźmy pod uwagę następujące dwa zdania:

1. Autorem artykułu X jest Szymon Pisarek.
2. Szymon Pisarek jest autorem artykułu X.

Dla każdego człowieka te dwa zdania znaczą dokładnie to samo (tj. Szymon Pisarek jest autorem artykułu X). Jednakże dla maszyny są to kompletnie różne dwa ciągi znaków. Używając trójki RDF: zasobu, właściwości i odpowiadającej jej wartości, język RDF stara się dostarczyć jednoznaczną metodę przedstawiania informacji w rozumiany przez maszyny format.

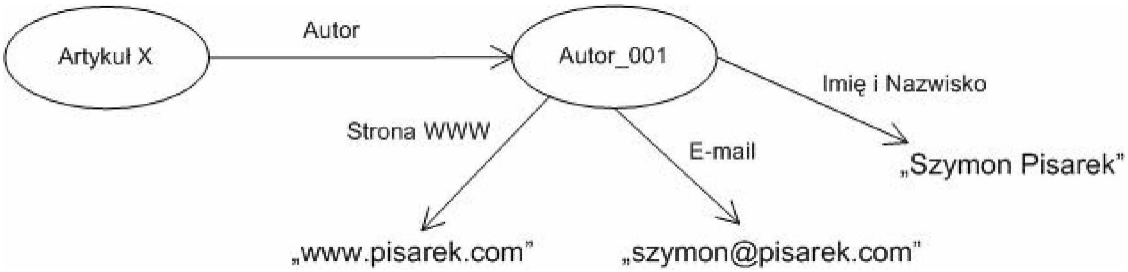
RDF dostarcza mechanizmu łączącego właściwości z zasobami, wiec zanim można powiedzieć coś o Artykule X, trzeba zadeklarować zasób reprezentujący Artykuł X. Dzieje się tak, ponieważ zdanie „Autorem artykułu X jest Szymon Pisarek” ma jeden zasób *Artykuł X*, właściwością jest *autor* i odpowiadającą jej wartością jest *Szymon Pisarek*. Aby rozróżnić elementy modelu danych, specyfikacja definiująca Model i Syntaktykę języka RDF [RDF Model and Syntax] reprezentuje relacje pomiędzy zasobami, właściwościami oraz wartościami właściwości przy pomocy skierowanego grafu. Zasoby reprezentowane są poprzez węzły, właściwości reprezentowane są poprzez skierowane, nazwane krawędzie natomiast wartości właściwości takie jak ciągi znaków

zawarte są w cudzysłowy. Model danych przedstawiający podane wyżej zdanie znajduje się na poniższym rysunku.



Rys. 3. Graf RDF.

Jeśli dodatkowe informacje na temat autora byłyby potrzebne, np.: jego adres e-mail i strona internetowa, musielibyśmy rozszerzyć nasz model. Załóżmy, że w tym przypadku informacje opisujące *Szymona Pisarka* są pożądane. Należy zatem stworzyć zasób (jednoznacznie identyfikowany) reprezentujący *Szymona Pisarka*. Graf RDF przedstawiający ten opis przedstawiony jest poniżej.



Rys. 4. Graf RDF.

W przykładzie tym ciąg znaków „Szymon Pisarek" został zamieniony poprzez zasób identyfikowany przez *Autor_001* razem z właściwościami: *Imię i Nazwisko*, *E-mail* oraz *Strona WWW*. Używanie URI dla identyfikowania zasobów pozwala na jednoznaczne określenie wartości właściwości. Proszę zauważyć, iż *Szymon Pisarek* może być wartością dla wielu różnych właściwości. *Szymon Pisarek* może być nie tylko autorem *Artykułu X*, może być on również programistą pracującym w *Firmie Y*. Jednoznaczne definiowanie zasobów umożliwia ponowne użycie informacji.

2.3.2 Schemat RDF (RDF Schema)

Schematy RDF (RDFS) [RDF Schema] używane są do tworzenia słowników. Schematy RDF definiują możliwe właściwości w danym opisie RDF (*RDF Description*), jak również wiele charakterystyk oraz restrykcji dotyczących wartości właściwości. Schematy RDF identyfikowane są przy pomocy XML'owych przestrzeni nazw (*XML namespaces*) [XML Namespace].

2.3.3 Ontologie (Ontology)

Ontologia jest terminem często pojawiającym się podczas rozmowy na temat Sieci Semantycznej. Czym zatem jest Ontologia?

Ontologia jest to dział filozofii, który zastanawia się na odpowiedzią na zasadnicze pytania: (a) co istnieje? oraz (b) jeśli to, co istnieje, można rozdzielić na części, to czym są te składniki i jakie są relacje między nimi? Jednakże w ujęciu informatycznym ontologia jest czymś trochę innym.

Ontologia opisuje encje razem z relacjami pomiędzy tymi encjami. Ontologie mogą opisywać strony internetowe, samochody, ludzi, zwierzęta jak również relacje między ludźmi, przyjęcia urodzinowe oraz praktycznie wszystko, co możemy sobie wyobrazić.

Język XML nie jest wystarczający, aby dobrze reprezentować Ontologie. Językiem służącym do przedstawiania ontologii może za to być RDF, jednakże nawet bardziej przystosowany do Ontologii jest *Ontology Web Language* [OWL]. OWL i RDF są bardzo podobne, jednakże OWL jest mocniejszym językiem, który może być łatwiej zrozumiany przez maszyny niż RDF. OWL posiada większy słownik i ma mocniejszą składnie. Język OWL został zaprojektowany, aby stworzyć wspólny sposób przetwarzania informacji – nie wyświetlania (jak robi to HTML). Dokumenty tworzone w języku OWL przeznaczone są dla programów komputerowych, nie dla ludzi. OWL ma trzy podjęzyki:

- OWL Lite
- OWL DL (zawiera OWL Lite)

- OWL Full (zawiera OWL DL)

Jednakże, w praktyce języki te są równie popularne. Równocześnie często kwestionowana jest skomplikowana struktura języka OWL (porównywana do takich języków programowania jak Algol czy Ada, które nigdy nie upowszechniły się min. właśnie z powodu nadmiernego skomplikowania). Tak więc, ze względu na jego prostotę i większą dostępność narzędzi, zadecydowaliśmy się posłużyć się w niniejszej pracy językiem RDF.

Przyjrzyjmy się zatem przykładowi Ontologii, niech będzie to *Ontologia Lotniska* (*Airport Ontology*) stworzona przez DARPA DAML Program [Airport Ontology].
Klasa *Airport* ma następujące właściwości:

- Elevation - elewacja
- iataCode – kod nadany przez *International Air Transport Association* ,[IATA]
- icaoCode – kod nadany przez *International Civil Aviation Authority* ,[ICAO]
- latitude – szerokość geograficzna
- location - lokazlizacja
- longitude – długość geograficzna
- name – nazwa lotniska

Poniżej przedstawiony jest zapis ontologii Airport Ontology w języku RDF.

```
<rdf:RDF xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.daml.org/2001/10/html/airport-ont">

  <owl:Ontology rdf:about="">
    <owl:versionInfo>$Id: airport-ont.daml,v 1.1 2002/03/14 06:24:16
        mdean Exp $</owl:versionInfo>
    <rdfs:comment>Airport</rdfs:comment>
  </owl:Ontology>
  <rdfs:Class rdf:ID="Airport">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#name"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:onProperty rdf:resource="#iataCode"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#icaoCode"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#location"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#latitude"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#longitude"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#elevation"/>
        <owl:allValuesFrom
               rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </rdfs:Class>

  <owl:DatatypeProperty rdf:ID="elevation"/>
  <owl:DatatypeProperty rdf:ID="iataCode"/>
  <owl:DatatypeProperty rdf:ID="icaoCode"/>
  <owl:DatatypeProperty rdf:ID="latitude"/>
  <owl:DatatypeProperty rdf:ID="location"/>
  <owl:DatatypeProperty rdf:ID="longitude"/>
  <owl:DatatypeProperty rdf:ID="name"/>

</rdf:RDF>
```

2.3.4 Agenci Programowi (Software Agents)

Czym jest *Agent Programowy*? Na pytanie to nie ma jednoznacznej odpowiedzi, istnieje wiele definicji *Agenta Programowego* [Paprzycki 2003]. Jednakże możemy przyjąć, iż *Agent* jest samodzielną jednostką programową [software artifact] umieszczoną w pewnym środowisku, zdolną do komunikowania się z innymi *Agentami*, przemieszczania się oraz podejmowania autonomicznych decyzji, w celu wykonania swojego zadania ustalonego podczas projektowania bądź w trakcie działania agenta. Chcielibyśmy również, aby agent umiał uczyć się ze swoich doświadczeń. Ważne też jest, aby agenci, którzy nie są sami w swoim środowisku – są tam również inni agenci – byli w stanie komunikować się i współpracować ze swoimi „sąsiadami". Zbiór agentów komunikujących się oraz współpracujących ze sobą tworzy *System Wieloagentowy* (*Multi-Agent System*). Cały czas dążymy do wyższego poziomu abstrakcji programowania, kiedyś to obiekt (object) był najwyższym poziomem abstrakcji, teraz jest to *Agent*.

Prace nad *Agentami Programowymi* zostały zapoczątkowane w celu pogłębienia wiedzy na temat modeli komputerowych zajmujących się zagadnieniami rozproszonej sztucznej inteligencji. Nowa fala zainteresowania *Agentami* powstała z powodu bardziej praktycznych zastosowań, takich jak: uproszczenie wykonywania zadań w systemach rozproszonych, lepsze zarządzanie informacją czy w końcu poprawienie interakcji użytkownika z programem komputerowym.

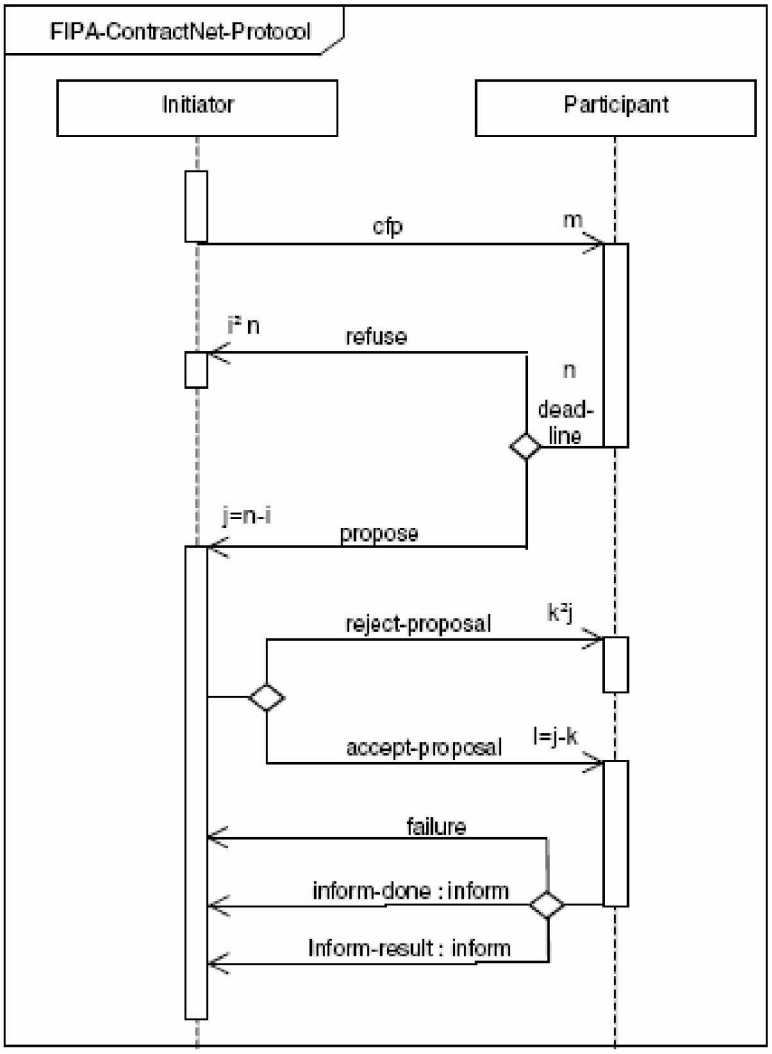
Wizja *Sieci Semantycznej* wymaga, aby *Agenci* umieli komunikować się oraz współpracować ze sobą. Aby było to możliwe potrzeba organizacji definiującej standardy pozwalające budować systemy oparte na *Agentach Programowych*. Organizacją taką jest powstała w 1996 roku w Szwajcarii: *Foundation for Inteligent Physical Agents* [FIPA]. Promuje ona rozwiązania agentowe a także współprace systemów agendowych z innymi technologiami. *FIPA* definiuje standardy obejmujące *Zarządzanie Agentami* (*Agent management*), *Transport Wiadomości* (*Agent Message Transport*) oraz *Komunikacje Agentów* (*Agent Communikaction*).

Specyfikacja *Zarządzanie Agentami* opisuje kontrolę oraz zarządzania agentami na konkretnych platformach (*Agents Platforms*) a także pomiędzy tymi platformami.

Specyfikacja *Transport Wiadomości* określa sposób transportu oraz reprezentacje wiadomości pomiędzy różnymi protokołami transportu danych w sieci, zarówno w środowiskach standardowych jak i bezprzewodowych.

Specyfikacja *Komunikacja Agentów* zajmuje się między innymi: językiem służącym do komunikacji: *Agent Communication Language* [ACL], wiadomościami zapisanymi w tym języku oraz protokołami wymiany wiadomości. Przyjrzyjmy się bliżej popularnemu protokółowi wymiany wiadomości w Systemach Wieloagentowych, *Contract Net Protocol* [Contract Net Protocol].

Contract Net Protocol (*CNP*)
CNP jest drobną modyfikacją oryginalnego protokółu stworzonego przez Smith'a oraz Davis'a, poprzez dodanie wiadomości typu: odrzucenie (*rejection*) oraz potwierdzenie (*confirmation*). W protokóle tym jeden z agentów odgrywa rolę inicjatora (*initiator*), który prosi jednego lub więcej agentów (*participants*), aby wykonali pewne zadanie. Następnie optymalizuje on funkcję charakterystyczną dla tego zadania. Charakterystyka ta bardzo często zawiera cenę wykonania zadania (oczywiście określoną w sposób specyficzny dla problemu). Kolejnym krokiem jest wybranie agenta do wykonania zadania. Rysunek 5 przedstawia protokół *CNP* wyrażony przy użyciu języka UML.



Rys. 5. Protokół CNP. Prawa autorskie FIPA.

Przeanalizujmy powyższy diagram. Inicjator wysyła propozycję zadania do wykonania (*call for proposal*) do *m* agentów. Agenci, którzy otrzymali tą propozycję są potencjalnymi wykonawcami i mogą przesłać *n* (*n* <= *m*) odpowiedzi. Mogą być to propozycje wykonania zadania (*propose*), bądź odmowy jego wykonania (*refuse*). Gdy tylko minie maksymalny czas oczekiwanie na odpowiedź (*deadline*), inicjator przegląda propozycje i wybiera agentów do wykonania zadania (optymalizując funkcję charakterystyczną zadania). Następnie inicjator przesyła akceptację na wykonanie zadania (*accept-proposal*) do wybranych agentów, natomiast pozostałym wysyła odrzucenie propozycji (*reject-proposal*). Gdy tylko agent, będący potencjalnym

wykonawcą, otrzyma *accept-proposal*, rozpoczyna wykonanie zadania. Po zakończeniu zadania, wykonawca przesyła wiadomość informującą o statusie, może być to: *inform-done* – zadanie wykonane poprawnie, *inform-result* – zadanie wykonanie poprawnie wraz z informacją mówiącą o rezultacie zadania, czy w końcu *failure* – w przypadku, gdy wystąpiły błędy podczas wykonywania zadania.

2.3.5 JADE – Fundament dla tworzenia Systemów Wieloagentowych

Szerokie wykorzystanie agentów w praktyce będzie możliwe tylko, gdy będą istniały powszechnie dostępne narzędzia umożliwiające w prosty sposób tworzenie oraz zarządzanie *Agentami Programowymi*. *Java Agent DEvelopment Framework* [*JADE*] jest fundamentem do tworzenia Systemów Agentowych zgodnych ze specyfikacjami *FIPA*, całkowicie stworzonym w języku Java [Java]. Więcej informacji na temat JADE można znaleźć na stronie internetowej http://jade.cselt.it.

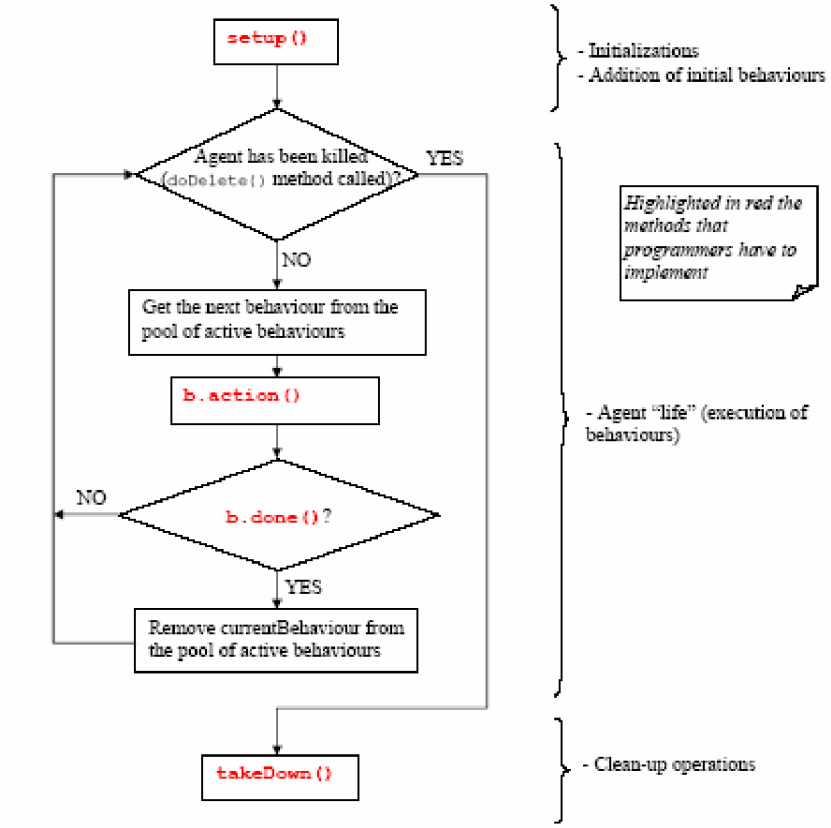
Tworzenie agentów w środowisku *JADE* jest bardzo proste, wystarczy stworzyć tylko klasę dziedziczącą z klasy Agent (*jade.core.Agent*). Z tego też powodu, z punktu widzenia programisty agentem jest dowolna instancja klasy rozszerzającej klasę bazową *jade.core.Agent*. Klasa pochodna klasy *Agent* musi implementować następujące metody:

- *setup*() - miejsce inicjalizacji, tutaj dodawane są *zachowania*
- *takeDown*() – miejsce na zwolnienie przetrzymywanych zasobów (podobne do destruktora z języka C++).

Każde zadanie, które agent wykonuje obsługiwane jest przez odpowiednie *zachowanie* (*behaviour*). *Zachowanie* jest obiektem dowolnej klasy, która dziedziczy z klasy *jade.core.behaviours.Behaviour*. Wewnętrzny mechanizm *JADE* odpowiedzialny jest za zarządzanie *zachowaniami* – w sposób niewidoczny dla programisty. W celu zmuszenia agenta, aby wykonał pewne zadanie wystarczy dodać odpowiadające mu *zachowanie* do puli *zachowań Agenta* (przy wykorzystaniu metody *addBehaviour*()). *Zachowanie* może być dodane w dowolnym momencie: przy starcie *Agenta* (w metodzie *setup*()) lub z poziomu innego *zachowania*.

Każda klasa dziedzicząca z klasy *Behaviour* musi implementować metodę *action*(), która w rzeczywistości definiuje operacje wykonywane podczas wykonania *zachowania* oraz metodę *done*(), która informuje, czy *zachowanie* to zakończyło pracę i może być usunięte z puli zachowań *Agenta*.

Ścieżka życia *Agenta* przedstawiona jest na poniższym rysunku.



Rys. 6. Ścieżka życia Agenta. Rysunek pochodzi z „JADE PROGRAMMING FOR BEGINNERS".

2.3.6 JENA – Ontologiczna Baza Danych

Jena jest fundamentem do tworzenia aplikacji przeznaczonych dla Sieci Semantycznej. Jest to produkt *open source*, który został zapoczątkowany przez firmę Hewlett-Packard.

Środowisko to dostarcza:

- interfejsy do pracy z językiem *RDF (*klasy odpowiedzialne za czytanie i pisanie w formatach: *RDF/XML*, *N3* oraz *N-Triples*)
- interfejsy do pracy z językiem *OWL*
- możliwość przechowywania danych w pamięci oraz w relacyjnych bazach danych
- język *RDQL* – język zapytań dla danych zapisanych w języku *RDF* [RDQL]

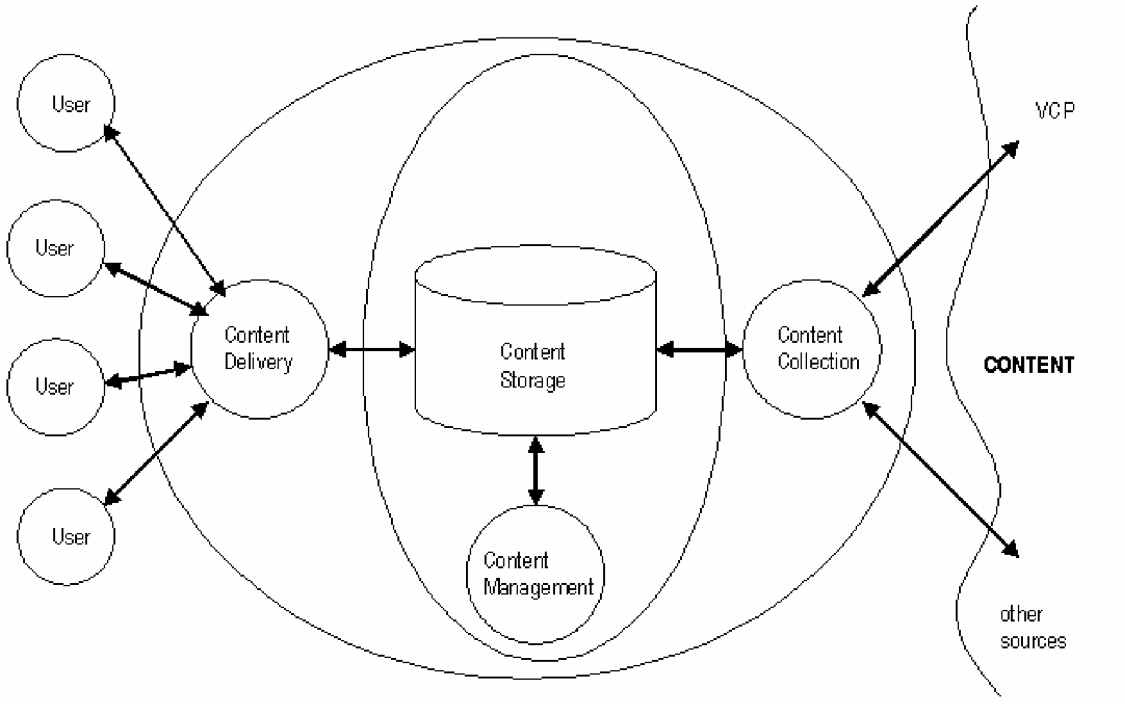
Więcej informacji można znaleźć w [JENA].

3. System Wspomagania Podróży

3.1 Wprowadzenie

Po przedstawieniu idea *Sieci Semantycznej* (*SS*), najwyższa pora na przedstawienie *Systemu Wspomagania Podróży* (*Travel Support Project, TSP*), systemu, który stara się w pełni wykorzystać możliwości *SS*. W rozdziale tym przedstawiona jest główna część systemu, główny nacisk położony jest na te elementy systemu, które są już wykonane lub zostaną wykonane wkrótce. W związku z tym, w opisie nie pojawią się wszyscy *Agenci* prezentowani w [Angryk, 2002], [Galant, 2002a] i [Gordon, 2005a].

3.2 Elementy Systemu Wspomagania Podróży



Rys. 7. Schemat Systemu Wspomagania Podróży.

Zarys architektury systemu przedstawiony został na Rysunku 7 (jego pełny opis znajduje się w [Gordon, 2005a]). W dalszej części rozdziału zaprezentowane są główne elementy systemu.

User (*U*) Dostęp do systemu możliwy jest poprzez różnego rodzaju „klientów internetowych", od przeglądarek, poprzez palmtopy, telefony obsługujące protokół WAP, skończywszy na agentach innych systemów.

Content Delivery Subsystem (*CDS*) odgrywa bardzo ważną rolę. Jest on odpowiedzialny po pierwsze za format i syntaktykę komunikatów wymienianych pomiędzy użytkownikami a samym systemem. Agenci odpowiedzialni za tą część a także szczegóły, w jaki sposób jest to realizowane można znaleźć w [Galant, 2002b], [Kaczmarek, 2005]. Z drugiej zaś strony, *CDS* odpowiedzialny jest za semantyczne znaczenia interakcji typu: użytkownik-system. Dwaj agenci odgrywają tutaj szczególne role. Pierwszy to *Personalization Infrastructure Agent* (*PIA*), który składa się z prostych „podagentów RDF" (każdy z nich jest klasą wewnątrz *PIA*). „Podagenci RDF" rozszerzają zbiór wybranych elementów (związanych z podróżą) jako odpowiedź dla oryginalnego zapytania w celu stworzenia *Maximum Response Set* (*MRS*). *MRS* jest przekazywany dalej to drugiego kluczowego agenta, *Agenta Personalnego* (*Personal Agent*, *PA*), który filtruje i pozycjonuje wynikowy zbiór. *PA* wykorzystuje *profil użytkownika* (*user profile*) w celu filtracji oraz stworzenia hierarchicznej struktury informacji otrzymanych od *PIA* (jako *MRS*). W celu stworzenie lepszego profilu użytkownika, *PA* zapamiętuje reakcje użytkownika na otrzymanie propozycji.

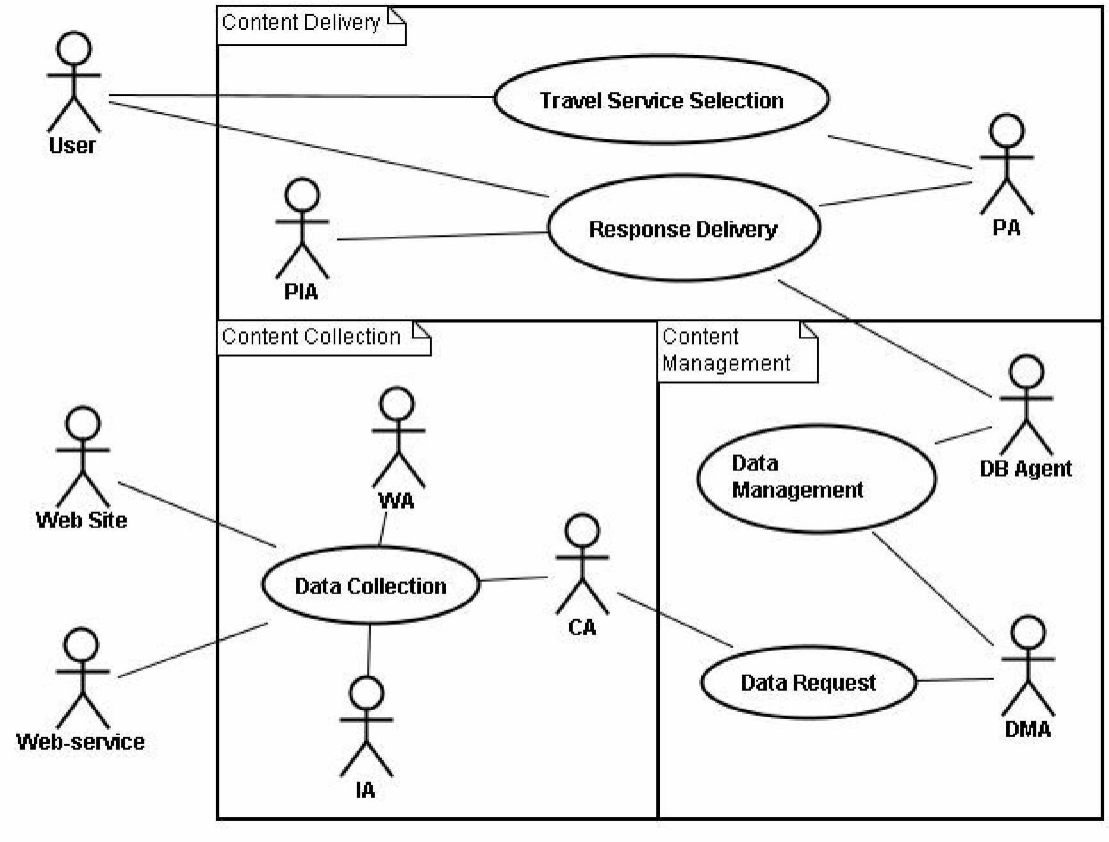
Content Management Subsystem (*CMS*) odpowiada za całość funkcji związanych z zarządzaniem danymi zgromadzonymi w centralnym repozytorium. Agenci tego podsystemu (typu *Data Management Agent* (*DMA*)) odpowiedzialni są m.in. za nadzór, aby dane w systemie były aktualne. Załóżmy, iż co piątek zmienia się repertuar pewnego kina, *CMS* o tym wie i to on informuje w odpowiednim czasie *CCS*, iż odpowiednie dane powinny zostać zaktualizowane. Innym zadaniem *CMS* jest uzupełnianie niekompletnych danych. Załóżmy, że posiadamy dane na temat hotelu: jego nazwę i adres, ale nie mamy numeru telefonu. W tej sytuacji *CMS* informuje *CCS*, że należy znaleźć brakujące informacje. Więcej danych na temat agentów *DMA* znajduje się w [Angryk 2002], [Gordon, 2005a].

Content Collection Subsystem (*CCS*) stanowi przedmiot niniejszej pracy i odpowiedzialny jest za gromadzenie danych (pochodzących z różnych źródeł) oraz przekształcanie ich do postaci wymaganej przez system (opisanej językiem RDF i odpowiadających zakładanej przez system ontologii podróży [Gordon, 2005b], [Gawinecki, 2005b]). *CCS* wykorzystuje zbiór agentów typu *Wrapper Agent* (*WA*), którzy odpowiedzialni są za „zeskrobywanie" informacji prezentowanych na znanych im stronach WWW. Bieżący status Sieci Semantycznej zmusza nas do ręcznego tworzenia każdego agenta *WA*. Warto jednak zauważyć, iż zakłada się, że w przyszłości dane w postaci trójek RDF będą standardem sieci Web. Jedyna operacja wymagana, aby przystosować system do nowej sytuacji będzie zamianą bieżących, statycznych agentów *WA* na ich „ontologicznych" odpowiedników bez najmniejszej zmiany pozostałej części systemu. To właśnie jest jeden z plusów tworzenia Systemów Wieloagentowych [(Jennings, 2001)].

Content Storage (*CS*) zawiera dane składające się z trójek języka RDF odpowiadających przyjętej w systemie ontologii. *CS* działa w oparciu o system JENA, który jest, opisaną powyżej, opracowaną przez HP bazą danych dla danych ontologicznych opisanych np. w języku RDF.

Verified Content Providers (*VCP*) oraz *Other Sources* (*OS*) to skutek podziału Internetu na źródła, którym możemy zaufać, oraz „całą resztę". *VCP* zawierają informacje, które są aktualne i spójne, natomiast *OS* to zbiornice danych, co do których takiej pewności nie mamy.

Kolejny rysunek (Rysunek 8) przedstawia system przy użyciu języka UML [UML]. Mamy tutaj do czynienia z diagramem typu przypadek użycia (*Use-case*). Widoczni są tam przedstawieni powyżej agenci wraz z pozostałymi agentami centralnej części systemu; jak również ich podstawowe oddziaływania.



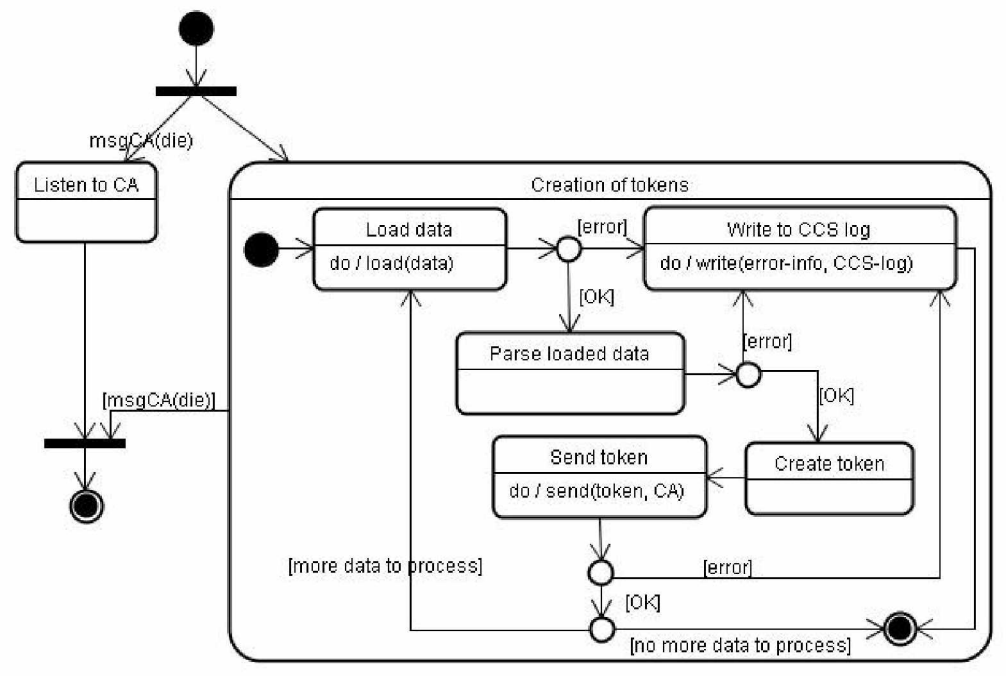
Rys. 8. System Wspomagający Podróżowania. Diagram UML – przypadek użycia.

Przeanalizujmy powyższy diagram. Zacznijmy od tego, że w chwili obecnej postanowiliśmy nie zajmować się źródłami danych należącymi do *Other Sources*. Tak więc na powyższym rysunku przedstawiamy wyłącznie strony WWW (*Web sites*) oraz serwisy internetowe (*Web-services*) należące do *VCP*. Źródła te używane są w funkcji *Gromadzenia Danych* (*Data Collection*), która wykonywana jest przez agentów typu *Wrapper Agent* (*WA*), agentów indeksujących dane *Indexing Agent* (*IA*) oraz jednego lub kilku koordynatorów *Coordinator Agent* (*CA*). Jak doskonale widać *CA* może otrzymywać żądania wyszukania informacji (*Data requests*) od *Data Management Agents* (*DMA*). Żądania danych reprezentują sytuację, gdy znalezione tokeny są potencjalnie nieaktualne (jako część funkcji *Zarządzania Danymi* (*Data Management*)) w wyniku czego nowy token, ze świeżymi danymi, powinien być dostarczony przy wykorzystaniu odpowiedniego agenta *WA*. Tylko dwaj agenci: *DMA* oraz *DB Agent* (*DBA*) mają bezpośredni dostęp do bazy danych *JENA*. W podsystemie *Content Delivery*

Subsystem mamy dwie funkcje. *Travel Service Selection* powiązana jest z użytkownikiem (*User*) korzystającym z systemu (przepływ informacji od *User*'a do centralnego repozytorium). Druga funkcja: *Response Delivery* reprezentuje operacje wykonywane pomiędzy czasem, gdy początkowa odpowiedź na zapytanie użytkownika jest pobierana z repozytorium JENA a momentem, gdy ostateczna, spersonalizowana odpowiedź dostarczana jest do użytkownika (przepływ informacji od centralnego repozytorium do *User*'a).

3.3 Agenci Systemu Wspomagania Podróży

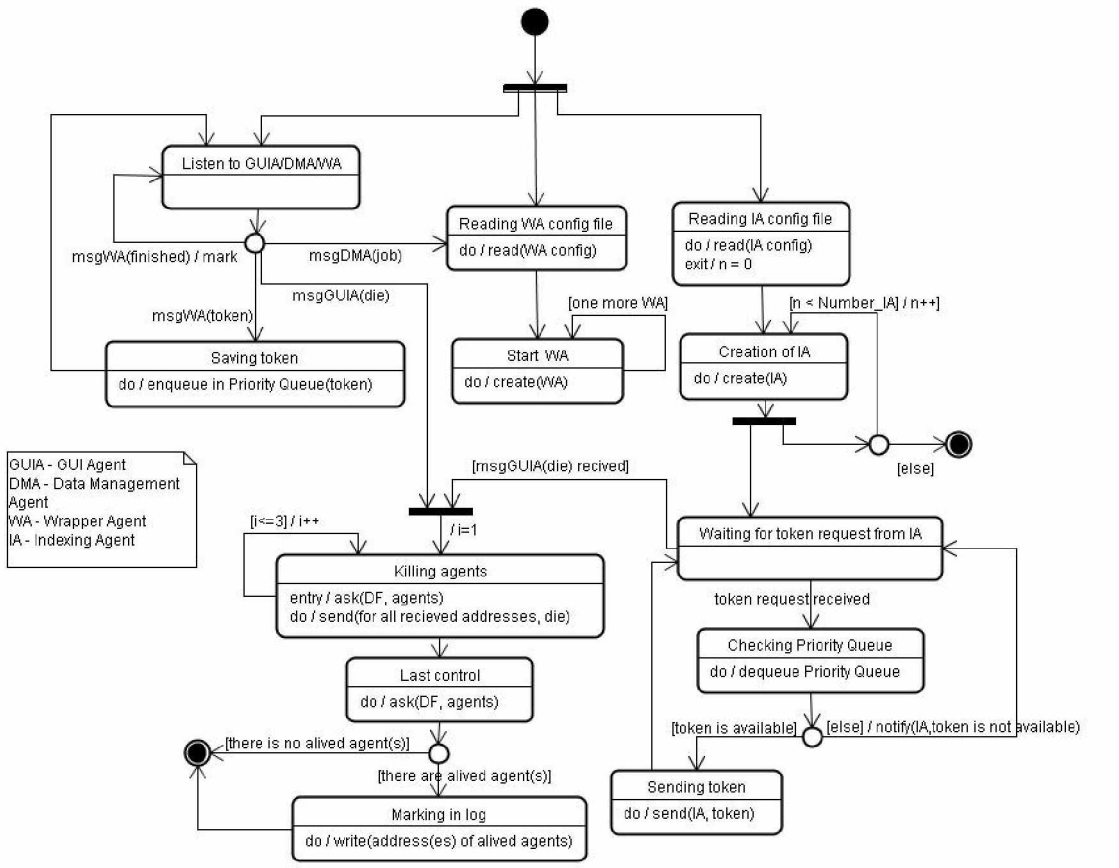
Wrapper Agent (*WA*) odwiedza różne strony WWW (HTML, XHTML) i przekształca zapisane tam informacje z formatu XML, HTML czy XHTML do trójek RDF opisujących obiekty powiązane z podróżowaniem przy wykorzystaniu odpowiedniej ontologii [Gawiniecki, 2005a], [Gawiniecki, 2005b]. *WA* tworzony jest przez koordynatora *CA* na podstawie informacji z *pliku konfiguracyjnego*. Plik konfiguracyjny może być stworzony przez administratora systemu i dostarczany do nowo stworzonego *CA* podczas startu systemu lub może być zawarty w wiadomości przesłanej od *DMA*, który to odkrył, iż jeden lub więcej tokenów powinno być uaktualnionych. Do każdego zbudowanego tokenu dołączana jest data stworzenia, określany jest jego priorytet i następnie token jest przesyłany do *CA*. Na zakończenie swojej pracy *WA* wysyła odpowiednią wiadomość do *CA* i ulega „samozniszczeniu". Nowy agent *WA* z taką samą funkcjonalnością tworzony jest przez *CA* za każdym razem, gdy jest to potrzebne. W celu ułatwienia zarządzania agentami, system nasz tworzy instancje agentów *WA* dla każdego „zadania", pomimo tego, iż mogą oni produkować token opisujący ten sam zasób. Dla przykładu weźmy agenta *WA*, który poszukuje informacji o wszystkich hotelach Marriott na świecie (zadanie to było zapisane w pliku konfiguracyjnym), w tym samym czasie inny *WA* może zostać poproszony o znalezienie informacji na temat warszawskiego hotelu Marriott (jako zadanie z wysokim priorytetem zgłoszone przez *DMA*). Rolą agenta indeksującego *Indexing Agent* (*IA*) jest zapewnienie, że token z najświeższymi danymi składowany jest w repozytorium JENA. Diagram stanu (UML) przedstawiający *WA* jest na Rysunku 9.



Rys. 9. Diagram stanu agenta WA

Coordinator Agent (*CA*), jak sama nazwa wskazuje, koordynuje zadaniami podsystemu *Content Collection Subsystem* (*CCS*). Jak tylko *CA* zostanie stworzony, czyta on plik konfiguracyjny dostarczony przez administratora systemu i tworzy odpowiednią liczbę agentów indeksujących *IA*. Tworzy on również odpowiednich agentów *WA* i przechodzi w stan nasłuchu (*listening mode*). Istnieje pięć typów wiadomości, które *WA* może otrzymać: (1) rozkaz zakończenia pracy otrzymany od *GUI* (wydany przez administratora) – powodujący, iż *CA* zabija wszystkich żyjących agentów *WA* oraz *IA* a następnie sam siebie. (2) Wiadomość od *WA* mówiąca, iż dany agent skończył swoją pracę i zaraz ulegnie samozniszczeniu – informacje te są zapamiętywane przez *CA*. (3) Wiadomość od *WA* zawierającą token, opisujący obiekt związany z podróżowaniem – token ten umieszczany jest w kolejce priorytetowej agenta *CA*. (4) Wiadomość od agenta *IA* proszącego o nowy token do zapisania w repozytorium – w rezultacie czego, token o najwyższym priorytecie usuwany jest z kolejki priorytetowej i wysyłany do *IA*. W przypadku, gdy kolejka jest pusta, wiadomość o tym wysyłana jest do agenta indeksującego a ten za chwilę ponownie prosi o token (co widać na Rysunku 11). W

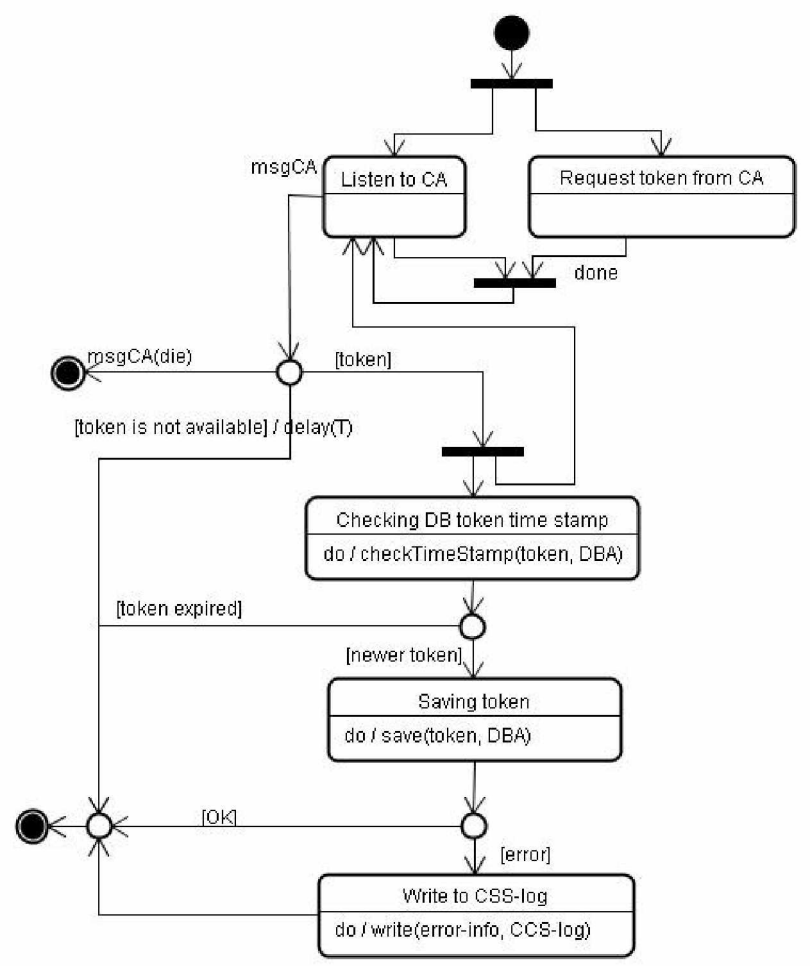
końcu, (5) *DMA* może przesłać wiadomość zawierającą prośbę (w postaci pliku konfiguracyjnego), że pewien token lub zbiór tokenów powinien być uaktualniony. Wiadomość ta powoduje, iż odpowiedni agent *WA* (lub agenci *WA*) jest tworzony. Diagram stanu agenta *Coordinator Agent* przedstawiony jest na Rysunku 10.



Rys. 10. Diagram stanu agenta CA.

Indexing Agent (*IA*) jest odpowiedzialny za umieszczanie tokenów w centralnym repozytorium. W przyszłości agent tego typu będzie odpowiedzialny również za wstępne przetwarzanie tokenów w celu zapewnienia, iż tylko „poprawne” dane będą przechowywane w systemie. Agent ten będzie sprawdzał m.in. spójność tokenów, które mają być zapisane z tokenami, które są już w repozytorium a następnie zaznaczał te, które wymagają rozwiązania konfliktu [Angryk, 2002]. W chwili obecnej *IA* sprawdza tylko czas stworzenia tokena, który ma być zapisany w repozytorium. Możliwe bowiem jest, iż wielu agentów *WA* stworzy token opisujący ten sam zasób, *IA* w pierwszej

kolejności sprawdza czas stworzenia tokenu, który znajduje się w repozytorium i zapisuje token tylko, gdy jest on nowszy. W przypadku, gdy żaden token nie jest dostępny (priorytetowa kolejka tokenów agenta *CA* jest pusta), żądanie tokenu będzie powtórzone po czasie *T*. Diagram stanu agenta *Indexing Agent* przedstawiony jest na Rysunku 11.



Rys. 11. Diagram stanu agenta IA.

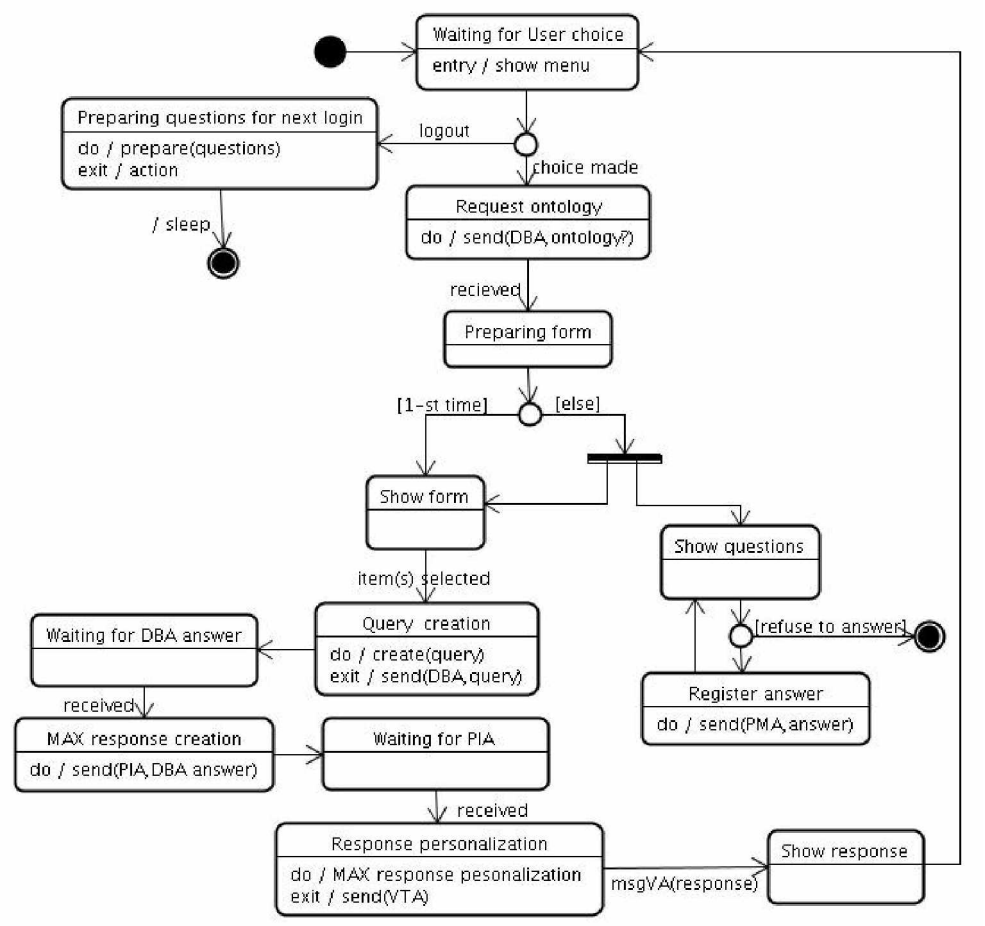
DB Agent (*DBA*) jest pomostem między bazą danych (repozytorium JENA w naszym przypadku) a systemem agentowym. Jest on tworzony zgodnie z podstawowymi zasadami projektowania systemów agentowych. Oddziela on system agentowy od „innych technologii", dzięki temu, zmiana w repozytorium powoduje zmiany tylko w tym agencie, podczas gdy pozostała część systemu pozostaje nienaruszona.

W bieżącej wersji systemu, agent *Data Management Agent* (*DMA*) jest bardzo prosty. Wiele agentów tego typu tworzonych jest podczas startu systemu. Głównym zadaniem *DMA* jest przeglądanie repozytorium w celu znalezienia „przeterminowanych" tokenów i zgłoszenia prośby, aby odpowiednie „świeże" tokeny zostały stworzone przez agentów *WA* i zapisane w repozytorium. Aby to wykonać, agenci *DMA* generują odpowiednie pliki konfiguracyjne, mówiące którzy agenci *WA* powinni zebrać potrzebne informacje a następnie wysyłają je do *CA*. W przyszłości agenci *DMA* odpowiedzialni również będą za kompleksowe zarządzania tokenami przechowywanymi w repozytorium, w celu zapewnienie kompletność, spójności oraz świeżości danych w systemie.

Agent *Personalization Infrastructure Agent* (*PIA*) składa się z kilku "podagentów RDF". Każdy podagent reprezentuje jedną lub kilka prostych zasad typu: „kuchnia włoska jest również kuchnią" czy „komedia romantyczna jest również komedią". Zasady te stosowane są do zbioru trójek RDF zwróconych przez zapytanie na podstawie oryginalnego zapytania użytkownika. Użycie tych zasad powoduje przeglądanie repozytorium i zwykle rozszerza zbiór wynikowy. Podagenci agenta *PIA* przekazują sobie kolejno zbiór wynikowy a ich zadaniem jest ulepszenie zbioru wynikowego zwracanego użytkownikowi. Rezultatem tych operacji jest *Maximal Response Set* (*MRS*), który obsługiwany jest przez agenta *Personal Agent.*

Personal Agent (*PA*) odgrywa dwie role w *Content Delivery Subsystem*. Po pierwsze, jest on centralnym koordynatorem. Przekierowuje on każde zapytanie użytkownika od jednego agenta, do kolejnych, cały czas monitorując postęp tworzenia odpowiedzi. Po drugie, wykorzystuje on *profil użytkownika* (*user profile*) w celu filtrowania i pozycjonowania odpowiedzi wysyłanych zwrotnie do użytkownika. Bardziej szczegółowo, zapytanie użytkownika po wstępnym przetworzeniu i przetransformowaniu do zapytania w języku RDQL [Kaczmarek, 2005] jest wysyłane do *DBA*. Agent *DBA* zwraca *początkową odpowiedź* (*initial response*) składającą się z kolekcji tokenów, które spełniają zapytanie. *PA* przekierowywuje odpowiedź do *PIA* w celu rozszerzenia jej. Jako rezultat operacji agenta *PIA*, tworzony jest *MRS* i wysyłany z powrotem do *PA*. *PA* korzysta z profilu użytkownika w celu: (1) usunięcia nieprzydatnych odpowiedzi ze

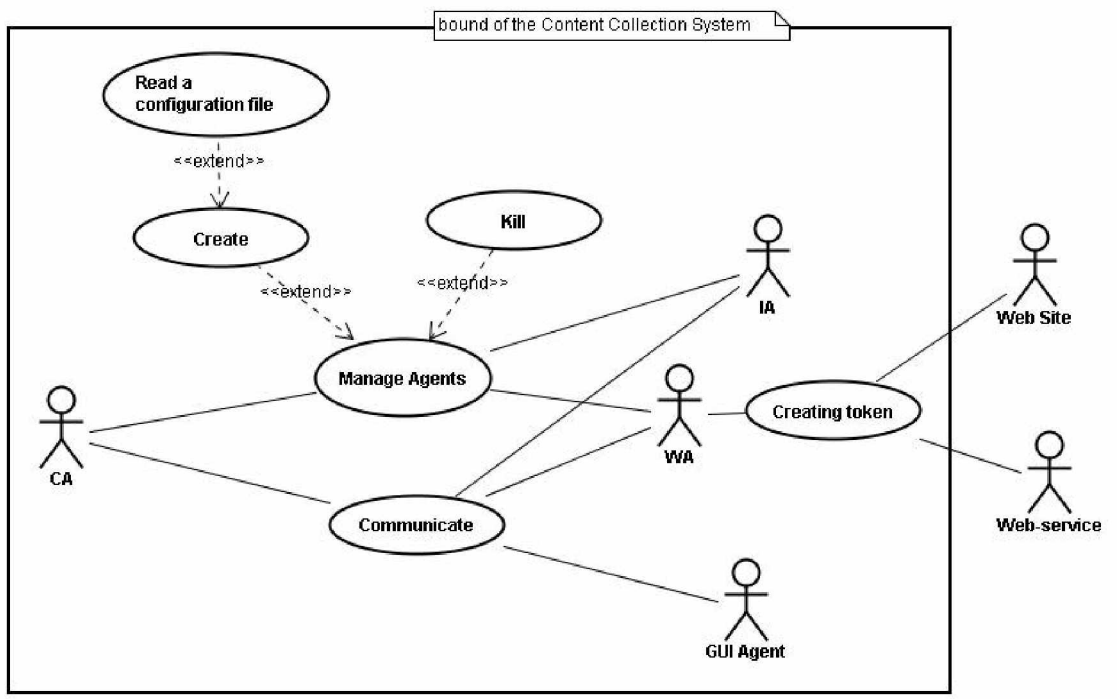
zbioru (np.: o danym użytkowniku wiadomo, iż nie lubi on kuchni chińskiej, zatem wszystkie restauracje serwujące tylko kuchnię chińską powinny zostać odrzucone); (2) pozycjonuje pozostałe propozycje, w ten sposób, aby te które wydają się być najbardziej odpowiednie dla użytkownika były wyświetlane jako pierwsze (np.: jeśli wiadomo, iż użytkownik lubi zatrzymywać się w hotelach Sheraton, propozycje odnośnie hotelów Sheraton będą wyświetlane jako pierwsze). Jeśli założymy, iż oddzielny agent personalny będzie tworzony dla każdego użytkownika systemu, role ich pozostaną takie same. Diagram stanu agenta *Personal Agent* przedstawiony jest na Rysunku 12.



Rys. 12. Diagram stanu agenta PA.

4. Podsystem Odpowiedzialny za Gromadzenie Danych

W rozdziale tym przedstawię dodatkowe, bardziej szczegółowe informacje na temat *Podsystemu Odpowiedzialnego za Gromadzenie Danych* (*Content Collection Subsystem*, *CCS*) [Pisarek 2005]. Poniższy Rysunek 13 przedstawia agentów systemu *CCS*. Agenci systemu zostali już przedstawieni w poprzednich rozdziałach, jednakże rysunek ten zawiera inne spojrzenie na *CCS*. Omówmy teraz jego poszczególne funkcje.

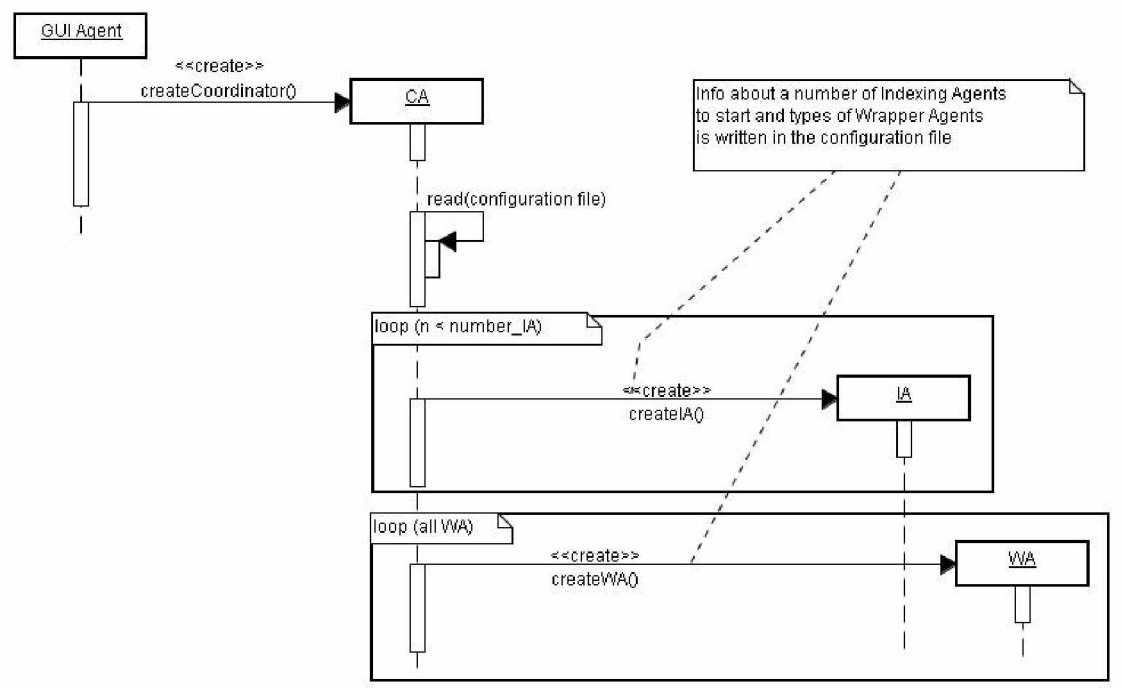


Rys. 13. CCS – Use-Case.

4.1 Start i zakończenie pracy podsystemu CCS

CCS jest niezależnym modułem wewnątrz *Systemu Wspomagającego Podróżowanie* (*SWP*), moduł ten może pracować nie tylko jako część *SWP* ale również jako samodzielna aplikacja zbierająca dane z Internetu i zapisująca je do bazy danych.

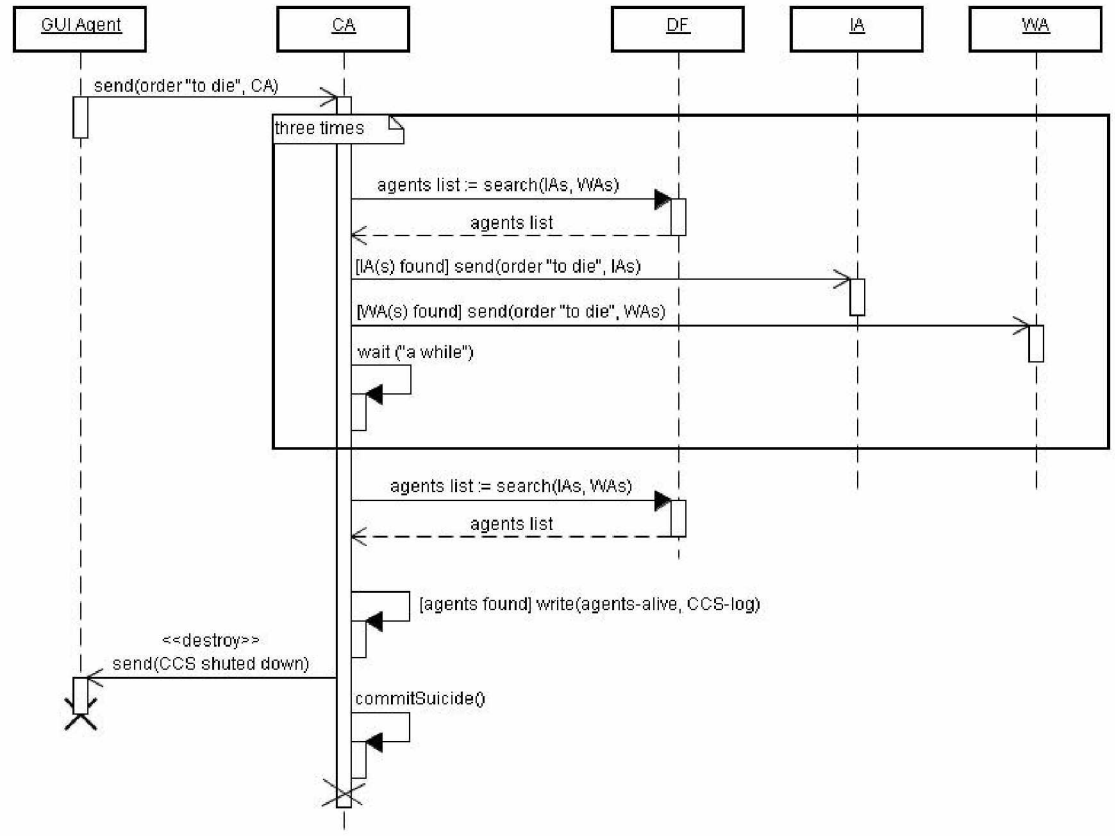
Kolejny diagram (Rysunek 14) przedstawia sekwencje operacji, które wykonywane są podczas startu *CCS*.



Rys. 14. Start pracy systemu CCS.

Niewidoczny na powyższym diagramie *administrator* systemu komunikuje się z agentem *GUI Agent* zlecając mu rozpoczęcie pracy *CCS*. W chwili obecnej *GUI Agent* jest bardzo prostą aplikacją desktopową, warto jednak zauważyć, iż mogłaby być to zarówno aplikacja webowa jak również aplikacja pracująca na urządzeniach mobilnych. Pierwsza i jednyna operacja wykonywana przez *GUI Agent*, to stworzenie koordynatora (*CA*). *GUI Agent* nie ma zaszytych w sobie żadnych informacji na temat tego w jaki sposób agenci *CCS* są tworzeni, wie on tylko jak utworzyć koordynatora. Agent *CA* rozpoczyna swoją pracę (metoda *setup*()) czytając plik konfiguracyjny. W pliku tym zapisane są informacje, jakich agentów *WA* stworzyć, jaki jest odstęp czasu pomiędzy kolejnymi uruchomieniami agentów *WA* a także ile agentów *IA* musi zostać stworzonych. W pierwszej kolejności tworzeni są agenci indeksujący znalezione dane *IA*, następnie agenci *WA*. Warto jednak podkreślić, iż kolejność tworzenia agentów nie wpływa na działanie systemu *CCS*.

Rozpoczęcie pracy *CCS* nie jest skomplikowanym procesem, zakończenie pracy *CCS* wymaga dużo większego nakładu pracy, co przedstawione jest na Rysunku 15.



Rys. 15. Zakończenie pracy CCS.

System *CCS* kończy swoją pracę tylko i wyłącznie na polecenie *administratora*, który to przy wykorzystaniu *GUI Agent*'a zleca koordynatorowi zabicie wszystkich pracujących agentów a następnie destrukcji samego siebie. *CA* po otrzymaniu rozkazu „order to *die"* (dokonaj samodestrukcji) przystępuje do operacji zabijania pracujących agentów *WA* oraz *IA*. W celu zabicia agentów *CA* wysyła im rozkaz samodestrukcji *„order to die"*, jednakże aby wysłać wiadomość potrzebny jest adres odbiorcy. *CA* nie posiada listy adresów pracujących agentów (co prawda wie on, którzy agenci pracują, ale nie jest to wystarczające). W celu zdobycia adresów, koordynator komunikuje się z agentem *Directory Faciliator* (*DF*). Agent *DF* jest częścią platformy JADE, można o nim myśleć

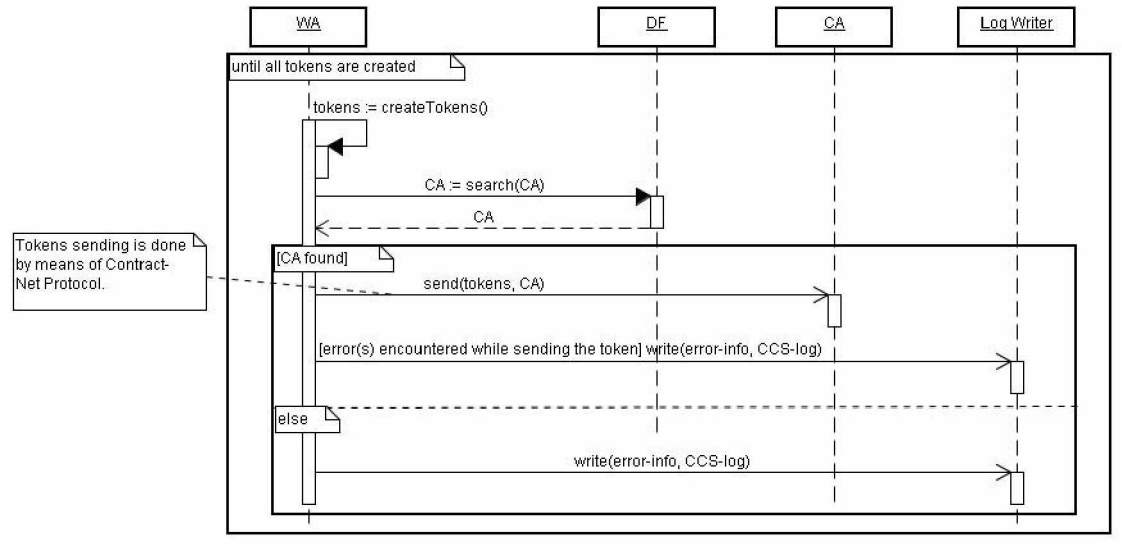
jak o serwisie dostarczającym informacji na temat platformy, agentów pracujących na platformie, itp. (więcej informacji znajduje się w [JADE]). Po otrzymaniu listy adresów od *DF*, *CA* wysyła agentom rozkaz samodestrukcji. Agenci *IA* oraz *WA* po otrzymaniu rozkazu natychmiastowo kończą bieżącą pracę, informują koordynatora, iż zaraz „zakończą istnienie" i dokonują samodestrukcji. Następnie *CA* czeka „chwilę" (ta „chwila" jest definiowana przez administratora) i powtarza czynność. Operacje poszukiwania pracujących agentów, wysłanie im rozkazu samodestrukcji oraz oczekiwanie powtarzane są w pętli trzy razy. Od razu narzuca się pytanie dlaczego trzy? Niestety nie istnieje praktyczne uzasadnienie wyboru cyfry trzy. W obecnej wersji systemu jest to trzy, jednakże w przyszłych wersjach liczba iteracji będzie określana w dedykowanym pliku konfiguracyjnym koordynatora. Kolejne pytanie mogłoby dotyczyć pętli, dlaczego operacje te wykonywane są w pętli? Czy nie wystarczy jak ciąg tych operacji zostanie wykonany tylko raz? Okazuje się, że nie. Należy pamiętać, iż *CCS* jest systemem rozproszonym, agenci *IA* oraz *WA* mogą pracować na różnych komputerach, można sobie wyobrazić, iż pracują oni w różnych podsieciach. Tak więc musimy założyć, że rozkaz samodestrukcji może do nich nie dotrzeć, wystarczyłoby na przykład, aby przez chwilę połączenie sieciowe zostało zerwane (wysyłanie wiadomości nie jest transakcyjne) aby nie dotarła ona do agenta do którego była skierowana. Kończąc pracę systemu, chcielibyśmy aby prawdopodobieństwo, iż jakiś agent pozostanie żywy było jak najmniejsze. Niestety stu procentowej pewności nie będziemy mieli nigdy. Z tego też powodu pierwszą operacją po wyżej przedstawionej pętli jest ponowne poszukiwanie wciąż pracujących agentów. Znowu może pojawić się pytanie: dlaczego ponownie *CA* poszukuje pracujących agentów, przecież tuż przed śmiercią każdy agent wysyła wiadomość do koordynatora, że zaraz ulegnie samodestrukcji? Ponowne poszukiwanie agentów typu *IA* oraz *WA* przy użyciu agenta *DF* jest bardziej niezawodne. Wiadomość od agenta, który zaraz zakończy działanie, może, po prostu, nie dotrzeć do celu bądź po wysłaniu wiadomości agent może nie ulec samodestrukcji. Jeśli istnieją wciąż pracujący agenci, informacja o nich zapisywana jest do logu *CCS*. Kolejna operacja jaką wykonuje *CA* to przesłanie wiadomości do *GUI Agent*'a mówiącej, iż *CCS* jest zamknięty (aby informacja o zakończeniu pracy *CCS* trafiła do administratora). Następnie *CA* ulega samodestrukcji.

Informacja na temat wciąż pracujących agentów zapisywana jest w logu systemowym. W obecnej wersji systemu, logi odnośnie agentów, którzy nie zakończyli działania podczas zamykania systemu wykorzystywane są tylko przez administratora systemu. Dla każdego wpisu w logu, administrator ręcznie dokonuje destrukcji, przy wykorzystaniu *JADE Remote Agent Management GUI* [JADE], agentów (oczywiście jeśli rzeczywiście jeszcze nadal istnieją). Każdy się zgodzi, iż nie jest to zbyt interesująca czynność, co więcej jest ona mechaniczna. Z tego też powodu, w następnej wersji *CCS* koordynator będzie wyręczał administratora. Przy starcie systemu, *CA* będzie czytał z logu (przy użyciu nowego agenta: *Logging Agent* (*LA*), więcej informacji na temat logowania w podrozdziale 4.5) informacje odnośnie agentów, którzy przeżyli ostatnie zamknięcie systemu i sam będzie próbował dokonać ich destrukcji przed rozpoczęciem innych czynności.

4.2 Komunikacja WA – CA oraz IA - CA

W podrozdziale tym przedstawię komunikację pomiędzy agentami jaka odbywa się w procesie tworzenia tokenu i zapisywania go w bazie danych. Komunikacja pomiędzy agentami systemu wykonywana jest przy użyciu wbudowanych mechanizmów JADE. Wiadomości przesyłane między agentami zapisane są w języku *ACL* [ACL]a komunikacja zgodna jest ze specyfikacją *FIPA* [FIPA]. W pierwszej kolejności przedstawię ścieżkę życia tokenu.

Token zawierający dane opisujące dowolny obiekt tworzony jest przez agenta *WA*. Następnie *WA* przesyła stworzony token do koordynatora (*CA*), koordynator zapisuje stworzony token do wewnętrznej priorytetowej kolejki tokenów. Agent *IA* przesyła zapytanie o token do *CA*, jeśli koordynator posiada co najmniej jeden token, wybiera ten o najwyższym priorytecie i przesyła go agentowi *IA*. Rysunek 16 przedstawia przesyłanie nowo stworzonych tokenów od agenta typu *WA* do koordynatora (*CA*).

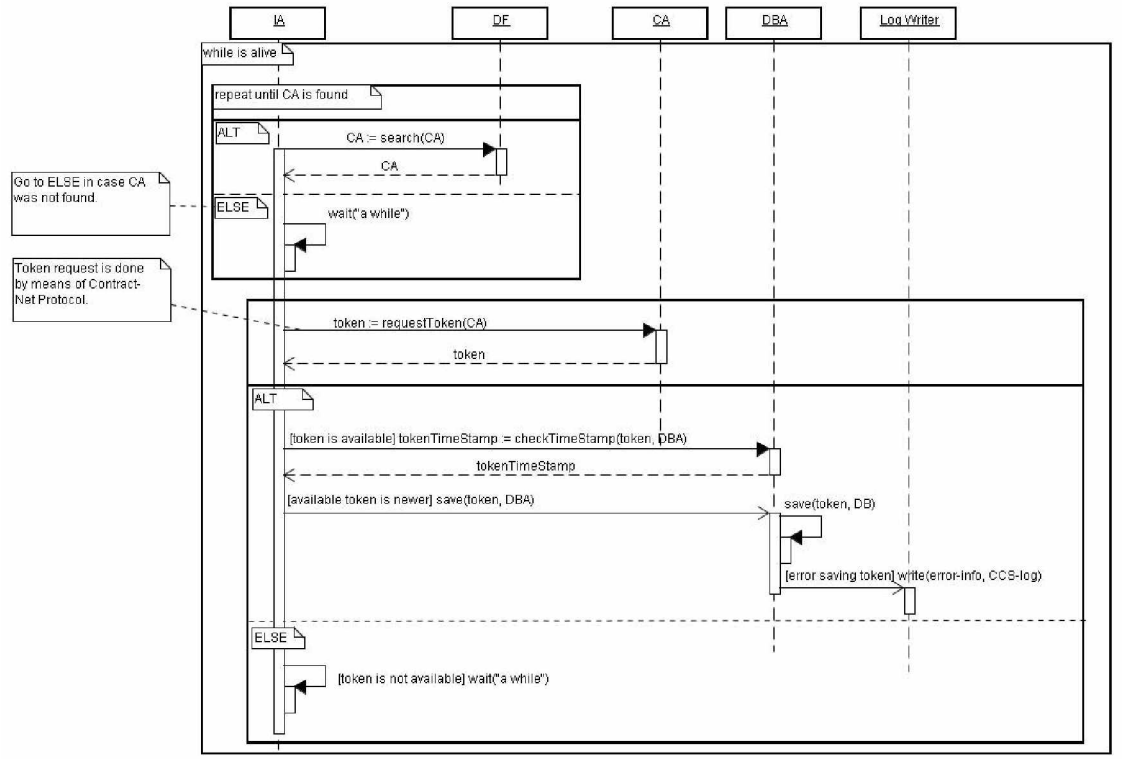


Rys. 16. Przesyłanie kolekcji tokenów od *WA* do *CA*.

Główna praca *WA* to tworzenie tokenów z danymi opisującymi pewne obiekty związane z turystyką. Z jednego źródła danych, np.: portalu informacyjnego może powstać wiele tokenów. W związku z tym należy rozpatrzyć dwa scenariusze wysyłania tokenów. Agent *WA* może przesyłać koordynatorowi po jednym tokenie lub może zbierać tokeny i przesyłać kolekcję tokenów za jednym razem. Bardziej uniwersalne jest przesyłanie kolekcji tokenów, przecież zawsze można przesyłać kolekcję składającą się tylko z jednego elementu. Poza tym, w przypadku, gdy tokeny są małe i wysyłane często niepotrzebnie zwiększany jest ruch sieciowy. W obecnej wersji systemu przesyłana jest kolekcja tokenów a maksymalny rozmiar kolekcji tokenów zaszyty jest na stałe w kodzie programu. W następnej wersji systemu maksymalny rozmiar kolekcji tokenów będzie definiowany w dedykowanym pliku każdego agenta *WA*. Administrator przyglądając się pracy agentów *WA* będzie mógł odpowiednio dostosować rozmiary kolekcji tokenów.

Z chwilą rozpoczęcia pracy, agent *WA* przystępuje do tworzenia tokenów. Po stworzeniu wystarczającej liczby tokenów, *WA* przystępuje do wysłania ich. Wysyłanie kolekcji tokenów wykonywane jest przy wykorzystaniu protokołu zdefiniowanego przez *FIPA*: *Contract-Net Protocol* (podstawowe informacje na temat protokołu przedstawione zostały w podrozdziale 2.3.4). Pierwszym etapem jest znalezienie listy koordynatorów (w chwili obecnej jest tylko jeden koordynator, jednakże zakładamy, iż w przyszłości może

być ich więcej). Poszukiwanie odbywa się przy pomocy wbudowanego agenta platformy JADE: agenta *Directory Faciliator*. Kolekcja tokenów przesyłana jest do znalezionego koordynatora. Jeśli wystąpił błąd podczas przesyłania kolekcji, informacja o błędzie wraz z tokenami zapisywana jest do logu systemowego. W przypadku, gdy koordynator nie został znaleziony, informacja o braku koordynatora wraz z całą kolekcją tokenów również zapisywana jest w logu. Dzięki temu, iż tokeny zapisywane są do logu systemowego, administrator będzie mógł „ręcznie" zapisać je w bazie danych. Można by zadać sobie następujące pytania: kiedy koordynator może nie być znaleziony oraz czy jeśli *WA* nie może znaleźć *CA* to czy dalej powinien wykonywać swoją pracę? Koordynator może nie zostać znaleziony, na przykład, gdy komputer na którym pracuje *CA* stracił połączenie z siecią, w której znajduje się agent *WA*. Odpowiedź na drugie pytanie nie jest prosta, w przypadku, gdy brak koordynatora spowodowany jest chwilowym brakiem dostępu do Internetu, *WA* może wykonywać swoją normalną pracę bez najmniejszego problemu – zaraz ponownie będzie miał połączenie z *CA*. Jednakże warto zauważyć, iż gdy koordynator nie istnieje – np. nastąpiła awaria serwera, na którym działał koordynator, wszyscy agenci podsystemu *CCS* powinni wstrzymać pracę, do czasu, aż koordynator zacznie ponownie pracować. W obecnej wersji systemu, agenci *WA* nie kończą swojej pracy nawet, gdy po wielokrotnych próbach *CA* nie jest znaleziony. W kolejnych wersjach systemu, administrator w pliku konfiguracyjnym dla podsystemu *CCS* będzie definiował maksymalną liczbę kolejno nieudanych prób znalezienia koordynatora, po których agenci podsystemu *CCS* będą wstrzymywali pracę. Wznowienie działania nastąpi zaraz po ponownym przystąpieniu koordynatora do pracy. Kolejny rysunek (Rysunek 17) przedstawia przesłanie tokenu od agenta *CA* do agenta *IA*.



Rys. 17. Przesyłanie kolekcji między *IA* a *CA*.

Agent indeksujący *IA* odpowiedzialny jest za zapisywanie tokenów w centralnym repozytorium. Prześledźmy jak agent ten pobiera token od koordynatora. Pierwszym etapem jest znalezienie koordynatora, jeśli *CA* nie został znaleziony, *IA* czeka „chwilę" a następnie ponownie próbuje zdobyć adres agenta *CA* przy wykorzystaniu agenta *Directory Faciliator*. Przejście do następnej operacji następuje dopiero po znalezieniu koordynatora. Do znalezionego *CA* wysyłane jest żądanie o token, jeśli *CA* posiada przynajmniej jeden token, przesyła go agentowi *IA*. Kolejnym krokiem jest porównanie daty stworzenia tokenu, który ma zostać zapisany z datą stworzenia tokenu znajdującego się w bazie danych opisującego ten sam zasób (jeśli oczywiście token opisujący ten sam zasób istnieje). Jeśli token ten jest nowy lub data stworzenia jego jest późniejsza niż data stworzenia tokenu z bazy danych, token ten zapisywany jest w bazie danych. Porównanie dat tokenów oraz zapisanie tokenu w centralnym repozytorium wykonywane jest przy pomocy agenta *DBA*. Jeśli podczas zapisu tokenu do bazy danych wystąpi błąd, informacja o tym wraz z całym tokenem zapisywana jest w logu systemowym. Dzięki

temu, administrator będzie mógł „ręcznie” zapisać token w bazie danych. W przypadku, gdy koordynator nie posiada żadnych tokenów, agent *IA* czeka „chwilę” i ponownie powtarza przedstawione powyżej czynności. W ten sposób zapewniamy stały odbiór tokenów od *CA*.

Omawiając przesyłanie tokenów między agentami *WA*, *CA* oraz *IA* warto przyjrzeć się jeszcze na chwilę protokołowi *Contract Net Protocol* (opisanemu w 2.3.4). W protokole tym zdefiniowany jest tylko jeden „timeout”. Inicjator protokołu określa maksymalny okres czasu na otrzymywanie odpowiedzi na jego propozycję zadania (*call for proposal*). Po zdefiniowany przez inicjatora czasie, odpowiedzi są ignorowane. Niestety podobnego „timeout’u” nie można zdefiniować na czas wykonywania powierzonego zadania. W protokole tym, inicjator może czekać w nieskończoność na wynik od agenta, który podjął się wykonania zadania. W bieżącej wersji systemu, w przypadku, gdy komputer, na którym pracuje wykonawca zadania uległ awarii a wykonawca zadania zostanie „zabity”, inicjator zadania zostanie zablokowany – będzie czekał na zakończenie zadania w nieskończoność. W kolejnej wersji systemu protokół ten zostanie trochę zmodyfikowany i zostanie dodana możliwość definiowania maksymalnego czasu na wykonanie zadania. Maksymalny czas na wykonanie zadania będzie określany przez administratora i dostarczany do systemu przy wykorzystaniu plików konfiguracyjnych.

4.3 Uwagi odnośnie koordynatora (CA)

W aktualnej wersji systemu koordynator służy jako „skrzynka” w procesie komunikacji między agentami *WA* a *IA*. Można by się zastanowić czy w pewnym momencie nie stanie się on tzw. „wąskim gardłem” podsystemu *CCS*. Jeśli duża liczba *Wrapper-ów* oraz agentów *IA* będzie pracowała w tym samym czasie, może to być rzeczywiście problem. W tej sytuacji możliwym jest zastosowanie puli agentów *CA* w miejsce pojedynczego koordynatora. W tym przypadku agenci *WA* oraz *IA* kontaktowaliby się z pulą agentów *CA*. Agent *WA* pytałby się agentów *CA*, który z nich ma najmniej tokenów i nowo stworzony token wysyłałby agentowi *CA* z najmniejszą ich liczbą. Z drugiej strony agent *IA* pytałby się agentów *CA*, który z nich ma najwięcej tokenów i pobierał token od agenta

z największą ich liczbą. Warto przypomnieć, iż komunikacja między agentami *WA* i *CA* oraz *IA* i *CA* zaimplementowana jest przy użyciu *Contract Net Protocol* [Contract Net Protocol]. Dzięki temu przejście do puli agentów *CA* wymagałoby minimalnych zmian w celu zaadoptowania nowego modelu.

Warto się zastanowić, gdzie przejście z pojedynczego koordynatora do puli koordynatorów wymagałoby również uwagi. Okazuje się, że start i zakończenie pracy systemu wymagałyby zmian. Start systemu mógłby wyglądać następująco. *GUI Agent* uruchamiałby jednego *CA* (tak jak jest to w obecnej wersji systemu), jednakże nowo uruchomiony *CA* byłby odpowiedzialny również za tworzenie puli koordynatorów. Liczba koordynatorów, jakie powinien stworzyć byłaby zapisana w pliku konfiguracyjnym (zatem w pliku konfiguracyjnym musiałby pojawić się nowy wpisy). Z drugiej strony przy zakończeniu pracy systemu, agent *CA* nie tylko byłby odpowiedzialny za „zabicie" agentów *IA* oraz *WA*, musiałby on również „zabić" pozostałych agentów *CA*. Zakończenie pracy systemu *CCS* mogłoby wyglądać następująco. *GUI Agent* wysyłałby do jednego koordynatora z puli (np. losowo wybranego) informacje o tym, iż system powinien zakończyć pracę. Koordynator ten powinien rozesłać rozkaz „*order to die*" nie tylko do agentów *WA* oraz *IA* tak jak robi to teraz ale rozkaz ten musiałby być również wysłany do pozostałych agentów *CA*. Warto tu podkreślić, iż agent typu *CA* musiałby inaczej reagować na odebranie tego rozkazu w przypadku, gdy nadawcą jest *GUI Agent* i w przypadku, gdy nadawcą jest inny agent *CA*.

4.4 Uwagi odnośnie agentów typu Wrapper Agent (WA)

Proponowany tutaj system odpowiedzialny za gromadzenie danych jest na tyle uniwersalny, iż można go traktować jako framework do budowy systemów klasy *Content Collection*. Jedyne elementy, które będą się zmieniały to agenci *WA*. Zauważmy, iż jeśli powstaną w Internecie repozytoria danych opisanych semantycznie, to wystarczy wówczas stworzyć agentów *WA* odczytujących tak opisane dane i tworzących odpowiednie tokeny. Konieczność tworzenia kolejnych *WA* jest słabym punktem

przedstawionego rozwiązania. Zastanówmy się w jaki sposób można by zmniejszyć ilość pracy potrzebnej do stworzenia agentów tego typu.

W chwili obecnej dla każdej strony internetowej musimy stworzyć „ręcznie” agenta *WA*, który doskonale rozumie, zna prezentowany tam format informacji. Nawet niewielka zmiana formatu powoduje konieczność przebudowy agenta. Pierwszym pomysłem jest system, który automatycznie generowałby agentów typu *Wrapper Agent* dla zadanej strony WWW. System taki, np. przy wykorzystaniu sieci neuronowych, byłby w stanie rozpoznawać strukturę strony, tabele, zwykły tekst, itp.. Czy jednak budowa takiego systemu nie byłaby o wiele trudniejsza od budowy naszego podsystemu *CCS*? Problem ten omawiany jest w [Ashish 1997], [Knoblock 2000], [Lerman 2004a], [Lerman 2004b], [Muslea 2001].

Inne rozwiązanie koncentruje się wokół aplikacji typu *Web Crawler* (*WC*), a bardziej szczegółowo wokół rozwiązań typu *Focused Crawler* (*FC*). Zamiast agenta *WA* przydzielonego do konkretnego źródła danych mamy tutaj agentów typu *Focused Crawler Agent* (*FCA*) odwiedzających zasoby, „czytających” informacje, próbujących je „zrozumieć” i przekształcić w trójki RDF. Największą trudnością w tym podejściu jest zbudowanie agenta, który zrozumie dane nieustruktualizowane. Więcej informacji o *Web Crawler* i *Focused Crawler* można znaleźć w [Bradshaw 2003], [Chau 2003], [Ehrig, Master’s Thesis], [Novak 2004], [Pant 2003].

Należy tutaj jednak podkreślić, że bez względu jak będzie rozwijała się wiedza na temat tworzenia agentów typu *WA*, architektura naszego systemu *CCS* pozwala na rozwijanie *Wrapper-ów*, bez ingerencji w innych agentów (w pozostałe części systemu).

4.5 Logowanie w systemie CCS

Skuteczne śledzenie pracy dowolnego systemu wymaga logowania różnego typu zdarzeń. Logowanie jest nieodłącznym elementem każdego dobrze zaprojektowanego i stworzonego systemu rozproszonego. W systemach tego typu, do diagnozowania błędów

standardowy *debbuger* nie wystarcza, potrzeba narzędzia zapisującego stan konkretnych elementów systemu w jednym miejscu. Analiza komunikatów pozwala na łatwiejsze wykrycie komponentu powodującego nieprawidłowe działanie. Przy budowie systemu *CCS* do logowania zdarzeń wykorzystany jest, znany w środowisku programistów Javy, moduł Log4j (podrozdział 3.8 zawiera więcej informacji). Jednakże wydaje się, iż rozwiązanie takie nie jest wystarczająco dobre dla systemów agentowych. Agenci mogą przecież pracować w różnych lokalizacjach, różnych podsieciach, w jaki sposób zatem konfigurowalibyśmy sposób logowania? Jedynym rozwiązaniem byłoby tworzenie dedykowanych plików konfiguracyjnych dla pojedynczych agentów (ewentualnie dla grup agentów). Co prawda pliki te byłyby takie same, jednakże zmiana sposobu logowania wymagałaby podmienienia wszystkich plików konfiguracyjnych. Co gorsza w przypadku, zmiany modułu odpowiedzialnego za logowanie zmianom ulegliby wszyscy agenci (wszyscy agenci korzystają z modułu logującego). Pomimo tych niedogodności, z powodu ograniczeń czasowych taki sposób logowania został zaimplementowany. Jednakże z nową wersją systemu pojawi się nowe rozwiązanie. Zostaną dodani do systemu agenci nowego typu: *Logging Agent* (*LA*). Agent typu *LA* będzie służył jako jedyny punkt dostępu do logu systemowego (podobnie jak agent *DBA* w przypadku dostępu do bazy danych). Dowolny agent systemu zamiast samemu wykorzystywać moduł logujący, będzie wysyłał komunikat (który powinien być zapisany do logu) do agenta typu *LA*. Dzięki takiemu podejściu, definiowanie sposobu logowania, lokalizacji logu, jego typu (zwykły plik tekstowy, baza danych czy Web service) byłoby zawarte w jednym miejscu. Co więcej zmian modułu odpowiedzialnego za logowanie wymagałoby zmian tylko w implementacji agentów *LA*, pozostałe części systemu pozostały by nienaruszone.

4.6 Raportowanie w systemie CCS

W obecnej wersji systemu agenci tylko gromadzą dane, gdy administrator systemu chce obejrzeć dane zebrane przez agentów musi „ręcznie” zaglądać do bazy danych. Przeglądanie tych danych nie jest wygodne, schemat bazy danych jest tworzony przez framework JENA a dane nie są bezpośrednio przechowywane w postaci trójek RDF.

Następna wersja systemu będzie powiększona o kolejnego agenta, agent typu: *Reporting Agent* (*RA*). Jak sama nazwa wskazuje, agent ten będzie odpowiedzialny za tworzenie raportów na podstawie danych zebranych w centralnym repozytorium. Dzięki agentowi *RA* administrator w łatwy sposób będzie mógł przyglądać się pracy konkretnych agentów. Przykładowe raporty to:

- Liczba wszystkich zasobów opisanych przy pomocy trójek RDF
- Liczba wszystkich trójek RDF
- Lista nazw wszystkich zasobów
- Lista nazw wszystkich zasobów, które zostały opisane przez konkretnego agenta *WA*, np. agenta odpowiedzialnego za gromadzenie danych na temat hoteli Sheraton
- Lista wszystkich trójek RDF opisujących konkretny obiekt, np. Hotel Marriott w Warszawie
- Lista zasobów, które zostały stworzone/zmodyfikowane przez konkretnego agenta *WA* w danym okresie czasu

Zdefiniowane na stałe raporty czasami nie są wystarczające, z tego też powodu administrator systemu będzie mógł wysyłać zapytania w języku RDQL do agent *RA* (z poziomu panelu administracyjnego), a ten zwracał będzie kolekcję trójek RDF spełniających zapytanie.

4.7 Implementacja

Podsystem *CCS* został zbudowany przy wykorzystaniu JADE – otoczenia do tworzenia aplikacji agentowych. Podczas „wyłuskiwania” danych ze stron WWW wykorzystywany jest CyberNeko HTML Parser [HTML Parser] – parser HTML, który dla zadanej strony tworzy reprezentujący ją obiekt DOM [DOM] oraz Jaxen [JAXEN] –XPath engine [XPath]. Strony ładowane są do pamięci przy użyciu biblioteki HTTP Client [HTTP Client]. Logowanie zdarzeń wykonywane jest przy użyciu biblioteki Log4j [LOG4J]. Jako centralne repozytorium wykorzystywana jest relacyjna baza danych MySQL [MySQL].

CyberNeko HTML Parser

NekoHTML jest narzędziem umożliwiającym programistom parsowanie dokumentów w formacie HTML. Parser ten „skanuje" pliki HTML i „naprawia" wiele często popełnianych błędów podczas tworzenia stron WWW zarówno przez ludzi jak i przy użyciu dedykowanego oprogramowania. NekoHTML dodaje, między innymi, brakujące nadrzędne elementy oraz brakujące tagi zamykające. Parser ten został napisany przy użyciu Xerces Native Interface [XNI]. Dostępny jest on pod licencją Apache (Apache-style open source license).

Jaxen

Jaxen jest to XPath engine, który potrafi „obliczać" wyrażenia XPath dla następujących modeli danych: dom4j, JDOM oraz DOM. Silnik ten jest produktem open source dostępnym pod licencją Apache (Apache-style open source license).

HTTP Client

Komponent ten jest produktem open source, całkowicie napisanym w języku Java, obsługującym protokół HTTP w wersji 1.0 oraz 1.1. HTTP Client implementuje wszystkie metody protokołu HTTP (GET, POST, PUT, DELETE, HEAD, OPTIONS oraz TRACE). Wspiera on między innymi: szyfrowanie przy użyciu HTTPS (HTTP przy użyciu protokółu SSL), autentykację BASIC oraz DIGEST czy automatyczną obsługę ciasteczek (cookies).

Log4j

Log4j jest modułem logującym, napisanym w języku Java. Moduł ten jest łatwy w obsłudze i konfiguracji, jego możliwości są naprawdę duże. Komunikaty, które chcemy zalogować można podzielić na pięć podstawowych typów:

- Fatal Error – błąd krytyczny, po którym aplikacja nie może pracować
- Error – „normalny" błąd aplikacji, aplikacja nie musi kończyć pracy
- Warning – ostrzeżenie, po którym aplikacja może pracować, jednakże istnieje pewne zagrożenie, iż może ona przestać poprawnie działać
- Info – komunikat informacyjny

- Debug - komunikat dla programisty analizującego działanie aplikacji

Komunikaty można wypisywać do pliku, na konsolę, do bazy danych czy poprzez wysłanie e-mailu.

4.8 Testowanie

Poprawność działania systemu *CCS* weryfikowana jest przy użyciu dwóch narzędzi do testowania. Pierwsze z nich to powszechnie znane środowisko przeznaczonego do tworzenia testów jednostkowych (unit tests) – JUnit [JUnit]. Jednakże środowisko to nie jest wystarczające w przypadku *CCS*, gdyż system nasz jest systemem agentowym a JUnit jest narzędziem zaprojektowanym do testowania obiektów i symuluje ich działania wywołując odpowiednie metody. Główną różnicą pomiędzy agentami a normalnymi obiektami jest to, iż obiekty udostępniają metody, które mogą być wywołane przez zewnętrzne encje w dowolnym czasie. Agenci, natomiast, nie udostępniają metod. Agenci są autonomicznymi jednostkami komunikującymi się ze sobą wysyłając wiadomości. Wiadomości przetwarzane są, kiedy to odbiorca wiadomości chce je przetworzyć – nie jak metody obiektów, które po wywołaniu natychmiast się wykonują. W związku z powyższym, środowisko JADE udostępnia specjalizowany framework do testowania zachowań agentów stworzonych na platformę JADE: JADE TestSuit [JADE]. JADE TestSuit oparty jest na dwu poziomowym modelu. Na pierwszym poziomie definiujemy główną funkcjonalność, dla przykładu niech będzie to komunikacja między agentami. W celu sprawdzenia, czy dobrze ona działa wiele przypadków tej funkcjonalności musi być rozpatrzonych. Prowadzi to nas do drugiego poziomu, na którym to identyfikujemy poszczególne jej aspekty. Poniżej przedstawione są główne testy systemu *CCS* przy wykorzystaniu JADE TestSuite.

Testy agentów typu „Wrapper Agent”

1. Agent próbuje połączyć się ze źródłem danych, jednakże dostęp do niego nie jest możliwy, np.: serwer WWW nie działa. Agent powinien zakończyć swoją pracę i zapisać informację o tym zdarzeniu do logu systemowego.

2. Agent czyta dane ze strony WWW, jednakże prezentacja informacji tam zawartych zmieniła się (w wyniku czego wystąpił błąd parsowania strony). Agent powinien zakończyć swoją pracę i zapisać informację o tym zdarzeniu do logu systemowego.

Testy agenta typu „Indexing Agent"

1. *IA* otrzymał token od koordynatora (*CA*) do zapisania. Próbuje zapisać dane z tokenu do bazy danych, jednakże występuje błąd. Agent powinien zapisać informację o błędzie wraz z całym tokenem do logu systemowego i powrócić do normalnej pracy.

Testy komunikacji Wrapper Agent – Coordinator Agent

1. *Wrapper* stworzył token i chce go wysłać, ale nie ma żadnego agenta typu *Coordinator Agent* – nie ma go komu wysłać. Agent powinien zapisać informację o braku koordynatora wraz z całym stworzonym tokenem do logu systemowego i powrócić do pracy.
2. *Wrapper* stworzył token i wysyła go do koordynatora, w pewnym momencie łączność z koordynatorem zostaje zerwana (np.: uległ awarii komputer, na którym działał koordynator). W takich sytuacjach *Wrapper* powinien powtarzać proces wysyłania tokenu, aż do skutku.

Testy komunikacji Indexing Agent – Coordinator Agent

1. *Indexing Agent* nie jest niczym zajęty, poprosi więc koordynatora o token do zapisania w bazie. Jednakże, nie ma żadnego agenta typu *Coordinator Agent – IA* nie ma do kogo wysłać prośby. *IA* powinien poczekać „chwilę" i spróbować ponownie.

5. Przykłady działania systemu CCS

W poniższym rozdziale zobrazuję proces transformacji danych ze stron typu *VCP* opisujących hotele w tokeny, które przechowywane są w centralnym repozytorium przy wykorzystaniu platformy JENA.

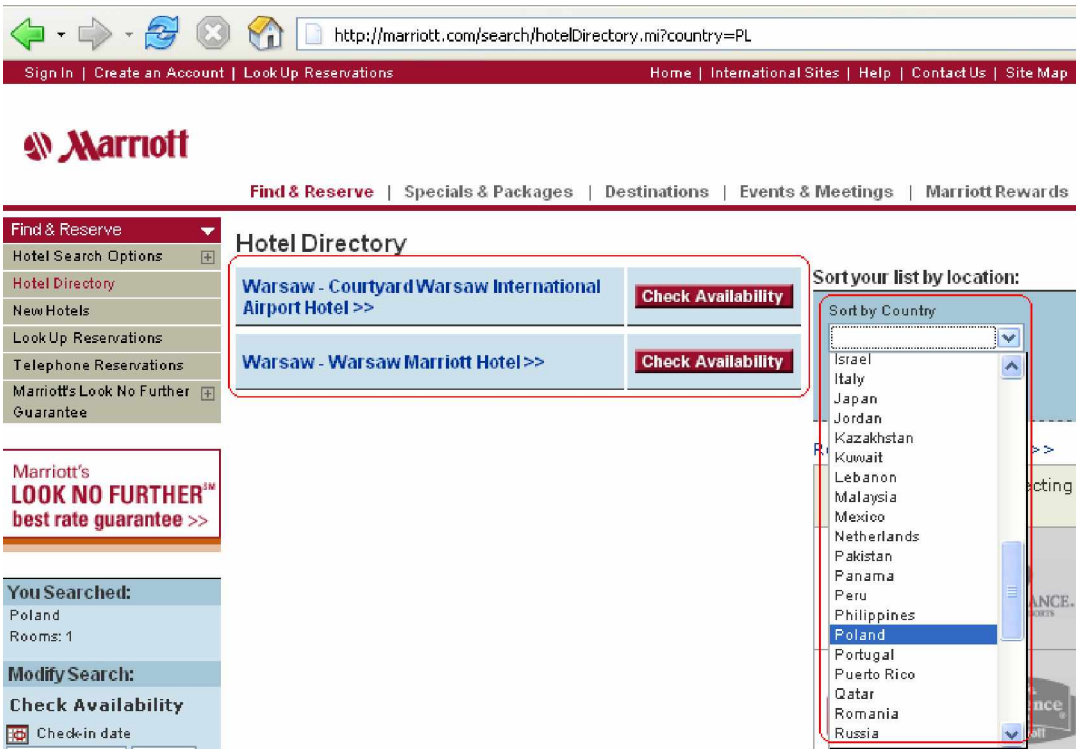
Schemat działania agentów zbierających dane na temat hoteli jest bardzo podobny. Pierwszym etapem jest zlokalizowanie miejsc, w których znajdują się hotele. Lokalizacje te dzielą się na państwa oraz stany (w przypadku USA). Po określeniu wszystkich lokalizacji, dla każdej z nich wyszukiwane są wszystkie znajdujące się tam hotele. Następnie dla każdego znalezionego hotelu odwiedzane są strony WWW zawierające szczegółowe informacje na jego temat. Agent *WA*, znając układ strony, wyłuskuje interesujące go informacje (zdefiniowane przez *Hotel Ontology*) przy wykorzystaniu wyrażeń XPath. Z wyłuskanych informacji budowane są trójki RDF tworzące instancję ontologii *Hotel Ontology*. Następnie do zbioru zbudowanych trójek dodawana jest data stworzenia oraz określany jest ich priorytet. W ten sposób powstaje token opisujący hotel. Agent *WA* po stworzeniu tokena sprawdza liczbę wcześniej wytworzonych tokenów w swojej wewnętrznej kolejce. Jeśli całkowita liczba tokenów jest równa maksymalnej liczbie tokenów, jaka może być jednorazowo przesłana w kolekcji do koordynatora bądź jeśli jest to ostatni stworzony token przez *WA*, agent buduje wtedy wiadomość w języku ACL (na podstawie tych tokenów) i wysyła ją do koordynatora (*CA*). W przeciwnym razie agent zapisuje token w wewnętrznej kolejce i wraca do dalszej pracy. Koordynator umieszcza przesłany token do kolejki priorytetowej, aby został on później zapisany w centralnym repozytorium przez agenta *IA*. Po zebraniu danych dla wszystkich hoteli, agent *WA* informuje koordynatora o zakończeniu zadania a następnie ulega samodestrukcji.

W kolejnych podrozdziałach przedstawię proces zbierania danych dla trzech agentów typu *Wrapper Agent*: *MarriottHotelsWrapperAgent*, *HiltonHotelsWrapperAgent* oraz *StarwoodHotelsWrapperAgent*.

5.1 Marriott Hotels Wrapper Agent

Agent *MarriottHotelsWrapperAgent* (*MHWA*) zbiera informacje na temat wszystkich hoteli Marriott na świecie. Jako źródło danych wykorzystuje on strony z portalu firmy Marriott: `http://marriott.com`.

Portal firmy Marriott umożliwia przeszukiwanie hoteli według państw oraz według stanów dla USA (Rysunek 21). Pierwszym zadaniem agenta *MHWA* jest znalezienie wszystkich lokalizacji, gdzie występują hotele Marriott. Strona WWW znajdująca się na Rysunku 21 jest pierwszą stroną odwiedzaną przez *Wrapper-a*:
`http://marriott.com/search/hotelDirectory.mi?country=PL`.
Poczynione zostało tutaj założenie, że w Polsce (query string: `?country=PL`) istnieje co najmniej jeden hotel Marriott. Po prawej stronie znajduje się lista rozwijana z państwami/stanami, w których występują hotele (na rysunku lista ze stanami jest zasłonięta przez rozwiniętą listę państw). *MHWA* „wyłuskuje" wszystkie kody państw oraz stanów USA z rozwijanych list i dla każdego kodu odwiedza stronę z detalami hotelu zmieniając odpowiednio „query string". Dla kodów państw „query string" jest następujący:
`?country=[kod państwa]`, np. dla Japonii jest to: `?country=JP`,
dla stanów USA:
`?state=[kod stanu]`, np. dla stanu Kalifornia jest to: `?state=CA`.



Rys. 18. Hotele Marriott (strona startowa agenta).

Z odwiedzonych stron agent odczytuje kody wszystkich hoteli występujących w danym państwie/stanie (środkowa część strony na Rysunku 21). Kod HTML, z powyżej przedstawionej strony, z którego wyłuskiwane są kody hoteli Marriott w Polsce to:

```
<div class="searchResultsList hotelDirectory">
  <table border="0" width="0" cellspacing="0">
    …
    <tr class="odd">
      <td valign="middle" class="propertyName rule">
        <a href="/property/propertyPage/WAWCY">Warsaw - Courtyard
            Warsaw International Airport Hotel &gt;&gt;</a>
      </td>
      …
    </tr>
    <tr class="odd">
      <td valign="middle" class="propertyName rule">
        <a href="/property/propertyPage/WAWPL">Warsaw - Warsaw Marriott
            Hotel &gt;&gt;</a>
      </td>
      …
    </tr>
  </table>
</div>
```

W celu wyłuskania kodów hoteli (zaznaczonych na kolor niebieski), agent wyznacza wartość następującego wyrażenia XPath:

```
//DIV[@class = 'searchResultsListhotelDirectory']
/TABLE[1]/TR[@class='odd']
```

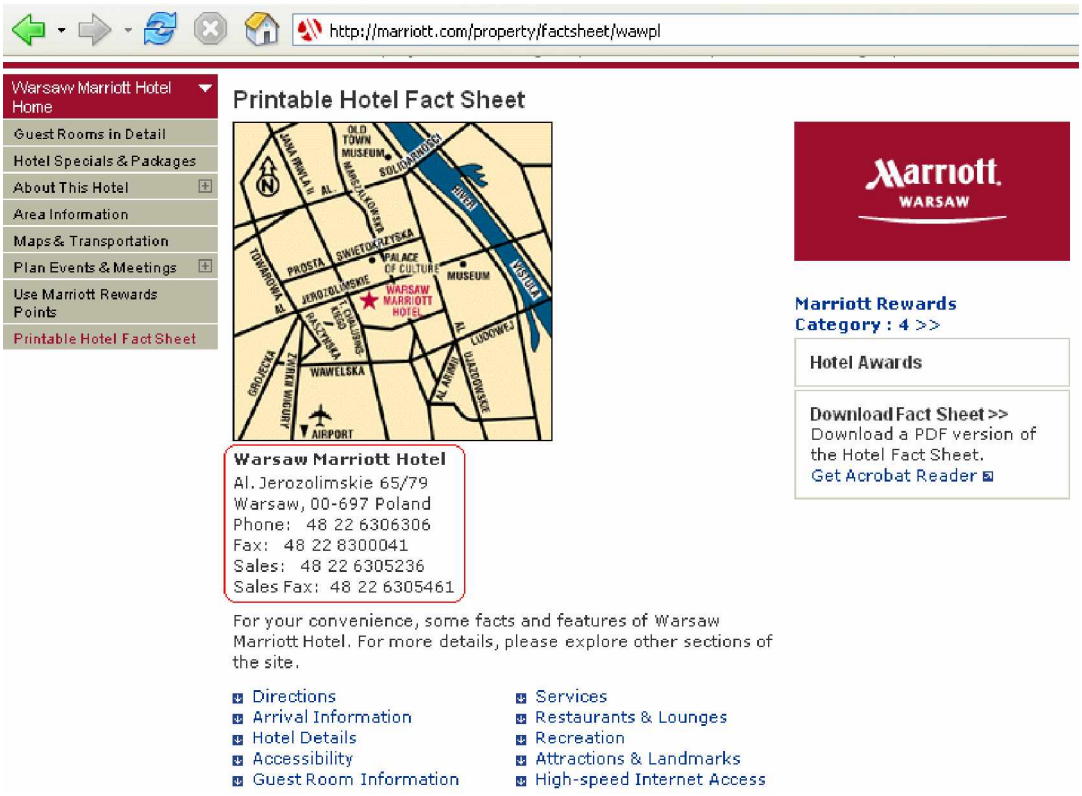
w odpowiedzi otrzymując wszystkie węzły typu *TR* z atrybutem *class* mającym wartość *odd*. Dla każdego otrzymanego węzła *TR*, wyznaczany jest atrybut *HREF* potomka typu *A*. Następnie z wartości tego atrybutu wycinany jest ciąg znaków znajdujący się za ostatnim ukośnikiem „/" – to właśnie jest kod hotelu.

Posługując się otrzymanymi kodami hoteli, można pobrać szczegóły. Adres strony z detalami hoteli ma następujący format:

`http://marriott.com/property/factsheet/`**`[kod hotelu]`**

Dla każdego kodu hotelu, *MHWA* odwiedza stronę z detalami podmieniając w powyżej podanym adresie URL posiadany wcześniej kod hotelu. Po załadowaniu strony z detalami, agent wyłuskuje interesujące go dane i tworzy trójki RDF opisujące hotel. Przyjrzyjmy się jakie dokładnie informacje wyłuskiwane są dla warszawskiego hotelu Marriott (kod: WAWPL).

Kolejny rysunek (Rysunek 22) przedstawia górną część strony z detalami na temat warszawskiego hotelu Marriott. Z fragmentu tego, agent odczytuje informacje adresowe oraz nazwę hotelu.



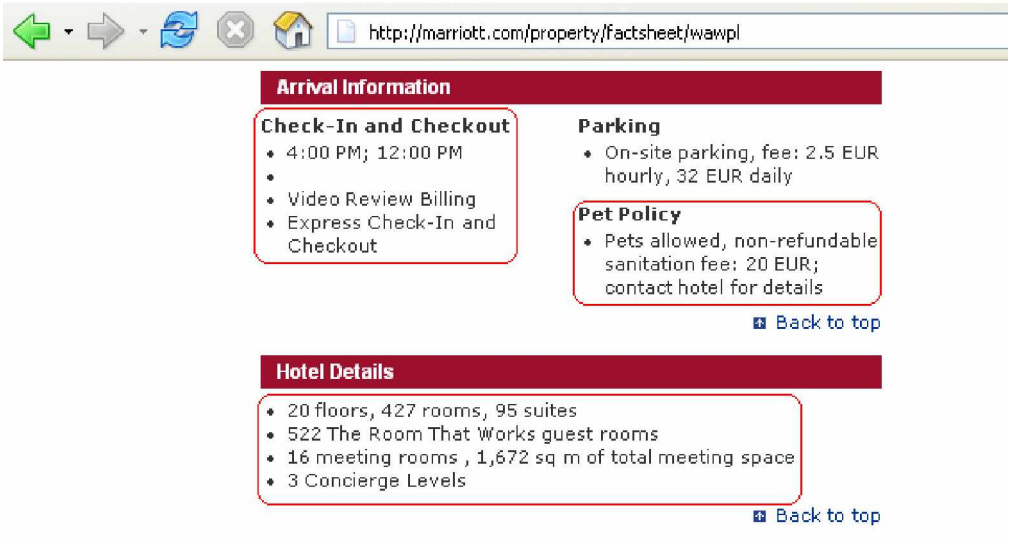
Rys. 19. Szczegóły hotelu Marriott w Warszawie (górna część strony WWW)

Z powyższego fragmentu strony WWW agent *MHWA* tworzy następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL">
  <j.0:address
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/Address" />
  <j.1:id>Marriott/WAWPL<j.0:id>
  <j.1:name>Warsaw Marriott Hotel</j.0:name>
  <j.0:phone>48 22 6306306</j.0:phone>
  <j.0:fax>48 22 8300041</j.0:fax>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/A
      ddress">
  <vcard:Country>Poland</vcard:Country>
  <vcard:Locality>Warsaw</vcard:Locality>
  <vcard:Pcode>00-697</vcard:Pcode>
  <vcard:Street>Al. Jerozolimskie 65/79</vcard:Street>
</rdf:Description>
```

Kolejnym krokiem jest zebranie informacji na temat zameldowania/wymeldowania do/z hotelu, polityki postępowania ze zwierzętami oraz szczegółowych informacji na temat

hotelu. Do tego celu wykorzystywany jest przedstawiony poniżej fragment tej samej strony (Rysunek 23).



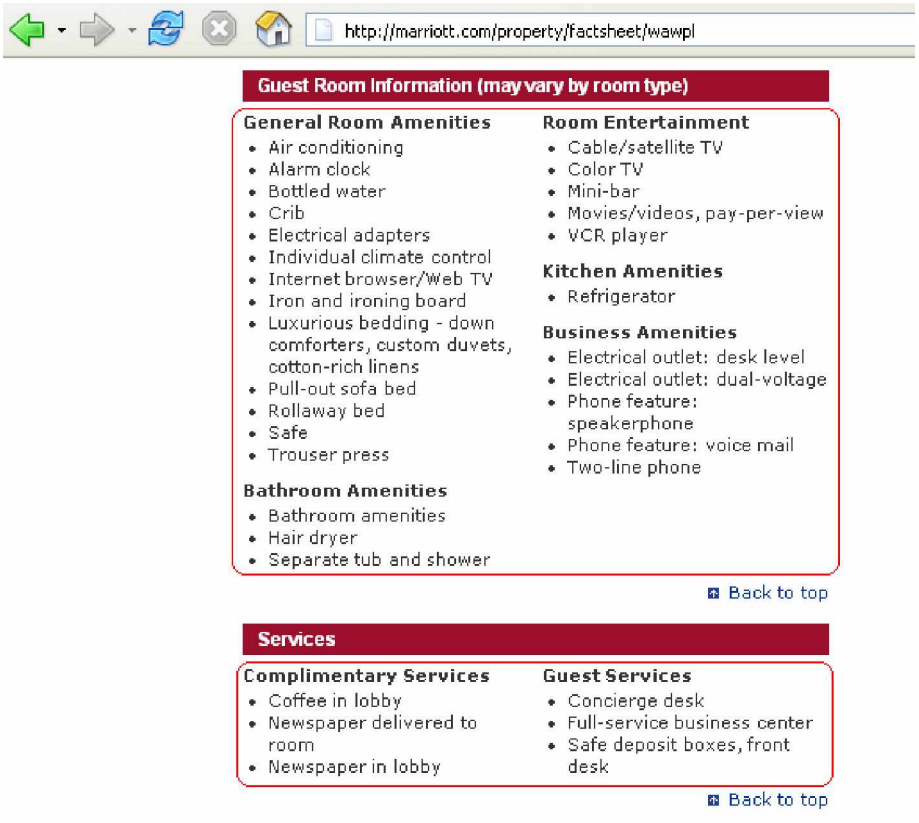
Rys. 20. Szczegóły hotelu Marriott w Warszawie (kolejna część strony WWW)

Z powyższego fragmentu strony WWW agent *MHWA* tworzy następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL">
  <j.1:petsPolicy rdf:resource="http://.../hotel.rdf#PetsAllowed"/>
  <j.1:additionalDetail
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/CheckIn-CheckOut"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/TotalRooms"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/Suites"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/Floors"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/TotalRoomsAndSuites"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/RoomsThatWork"/>
  <j.1:roomInfo
      rdf:resource="http://www.agentlab.net/travel/hotels/Marriott/WAWP
      L/ConciergeLevels"/>
</rdf:Description>
<rdf:Description
```

```
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL
      /CheckIn-CheckOut">
  <j.1:detail>Check-in: 4:00PM, Check-out: 12:00PM</j.1:detail>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/T
      otalRooms">
  <j.1:quantity>427</j.1:quantity>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/F
      loors">
  <j.1:quantity>20</j.1:quantity>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/S
      uites">
  <j.1:quantity>95</j.1:quantity>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/
      TotalRoomsAndSuites">
  <j.1:quantity>522</j.1:quantity>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/
      RoomsThatWork">
  <j.1:quantity>522</j.1:quantity>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL/C
      onciergeLevels">
  <j.1:quantity>3</j.1:quantity>
</rdf:Description>
```

Następnie – z tej samej strony – agent *MHWA* wyłuskuje szczegółowe informacje na temat pokoi gościnnych a także usług oferowanych przez hotel (Rysunek 24).



Rys. 21. Szczegóły hotelu Marriott w Warszawie (kolejna część strony WWW)

Z powyższego fragmentu strony WWW agent *MHWA* tworzy następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL">
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#AirConditioning"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#AlarmClock"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#BottledWater"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Cribs"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#ElectricalAdaptersAvailable"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#AirConditioningIndividuallyCon
      trolledInRoom"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#InternetAccess"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#InteractiveWebTV"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Iron"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#IroningBoard"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Safe"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#Trouser_PantPress"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#BathroomAmenities"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Hairdryer"/>
```

```
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#SeparateTubAndShower"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#CableTelevision"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#ColorTelevision"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Minibar"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#PayPerViewMoviesOnTV"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#VCRPlayer"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Refrigerator"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#DeskWithElectricalOutlet"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SpeakerPhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#VoiceMail"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Two-linePhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Newspaper"/>
  <j.1: businessServices rdf:resource="http://.../hotel.rdf#On-
      siteBusinessCenter"/>
</rdf:Description>
```

Ostatnia część wyłuskiwanych informacji dotyczy rekreacji (Rysunek 25).

Rys. 22. Szczegóły hotelu Marriott w Warszawie (kolejna część strony WWW)

Z powyższego fragmentu strony WWW agent *MHWA* tworzy następujące trójki RDF:

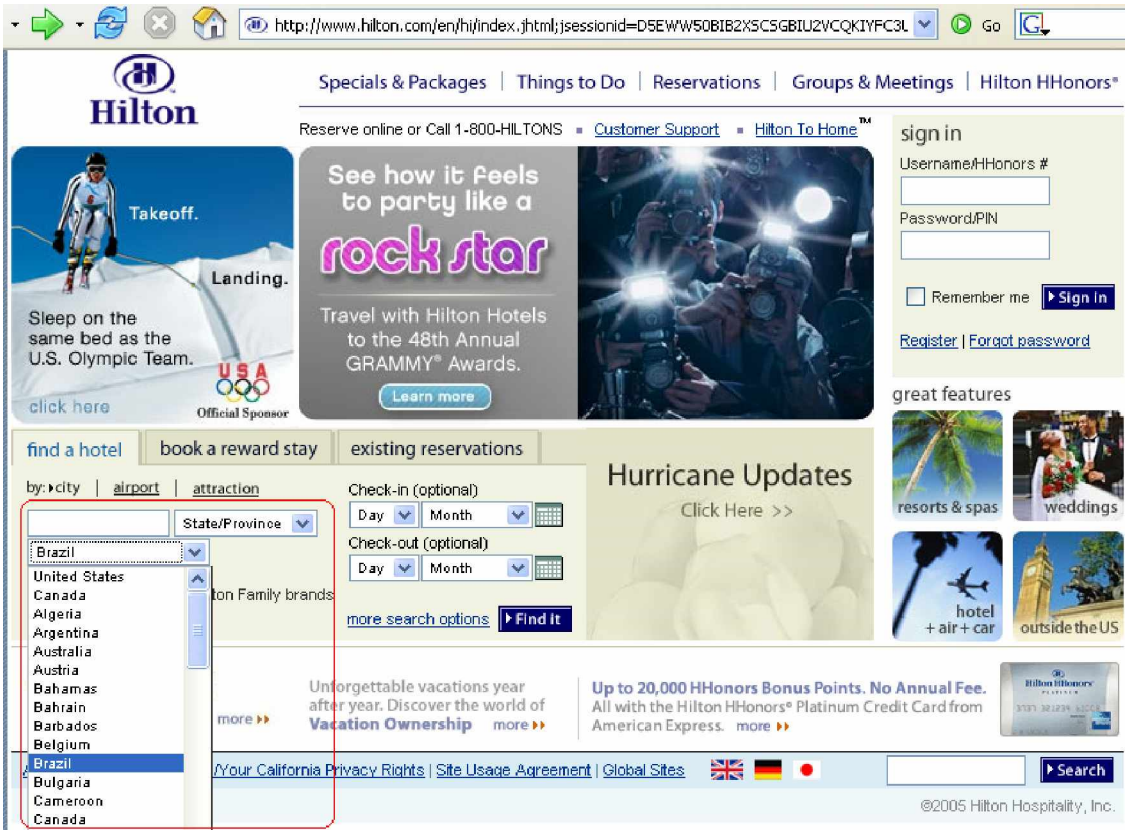
```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAWPL">
  <j.1: hotelAmenity rdf:resource="http://.../hotel.rdf#Pool"/>
  <j.1: hotelAmenity rdf:resource="http://.../hotel.rdf#Jacuzzi"/>
  <j.1: hotelAmenity
      rdf:resource="http://.../hotel.rdf#Whirlpool"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#BikeTrail"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#HorsebackRiding"/>
```

```
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#JoggingTrail"/>
  <j.1: recreationService rdf:resource="http://.../hotel.rdf#Sailing"/>
  <j.1: hotelAmenity rdf:resource="http://.../hotel.rdf#Sauna"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#SnowSkiing"/>
  <j.1: recreationService rdf:resource="http://.../hotel.rdf#Tennis"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#WaterSkiing"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#FitnessCenterOnsite"/>
  <j.1: recreationService
      rdf:resource="http://.../hotel.rdf#GolfLocationOnsite"/>
</rdf:Description>
```

5.2 Hilton Hotels Wrapper Agent

Agent *HiltonHotelsWrapperAgent* (*HHWA*) odpowiedzialny jest za zbieranie informacji na temat wszystkich hoteli Hilton na całym świecie. Agent ten wykorzystuje strony portalu Hilton: `http://www.hilton.com`.

Pierwszym zadaniem agenta *HHWA* jest znalezienie wszystkich państw i stanów (dla USA), gdzie występują hotele Hilton. Strona WWW znajdująca się na Rysunku 26 jest pierwszą stroną odwiedzaną przez *Wrapper-a* `http://www.hilton.com/en/hi/index.jhtml`.



Rys. 23. Hotele Hilton (strona startowa).

Fragment zaznaczony na czerwono zawiera listę państw oraz stanów (dla USA), gdzie znajdują się hotele Hitlon. *HHWA* wyłuskuje wszystkie kody państw oraz stanów z rozwijanych list a następnie odwiedza stronę prezentującą hotele z określonego regionu (państwa lub stanu). W celu odwiedzenia strony z hotelami z danego państwa, agent wstawia odpowiedni kod państwa w poniżej przedstawiony adres URL:

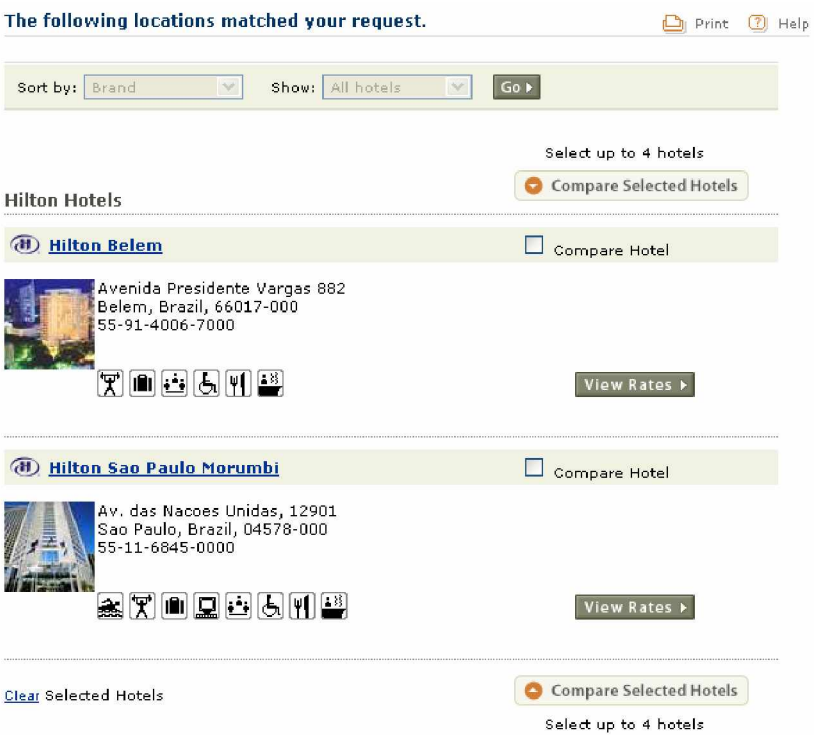
```
http://www.hilton.com/en/hi/hotels/search/index.jhtml?searchType=city&city=City&_D%3AstateDropBox=+&_D%3AcountryDropBox=&countrySelectName=countryDropBox&_D%3Abrandzzz=+&brandzzz=false&%2Fcom%2Fhilton%2Fsearch%2Fclient%2Fhandler%2FHandlerProxy.submit=true&_D%3A%2Fcom%2Fhilton%2Fsearch%2Fclient%2Fhandler%2FHandlerProxy.submit=+&_DARGS=%2Fen%2Fhi%2Fhotels%2Fsearch%2Fhome_tab_hotel_search.jhtml&stateDropBox=&countryDropBox=[kod państwa]
```

W przypadku hoteli z USA, agent wstawia odpowiedni kod stanu w następujący adres URL:

```
http://www.hilton.com/en/hi/hotels/search/index.jhtml?searchType=city&city=City&_D%3AstateDropBox=+&_D%3AcountryDropBox=&countrySelectName=countryDropBox&_D%3Abrandzzz=+&brandzzz=false&%2Fcom%2Fhilton%2Fsearch%2Fclient%2Fhandler%2FHandlerProxy.submit=true&_D%3A%2Fcom%2Fhilton%2Fsearch%2Fclient%2Fhandler%2FHandlerProxy.submit=&_DARGS=%2Fen%2Fhi%2Fhotels%
```

2Fsearch%2Fhome_tab_hotel_search.jhtml&countryDropBox=US&stateDropBox=**[kod stanu]**.

Zbudowanie wyżej przedstawionych adresów bazowych wymagało ode mnie użycia analizatora protokołów sieciowych Ethereal [Ethereal]. Okazuje się bowiem, iż po zdefiniowaniu reguł wyszukiwania i kliknięciu przycisku *szukaj* na stronie startowej, następuje przekierowanie do strony ze skryptem wyszukującym (adres tej strony nie jest widoczny w oknie przeglądarki). Z kolei strona ze skryptem wyszukującym po znalezieniu hoteli z danego państwa/stanu przekierowuje żądanie (HTTP Request) do strony prezentującej wyniki (adres tej stron jest widoczny w oknie przeglądarki). Na kolejnym rysunku przedstawiona jest fragment przykładowej strony z hotelami w Brazylii (Rysunek 27).



Rys. 24. Hotele Hilton w Brazylii

Mając stronę z hotelami w danym państwie/stanie, agent *HHWA* wyłuskuje kody hoteli a następnie odwiedza strony z detalami podstawiając ich kody do odpowiednich adresów

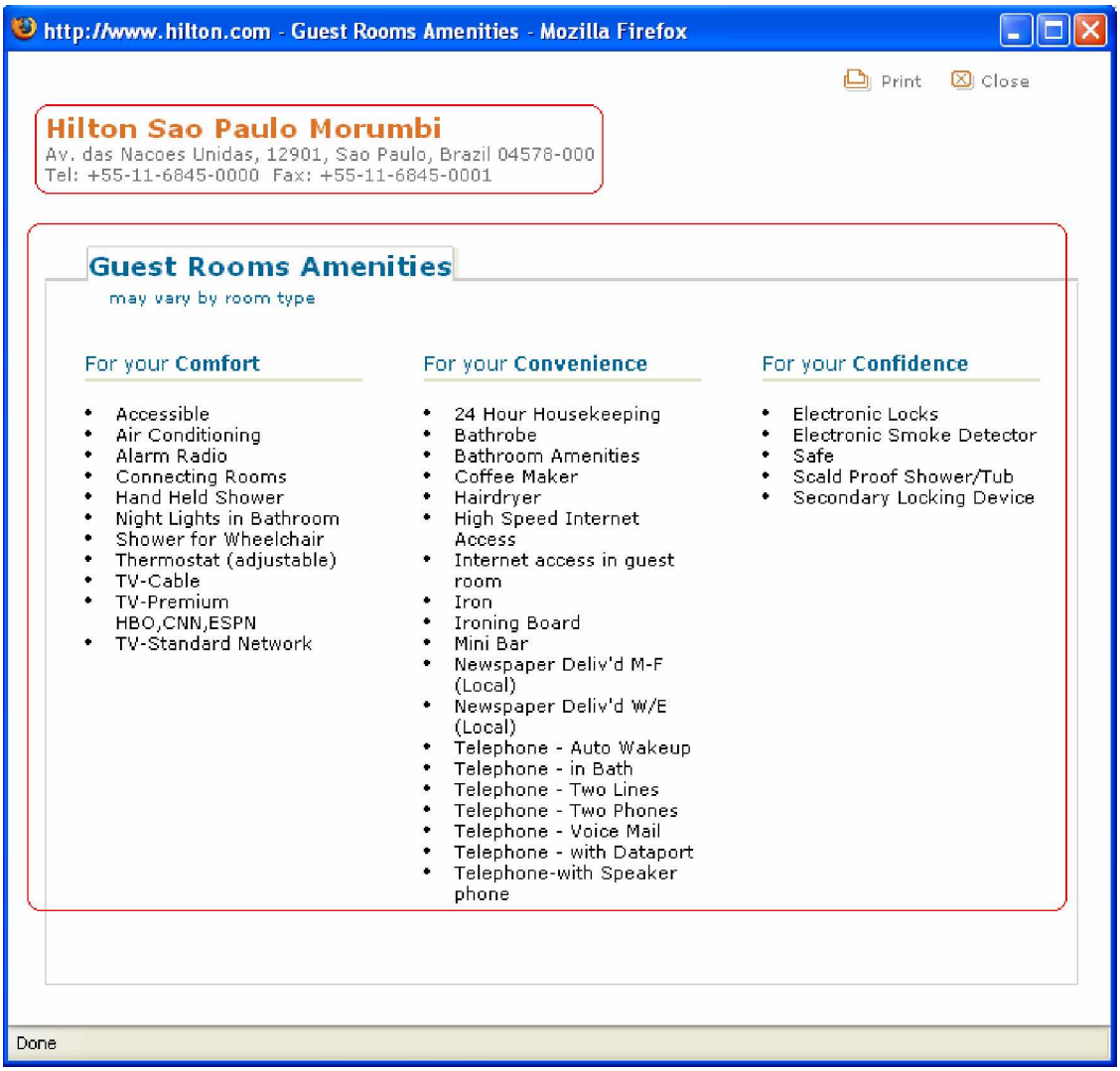
URL. W celu zebrania informacji na temat pojedynczego hotelu Hilton, agent *HHWA* odwiedza dwie strony:

1. Stronę zawierającą nazwę i adres hotelu oraz informacje o pokojach gościnnych. Jej bazowy adres URL to:

   `http://www.hilton.com/en/hi/hotels/accommodations_amenities_popup.jhtml?roomType=STD&ctyhocn=`**`[kod hotelu]`**
2. Stronę zawierającą informacje o typach rekreacji dostępnych w hotelu, na temat zameldowania/wymeldowania do/z hotelu oraz na temat polityki postępowania ze zwierzętami. Jej adres bazowy to:

   `http://www.hilton.com/en/hi/hotels/services.jhtml?ctyhocn=`**`[kod hotelu]`**

Przyjrzyjmy się zbieraniu danych przez agenta *HHWA* na przykładzie hotelu Hilton Sao Paulo Morumbi w Brazylii (jego kod to: SAOMOHI). Pierwsza strona, z interesującymi nas danymi, odwiedzana przez agenta *HHWA* przedstawiona jest na Rysunku 28.



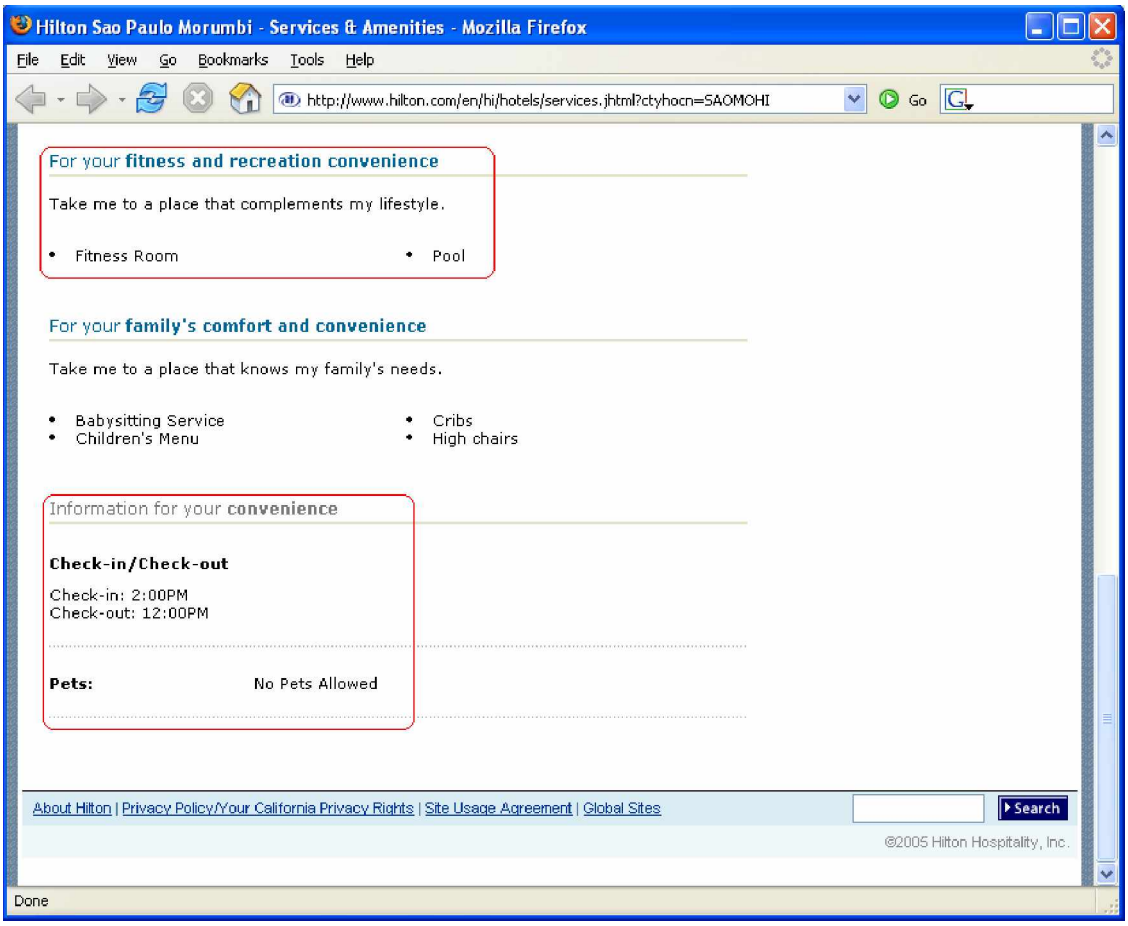
Rys. 25. Hotel Sao Paulo Morumbi w Brazylii

Jako rezultat transformacji informacji z tej strony, powstał następujący zbiór trójek RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI">
  <j.0:address
      rdf:resource="http://www.agentlab.net/travel/hotels/Hilton/SAOMOH
      I/Address" />
  <j.1:id>Hilton/SAOMOHI<j.0:id>
  <j.1:name>Hilton Sao Paulo Morumbi</j.0:name>
  <j.0:phone>+55-11-6845-0000</j.0:phone>
  <j.0:fax>+55-11-6845-0001</j.0:fax>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#AccessibleRoom"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#AirConditioning"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#ConnectingRooms"/>
```

```
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Shower"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#CableTelevision"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#CNNavailable"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Bathrobe"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#BathroomAmenities"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#Coffee_TeaMaker"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Hairdryer"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#HighSpeedInternetConnection"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#InternetAccess"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Iron"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#IroningBoard"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Minibar"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Newspaper"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Wake-upCalls"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Two-linePhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#VoiceMail"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#TelephoneWithDataPorts"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SpeakerPhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SmokeDetektors"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Safe"/>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI/A
ddress">
  <vcard:Country>Brazil</vcard:Country>
  <vcard:Locality>Sao Paulo</vcard:Locality>
  <vcard:Pcode>04578-000</vcard:Pcode>
  <vcard:Street>Av. das Nacoes Unidas, 12901</vcard:Street>
</rdf:Description>
```

Kolejna strona odwiedzana przez agenta *HHWA*, w celu zebrania informacji na temat hotelu Sao Paulo Morumbi, przedstawiona jest na Rysunku 29.



Rys. 26. Hotel Sao Paulo Morumbi w Brazylii

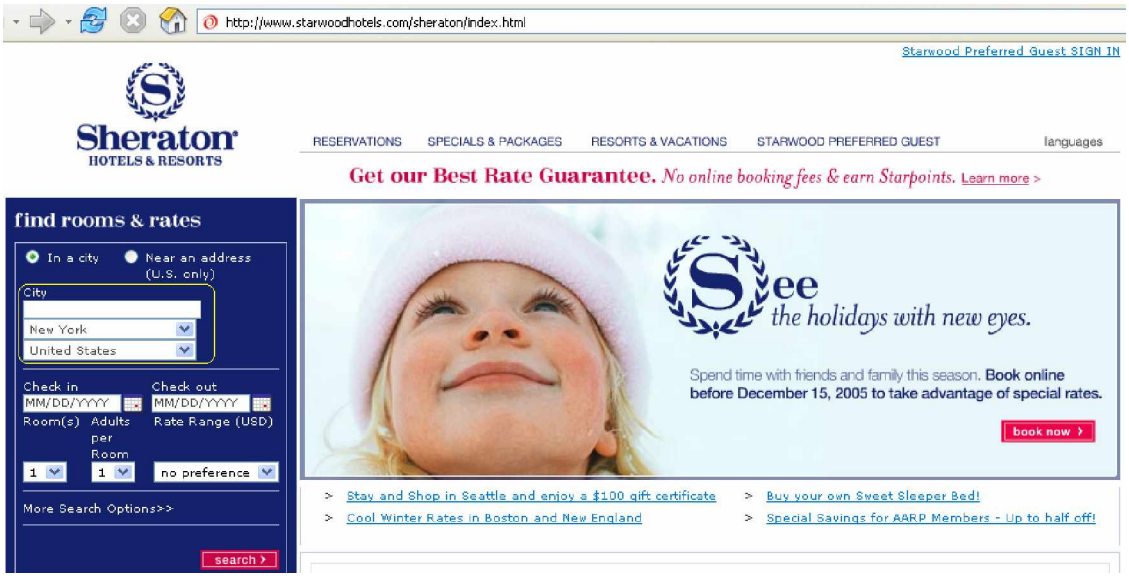
Poniżej przedstawione są wygenerowane trójki RDF na podstawie powyższej strony.

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI">
  <j.1:recreationService
      rdf:resource="http://.../hotel.rdf#FitnessCenterOnsite"/>
  <j.1:recreationService
      rdf:resource="http://.../hotel.rdf#IndoorOrOutdoorConnectingPool"
      />
  <j.1:petsPolicy rdf:resource="http://.../hotel.rdf#NoPetsAlowed"/>
  <j.1:additionalDetail
      rdf:resource="http://www.agentlab.net/travel/hotels/Hilton/SAOMOH
      I/CheckIn-CheckOut"/>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Hilton/SAOMOHI/C
      heckIn-CheckOut">
  <j.1:detail>Check-in: 2:00PM, Check-out: 12:00PM</j.1:detail>
</rdf:Description>
```

5.3 Starwood Hotels Wrapper Agent

Agent *StarwoodHotelsWrapperAgent* (*SHWA*) odpowiedzialny jest za gromadzenie informacji o hotelach należących do grupy Starwood Hotels and Resorts. Jako źródło danych dla agenta *SHWA* wykorzystywane są strony portalu `http://www.starwoodhotels.com`. W początkowym etapie implementacji agent ten zbierał informacje tylko o hotelach Sheraton (miał też inną nazwę: *SheratonHotelsWrapperAgent*). Okazało się jednak, iż wszystkie strony hoteli należących do grupy Starwood mają taki sam format i bez większych problemów nastąpiło przejście do gromadzenia danych na temat wszystkich hoteli tej grupy.

Stroną startową agenta jest `http://www.starwoodhotels.com/sheraton/index.html` (Rysunek 27). Adres WWW oraz wygląd tej strony jest trochę mylący i w pierwszym momencie wydawać by się mogło, iż znajdziemy tu tylko informacje na temat hoteli Sheraton. W rzeczywistości, rozpoczynając pracę od tej strony agent *SHWA* zbiera informacje na temat wszystkich hoteli z grupy.



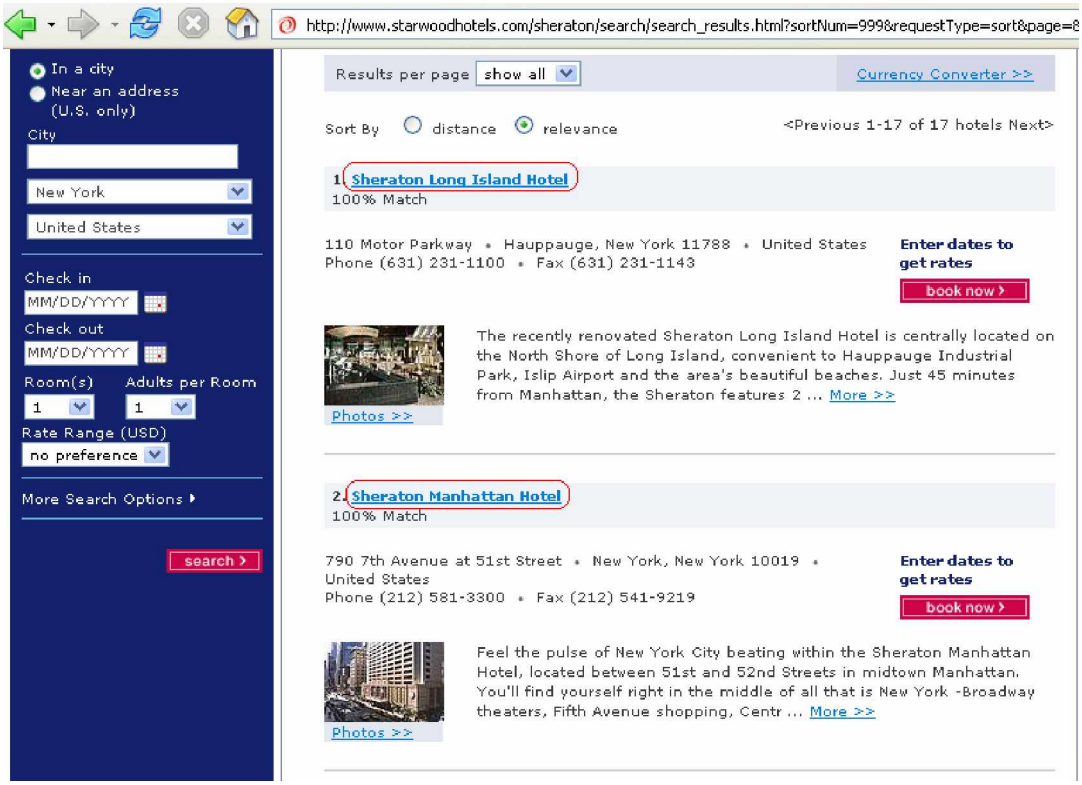
Rys. 27. Strona startowa agenta StarwoodHotelsWrapperAgent.

Podobnie jak w przypadku wcześniej opisanych agentów, pierwszym zadaniem agenta *SHWA* jest wyłuskanie wszystkich kodów państw oraz stanów (dla USA), w których

znajdują się hotele (z fragmentu zaznaczonego na żółto). Mając kody wszystkich lokalizacji hoteli, *SHWA* wyszukuje hotele dla każdej z nich. W tym celu agent odwiedza następujące strony:

1. Stronę wykonującą operacje wyszukiwania, jej bazowy adres to:
    a. `http://www.starwoodhotels.com/sheraton/go.search?searchType=city&country=`**`[kod państwa]`** – dla państw
    b. `http://www.starwoodhotels.com/sheraton/go.search?searchType=city&country=US&stateProvince=`**`[kod stanu]`** – dla stanów w USA
2. Stronę prezentującą wszystkie wyniki wyszukiwania, jej adres bazowy to:
    a. `http://www.starwoodhotels.com/sheraton/search/search_results.html?sortNum=999&requestType=sort&page=&sortType=relevance&country=`**`[kod państwa]`**– dla państw
    b. `http://www.starwoodhotels.com/sheraton/search/search_results.html?sortNum=999&requestType=sort&page=&sortType=relevance&country=US&stateProvince=`**`[kod stanu]`** – dla stanów USA

Fragment przykładowej strony prezentującej wyniki wyszukiwania hoteli z USA w stanie New York przedstawiony jest na kolejnym rysunku (Rysunek 28).



Rys. 28. Hotele należące do grupy Starwood Hotels & Resorts w Nowym Jorku

Mając stronę z wynikami wyszukiwania, agent *SHWA*, wyłuskuje kody hoteli (z fragmentów zaznaczonych na czerwono). W przypadku hoteli Starwood Hotels & Resorts kod składa się z trzech części. Wyznaczenie ciągów znaków, które jednoznacznie określają hotel odbywało się poprzez analizę adresów stron dla pojedynczych hoteli. Fragment kodu HTML zawierający kod hotelu Sheraton Long Island Hotel w Nowym Jorku wygląda następująco (części kodu hotelu zostały zaznaczone na niebiesko):

```
<div class="srInfo">
  <strong>1.
    <a href="javascript: void getDetails('855','sheraton','SI','SI')">
            Sheraton Long Island Hotel</a>
  </strong>
  <br />100% Match
</div>
```

Po wyznaczeniu kodów hoteli, agent *SHWA* odwiedza strony z detalami opisującymi hotele. Przyjrzyjmy się jakie strony i jakie informacje zbiera agent na przykładzie wyżej wymienionego hotelu w Nowym Jorku.

Agent rozpoczyna zbieranie informacji na temat hotelu od jego strony startowej, bazowy adres tej strony to:

`http://www.starwoodhotels.com/search/hotel_detail.html?propertyID=`**`[pierwsza część kodu hotelu]`**`& requestedChainCode=`**`[druga część kodu]`**`& requestedAffiliationCode=`**`[trzecia część kodu]`**`.`

Ze strony tej agent wyłuskuje nazwę hotelu, jego adres a także informacje na temat zameldowania/wymeldowania z/do hotelu (zaznaczone na niebiesko). Przykładowy kod HTML zawierający potrzebne dane wygląda następująco:

```
<div id="headlineContainer">
  <div id="headline">
    <h1 class="propName">Sheraton Long Island Hotel</h1>
  </div>
</div>
<div id="addressContainer">
  <div id="address">110 Motor Parkway
    <img src="/en_US/Media/Graphics/Brands/Sheraton/Images/sh_dot.gif"
      height="4" width="4" border="0" hspace="4" title="" alt=""/>
    Hauppauge, New York 11788
    <img src="/en_US/Media/Graphics/Brands/Sheraton/Images/sh_dot.gif"
      height="4" width="4" border="0" hspace="4" title="" alt=""/>
    United States
    <br/>
    Phone (631) 231-1100
    <img src="/en_US/Media/Graphics/Brands/Sheraton/Images/sh_dot.gif"
```

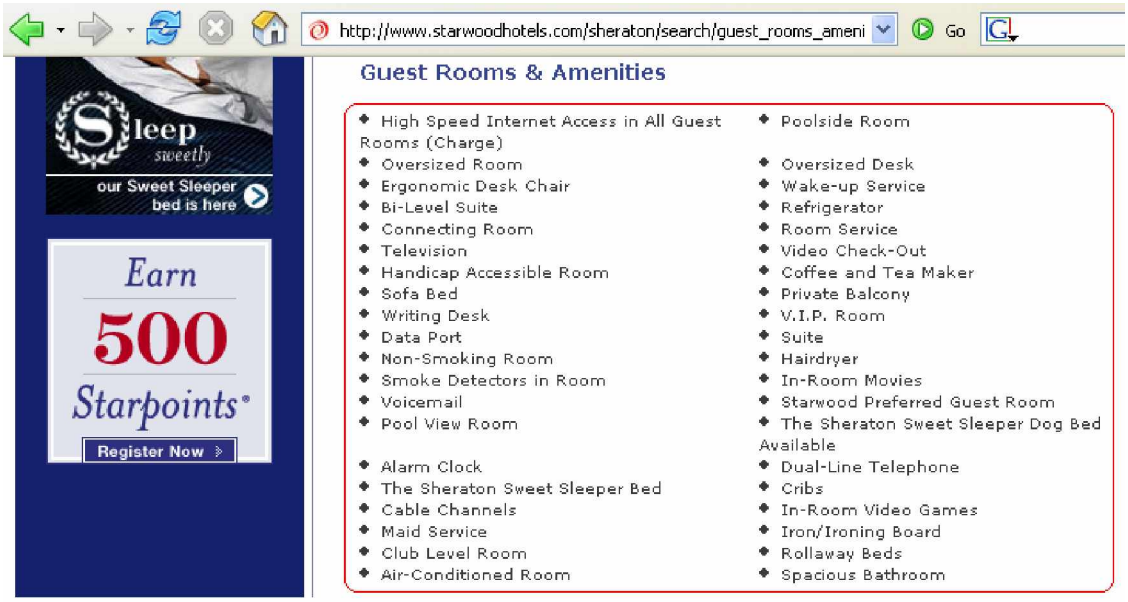
```
      height="4" width="4" border="0" hspace="4" title="" alt=""/>
    Fax (631) 231-1143
  </div>
</div>
…
<div id="checkInCheckOut">
  <b>CHECK IN 3:00 PM</b>
  <img src="/en_US/Media/Graphics/Brands/Sheraton/Images/sh_dot.gif"
      height="4" width="4" border="0" hspace="4" title="" alt=""/>
  <b>CHECK OUT 12:00 PM</b>
</div>
```

Po transformacji wyżej przedstawionego kodu HTML, agent produkuje następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Sheraton/855/SI/
      SI">
  <j.0:address
      rdf:resource="http://www.agentlab.net/travel/hotels/Sheraton/855/
      SI/SI/Address" />
  <j.1:additionalDetail
      rdf:resource="http://www.agentlab.net/travel/hotels/Sheraton/855/
      SI/SI/CheckIn-CheckOut"/>
  <j.1:id>Sheraton/855/SI/SI<j.0:id>
  <j.1:name>Sheraton Long Island Hotel</j.0:name>
  <j.0:phone>(631) 231-1100</j.0:phone>
  <j.0:fax>(631) 231-1143</j.0:fax>
</rdf:Description>
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/
      Sheraton/855/SI/SI/Address">
  <vcard:Country>United States</vcard:Country>
  <vcard:Locality>Hauppauge, New York</vcard:Locality>
  <vcard:Pcode>11788</vcard:Pcode>
  <vcard:Street>110 Motor Parkway</vcard:Street>
</rdf:Description>
<rdf:Description rdf:about="http://www.agentlab.net/travel/hotels/
      Sheraton/855/SI/SI/CheckIn-CheckOut">
  <j.1:detail>Check-in: 3:00PM, Check-out: 12:00PM</j.1:detail>
</rdf:Description>
```

Kolejna strona odwiedzana przez agenta zawiera szczegółowe informacje na temat pokoi gościnnych (Rysunek 29). Jej adres bazowy to:

`http://www.starwoodhotels.com/search/guest_rooms_amenities.html?propert yID=`**`[pierwsza część kodu hotelu]`**`& requestedChainCode=`**`[druga część kodu]`**`& requestedAffiliationCode=`**`[trzecia część kodu]`**`.`



Rys. 29. Hotel Sheraton – informacje na temat pokoi gościnnych

Na podstawie informacji prezentowanych na powyższej stronie WWW tworzone są następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Sheraton/855/SI/
      SI">
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#HighSpeedInternetConnection"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#ErgonomicChair"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#ColorTelevision"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#AccessibleRoom"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Desk"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#DataPort"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Non-smoking"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#SmokeDetectors"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#VoiceMail"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#AlarmClock"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#CableTelevision"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#AirConditioning"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Refrigerator"/>
  <j.1:roomAmenity
      rdf:resource="http://.../hotel.rdf#Coffee_TeaMaker"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Hairdryer"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Two-linePhone"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Cribs"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#Iron"/>
  <j.1:roomAmenity rdf:resource="http://.../hotel.rdf#IroningBoard"/>
</rdf:Description>
```

Ostatnia część wyłuskiwanych informacji dotyczy usług oferowanych przez hotel (Rysunek 30). Bazowy adres tej strony to:
`http://www.starwoodhotels.com/search/hotel_services.html?propertyID=`**`[pierwsza część kodu hotelu]`**`& requestedChainCode=`**`[druga część kodu]`**`& requestedAffiliationCode=`**`[trzecia część kodu]`**`.`

Rys. 30. Hotel Sheraton – informacje oferowanych usług

Trójki RDF wyprodukowane na podstawie powyższej strony to:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Sheraton/855/SI/
      SI">
  <j.1:hotelAmenity rdf:resource="http://.../hotel.rdf#IndoorPool"/>
  <j.1:recreationService
      rdf:resource="http://.../hotel.rdf#GolfLocationOffsite"/>
  <j.1:businessService rdf:resource="http://.../hotel.rdf#CarRental"/>
  <j.1:hotelAmenity
      rdf:resource="http://.../hotel.rdf#BellStaff_Porter"/>
  <j.1:hotelAmenity rdf:resource="http://.../hotel.rdf#Florist"/>
  <j.1:hotelAmenity rdf:resource="http://.../hotel.rdf#Sauna"/>
  <j.1:hotelAmenity rdf:resource="http://.../hotel.rdf#ConciergeDesk"/>
  <j.1:hotelAmenity
      rdf:resource="http://.../hotel.rdf#ValetDryCleaning"/>
  <j.1:hotelAmenity rdf:resource="http://.../hotel.rdf#WakeupService"/>
  <j.1:recreationService
      rdf:resource="http://.../hotel.rdf#FitnessCenterOnsite"/>
</rdf:Description>
```

6. Podsumowanie

W niniejszej pracy przedstawiłem ogólną architekturę agentowego *Systemu Wspomagania Podróży*, szczególną uwagę przywiązując do podsystemu gromadzenia informacji (*CSS*). Omówiłem wszystkich agentów systemu, przyglądając się bliżej agentom *CCS*: *Coordinator Agent*, *Wrapper Agent*, *Indexing Agent*. Rozważyłem możliwości ulepszenia podsystemu, między innymi, poprzez automatyczne generowanie *Wrapper-ów*, stworzenie nowego typu agentów dostarczających dane do systemu: *Focused Crawler Agent* czy w końcu dodanie agenta typu *Logging Agent*. Omówiłem sposób implementacji systemu, jego testowania a także zaprezentowałem narzędzia wykorzystane do tych celów. W rozdziale piątym przedstawiłem przykład działania *Wrapper-ów* zbierających dane na temat hoteli Marriott, Hilton oraz hoteli z grupy Starwood Hotels and Resorts.

7. Bibliografia

ACL, Agent Communication Language FIPA ACL Message Structure Specification.... 16

Airport Ontology, http://www.daml.org/2001/10/html/airport-ont..... 13

Angryk R., Galant V., Gordon M., Paprzycki M. (2002): Travel Support System - an Agent Based Framework. In: H. R. Arabnia, Y. Mun (eds.), Proceedings of the International Conference on Internet Computing (IC'02), CSREA Press, Las Vegas, NV 719-725, 2002..... 21

Ashish Naveen, Knoblock Craig; Semi-automaticWrapper Generation for Internet Information Sources; In Proceedings of the Second IFCIS International Conference on Cooperative Information Systems, Kiawah Island, SC, 1997..... 41

Berners-Lee T., Hendler J., Lassila O., (2001): The Semantic Web, Scientific American, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21..... 8

Bradshaw Shannon, Menczer Filippo, Pant Gautam; Search Engine-Crawler Symbiosis: Adapting to Community Interests; Proc. 7th European Conference on Research and Advance d Technology for Digital Libraries (ECDL 2003)..... 41

Chau Michael, Chen Hsinchun; Personalized and Focused Web Spiders; in N. Zhong, J. Liu, Y. Yao (Eds), Web Intelligence, Springer-Verlag, February 2003, pp. 197-217.. 41

Contract Net Protocol, FIPA Contract Net Interaction Protocol Specification..... 16, 39

DOM, Document Object Model: http://www.w3.org/DOM/..... 43

Ehrig Marc; Ontology-Focused Crawling of Documents and Relational Metadata; Master's Thesis..... 41

Ethereal, http://www.ethereal.com/..... 55

FIPA, Foundation for Inteligent Physical Agents http://www.fipa.org/..... 15

Galant V., Gordon M., Paprzycki M. (2002a): Knowledge Management in an Internet Travel Support System. B. Wiszniewski (ed.), Proceedings of ECON2002, ACTEN, Wejcherowo, 97-104, 2002..... 21

Galant V., Gordon M., Paprzycki M. (2002b): Agent-Client Interaction in a Web-based E-commerce System. D. Grigoras (ed.), Proceedings of the International Symposium

on Parallel and Distributed Computing, University of Iaşi Press, Iaşi, Romania, 2002b, 1-10 22
Gawinecki M., Gordon M., Nguyen N., Paprzycki M., Szymczak M. (2005b): RDF Demarcated Resources in an Agent Based Travel Support System. In: Proceedings of the 17th Mountain Conference of the Polish Information Society..... 23, 25
Gawinecki M., Gordon M., Paprzycki M., Szymczak M., Vetulani Z., Wright J. (2005a): Enabling Semantic Referencing of Selected Travel Related Resources. In: W. Abramowicz, Proceedings of the BIS'2005 Conference, Poznań University of Economics Press, Poznań, Poland, (2005) 271-290..... 25
Gordon M., Kowalski A., Paprzycki N., Pełech T., Szymczak M., Wasowicz T. (2005b) Ontologies in a Travel Support System. In: Proceedings of the Internet 2005 Conference..... 23
Gordon M., Paprzycki M. (2005a): Designing Agent Based Travel Support System, in: Proceedings of the ISPDC 2005 Conference, IEEE Computer Society Press, Los Alamitos, CA, 207-214, 2005..... 21
HTML Parser, CyberNeko HTML Parser: http://people.apache.org/~andyc/neko/doc/html/index.html..... 43
HTML, http://www.w3.org/MarkUp/..... 7
HTTP Client: http://jakarta.apache.org/commons/httpclient/..... 43
IATA, International Air Transport Association, http://www.iata.org..... 13
ICAO, International Civil Aviation Authority, http://www.icao.int..... 13
JADE, Java Agent DEvelopment Framework, http://jade.cselt.it..... 18
Java, http://java.sun.com..... 18
Jaxen: http://jaxen.org/..... 43
JENA, http://www.hpl.hp.com/semweb/jena.htm..... 20
Jennings N. R. (2001): An agent-based approach for building complex software systems. CACM 44, 4, 2001, 35-41..... 23
JUnit, www.junit.org..... 45
Kaczmarek P., Gordon M., Paprzycki M., Gawinecki M. (2005): The Problem of Agent-Client Communication on the Internet. Scalable Computing: Practice and Experience, 6(1), 2005, 111-123..... 22

Knoblock Craig, Lerman Kristina, Minton Steven, Muslea Ion; Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach; IEEE Data Engineering Bulletin, 23(4):33--41, December 2000 41
Lerman Kristina, Gazen Cenk, Minton Steven, Knoblock Craig; 2004a; Populating the SemanticWeb; In Proceedings of the AAAI 2004 Workshop on Advances in Text Extraction and Mining, 2004 41
Lerman Kristina, Getoor Lise, Minton Steven, Knoblock Craig; 2004b; Using the Structure of Web Sites for Automatic Segmentation of Tables; In Proceedings of ACM SIG on Management of Data (SIGMOD-2004), 2004b 41
LOG4J: http://logging.apache.org/log4j/docs/index.html 43
Maes P.; Agents that Reduce Work and Information Overload; Communications of the ACM, 37, 7, (1994) 31-40 4
Muslea Ion, Minton Steven, Knoblock Craig; Hierarchical Wrapper Induction for Semistructured Information Sources; Autonomous Agents and Multi-Agent Systems, 4(1/2), March 2001 41
MySQL, www.mysql.com 43
Novak Blaž; A Survey of Focused Web Crawling Algorithms; Conference on Data Mining and Warehouses (SiKDD 2004) 41
OWL, Ontology Web Language, http://www.w3.org/TR/owl-features/ 12
Pant Gautam, Srinivasan Padmini, Menczer Filippo; Crawling the Web; in M. Levene and A. Poulovassilis, editors: Web Dynamics, Springer-Verlag, 2003 41
Paprzycki Marcin, *Agenci programowi jako metodologia tworzenia oprogramowania*, w: Z. Huzar, Z. Mazur (ed.), Problemy i Metody Inżynierii Oprogramowania, WNT, Warszawa, 2003 15
Pisarek Sz., Paprzycki M. (2005) Internetowy System Wspomagania Podróży - Tworzenie Podsystemu Odpowiedzialnego za Gromadzenie Danych. In: Proceedings of the 17th Mountain Conference of the Polish Information Society, Szczyrk, Poland31
RDF Model and Syntax, http://www.w3.org/RDF/Group/WD-rdf-syntax/ 11
RDF Schema, http://www.w3.org/TR/rdf-schema/ 12
RDF, http://www.w3.org/RDF/ 9
RDQL, http://www.w3.org/Submission/RDQL/ 20