Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

Rafał Gąsiorowski
nr albumu 177309

Zastosowanie ontologii do organizacji informacji pozyskiwanych z Internetu

Praca magisterska na kierunku: Informatyka

Promotor: Prof. Marcin Paprzycki

Warszawa 2006

Streszczenie

Praca opisuje gromadzenie informacji pochodzących z wielu źródeł w Internecie z zastosowaniem ontologii do ich katalogowania. Przedstawiona jest metoda łączenia danych zapisanych w `RDF` z uwzględnieniem rozwiązywania problemów wynikających z niekompletności i niezgodności informacji. Zbieranie i porządkowanie informacji jest wykonywane przez system wieloagentowy.

Słowa kluczowe

- Ontologie
- Sieć semantyczna
- Katalogowanie informacji
- System wieloagentowy
- Planowanie podróży

Spis treści

1 Wstęp **5**
1.1 Przeciążenie informacją 5
1.1.1 Czym jest przeciążenie informacją? 5
1.1.2 Lokalizacja informacji w Internecie 5
1.1.3 Reprezentacja informacji w Internecie 7

2 Sieć semantyczna **8**
2.1 Co to jest Sieć Semantyczna? 8
2.2 Warstwy Sieci Semantycznej 8
2.3 Resource Description Framework 9
2.3.1 Zasoby 9
2.3.2 Stwierdzenia 10
2.3.3 Zapis RDF w XML 11
2.3.4 Typowane literały 13
2.3.5 Puste węzły 14
2.3.6 Kontenery 15
2.3.7 Kolekcje 16
2.3.8 Reifikacje 18
2.4 Schematy RDF 20
2.4.1 Klasy i podklasy w RDFS 20
2.4.2 Właściwości 22
2.5 RDF Query Language 23
2.6 Ontologie 24
2.7 Jena 26

3 Systemy wieloagentowe **26**
3.1 Czym jest agent? 26
3.2 Standardy FIPA 28
3.3 Java Agent Development Framework 30
3.3.1 Realizacja agenta w JADE 30
3.3.2 Komunikacja między agentami 33

4 System wspomagania planowania podróży **33**
4.1 Podsystemy PTIS 34
4.1.1 Podsystem pozyskiwania informacji 35
4.1.2 Podsystem przechowywania informacji 35
4.1.3 Podsystem zarządzania informacjami 36
4.1.4 Podsystem dostarczania informacji 36

5 Ontologiczny system katalogowania informacji **36**
5.1 Reprezentacja danych 36
5.2 Agenci działający w podsystemie OntoStorage 37
5.2.1 Agent bazodanowy (DBA) 38
5.2.2 Agent Nadzorca Spójności Danych (DSA) 40
5.2.3 Agent Indeksujący (IA) 42
5.2.4 Agent Koordynator Wyszukiwania Danych (SCA) . . . 48
5.2.5 Agent Dostawca Danych (WA) 50
5.2.6 Główny Agent Nadzorca (MA) 51
5.3 Korzystanie z systemu 51
5.3.1 Plik konfiguracyjny 51
5.3.2 Interfejs użytkownika 52
5.4 Podsumowanie 53

6 Bibliografia **55**

1 Wstęp

1.1 Przeciążenie informacją

1.1.1 Czym jest przeciążenie informacją?

Internet jest olbrzymią kopalnią wiedzy na praktycznie dowolny temat, liczba stron internetowych przekroczyła już dawno 10 miliardów i ciągle wzrasta. Ze względu na architekturę Sieci dostęp do każdej z tych stron jest niemalże natychmiastowy. Internet stanowi więc olbrzymią zbiornicę nieuporządkowanej informacji. To właśnie ów brak uporządkowania stanowi jedną z jego najistotniejszych wad. Już w początkowym okresie powstawania Internetu problemem stało się bowiem wyłowienie z morza danych tych istotnych informacji, które są związane z tematem interesującym użytkownika. Ilość źródeł danych jest tak duża, że użytkownik nie jest w stanie samodzielnie zapoznać się z każdym z nich, aby sprawdzić użyteczność dostarczanych przez nie informacji i oddzielić istotne fakty (sygnał) od błędnych lub nieistotnych z jego punktu widzenia (szum). Taka sytuacja jest określana mianem przeciążenia informacją (ang. *information overload*). Termin ten został użyty po raz pierwszy przez socjologia i futurologa, Alvina Tofflera w książce "Szok przyszłości"[52]. Wprawdzie określenie to pierwotnie nie odnosiło się do Internetu (zostało po raz pierwszy użyte w roku 1970, a więc praktycznie rzecz biorąc przed jego powstaniem) ale w obecnych czasach pasuje do tego medium jak do żadnego innego.

1.1.2 Lokalizacja informacji w Internecie

Aby w pełni skorzystać z zasobów, jakie znajdują się w Internecie użytkownik musi zdać się na usługi polegające na lokalizacji informacji. Potrzebuje przewodnika, któremu mógłby zaufać i zdać się na jego rekomendacje. Obecnie istnieją dwa główne mechanizmy wspomagania wydobywania informacji z Internetu - katalogi i wyszukiwarki. Katalogi są tworzone ręcznie przez ludzi (lub pół-automatycznie na podstawie zgłoszeń stron nadsyłanych przez ich właścicieli) i są dobre, gdy szukamy informacji bardzo ogólnych choć dobrze określonych jeśli chodzi o tematykę, albowiem wykorzystując katalogi łatwo zwykle dotrzeć do serwisów tematycznych. Niestety w takim katalogu znalezienie informacji bardziej szczegółowych jest praktycznie niemożliwe.

Można oczywiście skorzystać z chyba najbardziej popularnego obok poczty elektronicznej narzędzia Internetowego - wyszukiwarki. Już tylko powierzchowne zapoznanie się z jakością otrzymanych rezultatów pokazuje, że w bardzo wielu przypadkach dobranie listy słów kluczowych dających pożądane

wyniki jest niezwykle trudne. Jeżeli będzie ich zbyt mało i będą zbyt ogólne to rezultatem będzie bardzo duża lista odnośników nie mających wiele wspólnego z poszukiwaną informacją. Natomiast zapytania zbyt szczegółowe mogą niekiedy doprowadzić do stanu, w którym nie otrzymamy żadnych wyników lub zaczną się pojawiać strony nie związane z interesującym nas tematem. Dzieje się tak dlatego, że niewyspecjalizowane, ogólne wyszukiwarki wciąż bazują na indeksowaniu ciągów znaków i ich porównywaniu z ciągami wprowadzonymi w formularzu przez użytkownika. Nie wiedzą one nic o znaczeniu tych słów i ich wzajemnych powiązaniach. Poza tym w większości przypadków na miejsce na liście wyników ma również wpływ popularność strony rozumiana jako liczba odnośników prowadzących do niej z innych serwisów (nie mówiąc już o istniejącej w wielu serwisach możliwości zakupu wysokiej pozycji pomiędzy wynikami wyszukiwania). Tak więc, w typowych wyszukiwarkach Internetowych brane są pod uwagę preferencje ogółu użytkowników Sieci a nie konkretnej osoby zlecającej poszukiwania. Wprawdzie bardzo często preferencje jednostki zbliżone są do preferencji ogółu, ale oznacza to, że użytkownik otrzyma wyniki "uśrednione", które mogą, ale nie muszą zawierać interesujące go informacje.

Istnieją także wyszukiwarki wyspecjalizowane, bazujące przedstawiane użytkownikowi wyniki na wewnętrznych bazach danych i dostarczające tylko informacji na pewne wybrane tematy. Pojawiają się w nich wyspecjalizowane formularze, kryteria filtrowania i sortowania wyników umożliwiające bardziej precyzyjne konstruowanie zapytań i pewne formy przystosowywania rezultatów wyszukiwania do preferencji użytkowników. Wadą takich wyszukiwarek jest natomiast ograniczona ilość i zakres danych, które są poddawane przeszukiwaniu. Twórcy takiego serwisu muszą bowiem najpierw opracować własną strukturalizację informacji, najczęściej w formie relacyjnej bazy danych. Katalogowanie informacji w taki sposób sprawia, że komunikacja między źródłami danych jest mocno utrudniona (jeśli nie niemożliwa), albowiem każdy dostawca wyszukiwarki dziedzinowej ma własny, charakterystyczny format danych. W związku z tym konieczne i czasami zachodzące procesy rozszerzania modeli czy też ich łączenia są niezwykle żmudne. Ponadto nadal nie mamy do czynienia z personalizacją wyników, a jedynie z ich uporządkowaniem według kryteriów opracowanych wstępnie w ramach strukturalizacji rzeczywistości dokonanej przez autorów wyszukiwarki tematycznej, a następnie wykorzystanych przez użytkownika dla odpytania danej wyszukiwarki.

1.1.3 Reprezentacja informacji w Internecie

Można powiedzieć, że główną przeszkodą dla zautomatyzowanego pozyskiwania i indeksowania informacji jest najpopularniejszy w Internecie format opisu i prezentacji danych czyli HTML (ang. *HyperText Markup Language*). W tym kontekście największym grzechem HTML jest przemieszanie w sposób niemal dowolny samych danych z opisem ich prezentacji. Jedynie opcjonalne metadane umieszczane w znacznikach `<meta>` są opisane w sposób dość formalny. Jednak także tutaj ich postać jest zależna od twórcy strony i ich zgodność z zawartością dokumentu jest często problematyczna (np. na liście słów kluczowych pojawiają się pojęcia niezwiązane z treścią dokumentu, natomiast bardzo popularne - co za tym idzie zwiększające potencjalną popularność strony, podobnie dane o autorze mogą być błędne, etc.). W tej sytuacji analiza zawartości dokumentu może się opierać w zasadzie tylko na badaniu częstości występowania danych wyrazów, czy też ich umiejscowienia (można bowiem przypuszczać, że wystąpienie wyrazu w tytule strony lub nagłówku tabeli zwiększa prawdopodobieństwo, że dokument jest związany z pożądaną tematyką). Jednakże, także i w tym przypadku oprogramowanie indeksujące nie wie nic o kontekście w jakim dane słowo zostało użyte w zdaniu a więc o jego prawdziwym znaczeniu.

Ekstrakcja szczegółowych informacji jest więc bardzo trudna, dla każdej strony trzeba bowiem tworzyć parser niemal od zera. Trzeba poznać strukturę formatowania dokumentu (przy optymistycznym założeniu, że jest ona stała dla wszystkich podstron w serwisie) i samodzielnie oddzielić opis wyglądu od samych danych.

Od kilku lat można obserwować odchodzenie od stosowania czystego HTML. W zamian za to powoli wprowadzana jest zasada oddzielenia modelu danych od opisu metod wizualizacji. Jest to możliwe dzięki zastosowaniu dokumentów zapisanych w XML (ang. *eXtensible Markup Language*). XML pozwala na tworzenie własnego języka znaczników, opisywanego przez definicję typu dokumentu DTD (ang. *Document Type Definition*). Taka specyfikacja może być również tworzona przy użyciu schematów XML (ang. *XML Schema Definition*, XSD), których zaletą jest to, że same są też dokumentami XML. Proces wizualizacji dokumentu XML może być natomiast realizowany przez wiele różnych technologii takich jak kaskadowe arkusze stylów (*Cascading Style Sheets*, CSS) pozwalające na tworzenie dobrze sformatowanych stron HTML czy XSL (ang. *eXtensible Stylesheet Language*) umożliwiający m.in. konwersję XML do PDF.

Mając dokument XML i jego definicję oprogramowanie może dokonać weryfikacji, czy dokument jest dobrze sformatowany i zbudowany zgodnie z

określonymi zasadami. Dzięki standardowej formie zapisu dokumentu XML aplikacja może łatwo dokonać ekstrakcji jego zawartości. Nadal jednak pozostaje problem znaczenia pozyskanych danych.

Aby znaleźć odpowiedź, od pewnego czasu coraz większego znaczenia nabiera projekt Sieci Semantycznej (ang. *Semantic Web).*

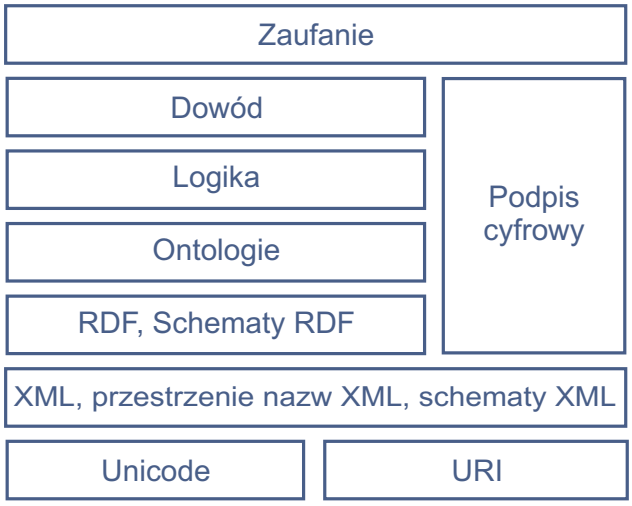
2 Sieć semantyczna

2.1 Co to jest Sieć Semantyczna?

Projekt Sieci Semantycznej (ang. *Semantic Web*, SW) został zainicjowany przez Tima Bernersa-Lee, jednego z najważniejszych twórców WWW, HTTP i HTML. Podstawową ideą tego projektu jest zestaw standardów i technologii wprowadzający strukturę semantyczną do Sieci. Taka struktura ma wspomóc komputery w lepszym wykorzystaniu informacji zgromadzonej w Internecie. Znacznie bardziej usystematyzowane i sformalizowane dane mają pozwalać na powstanie globalnej sieci powiązanych informacji. Taka sieć będzie mogła być wykorzystywana w prosty sposób przez odpowiednio dostosowane oprogramowanie. Użycie danych zapisanych w formatach opartych o XML, a równocześnie opartych o ustandaryzowane słownictwo zwiększy ich dostępność dla oprogramowania, ułatwi przepływ informacji między systemami i pozwoli na zautomatyzowanie wielu zadań, które obecnie są niemożliwe do zrealizowania przez oprogramowanie lub ich działanie odbiega od oczekiwań użytkowników.

2.2 Warstwy Sieci Semantycznej

Model Sieci semantycznej można podzielić na kilka warstw uporządkowanych w sposób przestawiony na rysunku 1, pochodzącym z artykułu [7]. Dolne warstwy opierają się na sprawdzonych i stosowanych już powszechnie technologiach, co ma ułatwić wprowadzenie do Internetu rozwiązań z wyższych warstw i pozwolić na wykorzystanie istniejącego już oprogramowania. Zastosowany tutaj unikod upraszcza reprezentację informacji zapisanych w wielu językach. Z kolei swobodna wymiana danych między różnymi aplikacjami wymaga stosowania uniwersalnych, unikalnych identyfikatorów - ich rolę pełnią URI (ang. *Uniform Resource Identifier*). Technologia ta wykorzystywana jest przez XML i oparty na nim RDF (ang. *Resource Description Framework*), podstawowy format danych dla sieci semantycznej, który zostanie szczegółowo omówiony w rozdziale 2.3. Ontologie umożliwiają opisywanie w sposób zrozumiały dla oprogramowania pojęć abstrakcyjnych i charakte-



Rysunek 1: Warstwy Sieci Semantycznej

rystyk obiektów ze świata rzeczywistego oraz specyfikowanie relacji między nimi. Zostały one szerzej opisane w rozdziale 2.6.

Sieć semantyczna w zamyśle jej pomysłodawców ma stać się globalną bazą danych. Pojawiają się więc kwestie związane z bezpieczeństwem i pytanie, czy można polegać na pozyskanych z Internetu informacjach - nie do wszystkich źródeł danych można przecież mieć pełne zaufanie. Docelowo zapytania do aplikacji wykorzystujących Sieć Semantyczną będą mogły zwrócić nie tylko wyniki wyszukiwania ale również "dowód" jak dana informacja została uzyskana. Dzięki podpisowi cyfrowemu można będzie zweryfikować, kto jest autorem danej informacji, co z kolei pozwoli na ocenianie wiarygodności źródeł danych jak i gradację stopnia zaufania do nich. Należy tutaj zaznaczyć, że wyższe warstwy sieci semantycznej są nadal głównie w fazie badawczej.

2.3 Resource Description Framework

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

2.3.1 Zasoby

W dosłownym tłumaczeniu RDF jest środowiskiem do opisu zasobów. Zacząć trzeba więc od wyjaśnienia co jest rozumiane przez jego twórców pod

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

2.3.2 Stwierdzenia

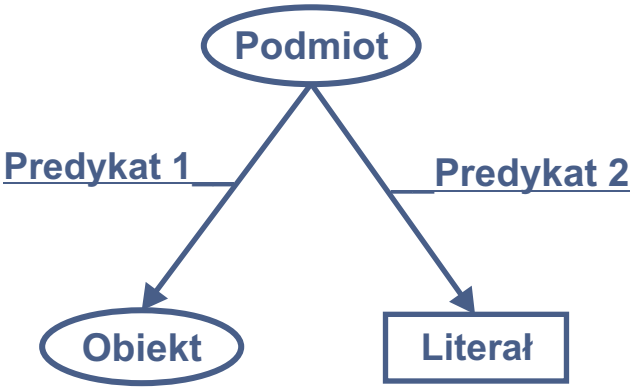
Do opisu zasobów używa się tzw. stwierdzeń (ang. *statements*). Często zamiennie z terminami zdanie i stwierdzenie używane jest jeszcze określenie trójka (ang. *triple*). To ostatnie określenie wynika z faktu, że każde stwierdzenie w RDF ma strukturę podobną do prostego zdania w języku naturalnym i składa się z trzech elementów:

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

2. **Orzeczenia (predykatu)** (ang. *predicate*). Orzeczenie jest nazwą cechy lub relacji dotyczącej opisywanego podmiotu i podobnie jak podmiot jest reprezentowane przez URI.

3. **Obiektu** (ang. *object*). Obiektem może być inny zasób (reprezentowany przez URI) lub stała wartość określana mianem **literału** (ang. *literal*). Najczęściej mamy do czynienia z jedną z dwóch sytuacji - opisujemy relację między dwoma zasobami (wtedy obiektem jest URI reprezentujące zasób) lub mówimy, że podmiot ma jakąś cechę o zadanej wartości (wtedy używamy literału). Literał reprezentować może dowolną wartość o ile daje się ona zapisać w postaci ciągu znaków. W RDF literałami mogą być tylko obiekty, nie jest możliwe ich użycie w charakterze podmiotu czy orzeczenia.

Każde zdanie można uważać za etykietowany graf skierowany (rysunek 2). Podmiot i obiekt są węzłami takiego grafu, natomiast orzeczenie jest reprezentowane jako krawędź skierowana od węzła podmiotu do węzła obiektu i



Rysunek 2: Graf przedstawiający dwa zdania RDF

etykietowana przy pomocy URI orzeczenia. Przy rysowaniu grafów wizualizujących dane zapisane w RDF przyjęło się używanie owali do oznaczania węzłów, które są zasobami (są reprezentowane przez URI). Obiekty będące literałami są natomiast reprezentowane przez prostokąty.

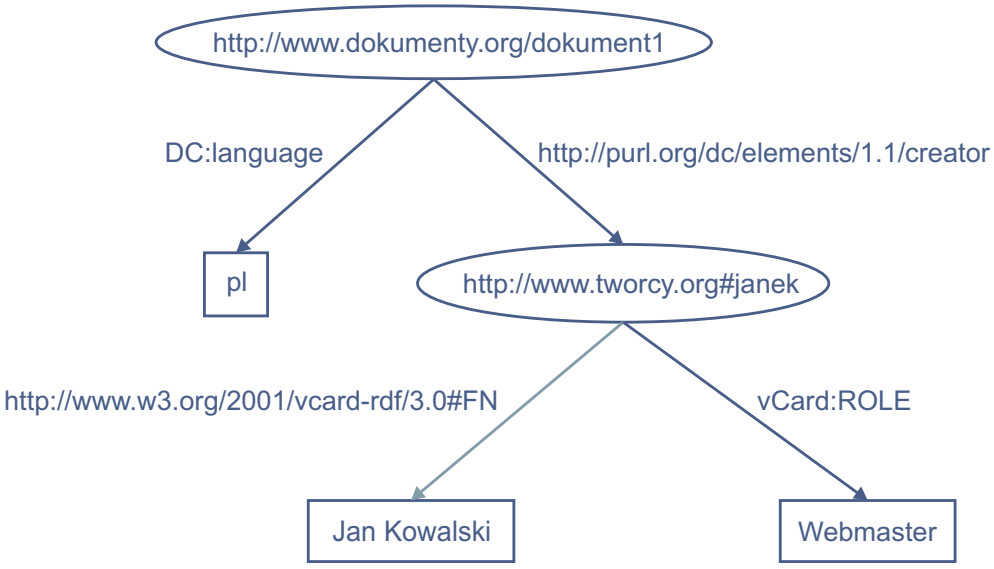
2.3.3 Zapis RDF w XML

Rozważmy teraz bardziej rozbudowany graf będący opisem (w RDF) pewnego "zasobu", przedstawiony na rysunku 3 i jego zapis w XML.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:DC="http://purl.org/dc/elements/1.1/"
         xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#"
 >
  <rdf:Description rdf:about="http://www.tworcy.org#janek">
    <vCard:FN>Jan Kowalski</vCard:FN>
    <vCard:ROLE>Webmaster</vCard:ROLE>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.dokumenty.org/dokument1">
    <DC:language>pl</DC:language>
    <DC:creator resource="http://www.tworcy.org/janek"/>
  </rdf:Description>
</rdf:RDF>
```

Użyte są w nim predykaty pochodzące z "biblioteki" Dublin Core (słownictwo zdefiniowane do opisu metadanych różnych dokumentów - w szczególności bibliotecznych: daty utworzenia, autora itp.) oraz zapisu wirtualnych wizytówek (vCard) w RDF. Jest to oznaczone w korzeniu dokumentu (którym jest zawsze znacznik `rdf:RDF`). Dzięki temu, że wszystkie predykaty w



Rysunek 3: Bardziej skomplikowany graf RDF

Dublin Core i vCard są zdefiniowane w swoich przestrzeniach nazw XML możemy zaznaczyć użycie tych przestrzeni w atrybutach węzła `rdf:RDF` i dalej w dokumencie odwoływać się do tych predykatów przez aliasy (zamiast za każdym razem przytaczać cały URI). Podmioty definiujemy znacznikiem `rdf:Description` z atrybutem `rdf:about` zawierającym URI zasobu. Zdania opisujące ten podmiot są zapisywane jako jego węzły potomne. Obiekty, które są jednocześnie zasobami zapisujemy w postaci węzła skróconego z URI tego zasobu umieszczonym w atrybucie `rdf:resource`.

Inną użyteczną funkcją zapisu RDF w XML jest wykorzystanie atrybutu `xml:base` służącego do zdefiniowania URI bazowego dokumentu. Aby skorzystać z tej funkcjonalności dodajemy atrybut `xml:base` do korzenia dokumentu (czyli w przypadku dokumentu RDF do węzła `rdf:RDF`). Po wyspecyfikowaniu tego atrybutu na `xml:base="http://www.tworcy.org"`, wiersz

```
<rdf:Description rdf:about="http://www.tworcy.org#janek">
```

możemy zamienić na formę z użyciem atrybutu `rdf:ID`:

```
<rdf:Description rdf:ID="janek">
```

Przestrzenie nazw pozwalają przede wszystkim na łatwiejsze łączenie wielu dokumentów przez eliminację konfliktów związanych z nazewnictwem. Dwa dokumenty mogą definiować tak samo nazwaną właściwość (np. tytuł)

ale dzięki zastosowaniu dwuczłonowej nazwy, składającej się z URI przestrzeni nazw i nazwy lokalnej, te dwa predykaty będą mogły być odróżnione, jeżeli będziemy chcieli użyć obu w jednym grafie RDF. Nazwa przestrzeni nazw często związana jest z nazwą instytucji tworzącej dane słownictwo, co ułatwia identyfikację autora. W przypadku aktualizacji wcześniej opracowanego słownictwa można odróżnić starsze predykaty od nowszych przez zaszycie w nazwie przestrzeni daty jego utworzenia. Takie rozwiązanie jest właśnie stosowane do standardowego zbioru predykatów RDF. Umieszczony jest on w przestrzeni nazw o URI `http://www.w3.org/1999/02/22-rdf-syntax-ns`. Zastosowanie przestrzeni nazw zwiększa również czytelność dokumentu RDF - zamiast długich URI mamy krótkie identyfikatory.

2.3.4 Typowane literały

Literały w RDF mogą mieć wyspecyfikowany typ. Do oznaczenia faktu, że literał powinien być traktowany jako dana określonego typu używamy atrybutu `rdf:datatype`. Najczęściej używane są typy danych zdefiniowane przez W3C w schematach XML. I tak do poprzedniego przykładu możemy dodać zdanie informujące o dacie utworzenia dokumentu, węzeł opisujący ten zasób zyska dodatkowy węzeł potomny:

```
<DC:date rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
   2005-09-01
</DC:date>
```

Zastosowanie ścisłego systemu typów nakłada ograniczenia na wartości jakie mogą przyjąć literały. Wzbogacone zostaje znaczenie predykatów i aplikacja wykorzystująca informacje o typach może w zautomatyzowany sposób wykryć błędne zastosowania literałów takie jak użycie słowa gdy spodziewana jest liczba. Ułatwia to zachowanie spójności.

Dla poprawienia czytelności dokumentu często wykorzystywane są encje XML. W tym celu należy umieścić tuż po deklaracji dokumentu XML dodatkową deklarację `DOCTYPE`:

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd"http://www.w3.org/2001/XMLSchema#">]>
```

Teraz w przykładzie z datą możemy zastąpić długi URI przez referencję do encji:

```
<DC:date rdf:datatype="&xsd;date">2005-09-01</DC:date>
```

Podsumowując wszystkie wprowadzone zmiany rozważany przykład grafu RDF będzie teraz miał postać przedstawioną jak na listingu poniżej:

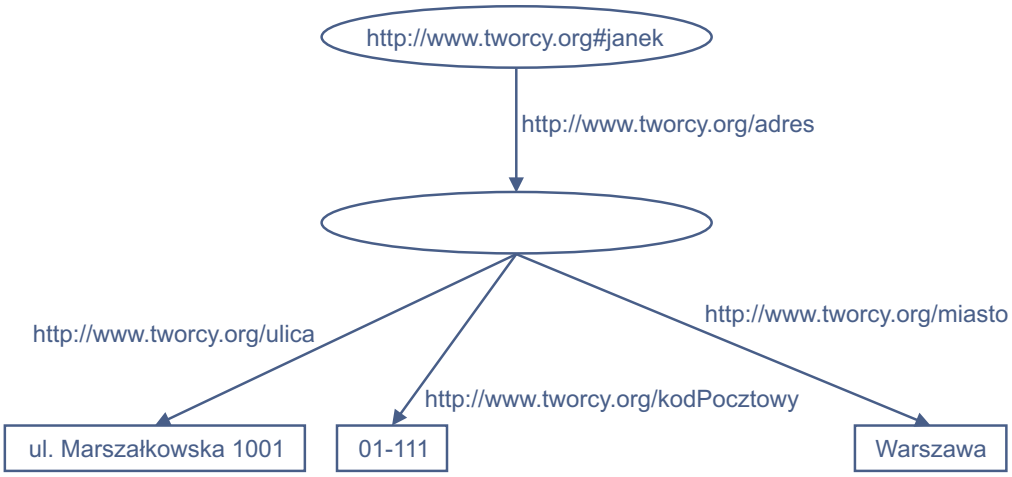
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xml:base="http://www.tworcy.org"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:DC="http://purl.org/dc/elements/1.1/"
         xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#"
 >
   <rdf:Description rdf:ID="janek">
      <vCard:FN>Jan Kowalski</vCard:FN>
      <vCard:ROLE>Webmaster</vCard:ROLE>
   </rdf:Description>

   <rdf:Description rdf:about="http://www.dokumenty.org/dokument1">
      <DC:language>pl</DC:language>
      <DC:creator resource="http://www.tworcy.org/janek"/>
      <DC:date rdf:datatype="&xsd;date">2005-09-01</DC:date>
   </rdf:Description>
</rdf:RDF>
```

2.3.5 Puste węzły

W rozważanych do tej pory sytuacjach węzeł albo był etykietowany pewnym URI, albo był literałem. RDF dopuszcza jeszcze tzw. puste węzły (ang. *blank nodes*). Są to węzły, które nie są literałami i nie są opisane żadnym URI. Podczas budowania bardziej skomplikowanych grafów często zachodzi sytuacja, w której wstawiany jest węzeł o roli węzła pośredniego. Najczęściej ma to miejsce wtedy, gdy kilka zdań opisuje pewien koncept, który z kolei jest wykorzystywany do charakteryzowania obiektu nadrzędnego. Do takiego węzła pośredniego często nie ma innych odwołań poza przypisaniem do węzła nadrzędnego, więc nie ma potrzeby nadawania "pośrednikowi" żadnego URI. Prosty przykład takiej sytuacji i zastosowania węzła pustego podaje W3C RDF Primer [32]. Rozważmy mianowicie osobę i jej adres zamieszkania. Na rysunku 4 widać, że pusty węzeł jest trzykrotnie użyty jako podmiot (adres jest opisywany przez ulicę, miasto i kod pocztowy) a raz jako obiekt (osobie przypisujemy adres).

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:to="http://www.tworcy.org/"
 >
  <rdf:Description rdf:about="http://www.tworcy.org#janek">
    <to:adres rdf:nodeID="blank"/>
  </rdf:Description>
```



Rysunek 4: Przykład zastosowania pustego węzła

```
  <rdf:Description rdf:nodeID="blank">
    <to:ulica>ul. Marszałkowska 1001</to:ulica>
    <to:kodPocztowy>01-111</to:kodPocztowy>
    <to:miasto>Warszawa</to:miasto>
  </rdf:Description>
</rdf:RDF>
```

Zapis tego grafu w XML będzie miał postać jak na listingu powyżej. Pusty węzeł w RDF/XML jest oznaczany przy użyciu atrybutu `rdf:nodeID`. Wartość tego atrybutu jest używana do odwołań i identyfikacji węzła wewnątrz dokumentu na podobnej zasadzie jak "pełnowartościowe" URI. Natomiast na zewnątrz dokumentu taki identyfikator nie jest widoczny.

2.3.6 Kontenery

Język RDF posiada także udogodnienia do opisu grup obiektów. W terminologii RDF do przypisywania zasobom grup obiektów używamy jednego z trzech typów kontenerów (ang. *containers*), zależnie od wzajemnych relacji między elementami grupy:

- Worek (ang. *bag*, znacznik `rdf:Bag`) jest najprostszym kontenerem. Nazwa dobrze oddaje charakter i przeznaczenie tego kontenera, obiekty są do niego wrzucane jak do worka. Nie są w żaden sposób uporządkowane i mogą się powtarzać.

- Sekwencja (ang. *sequence*, znacznik `rdf:Seq`) służy do reprezentowania listy obiektów tzn. ich kolejność w kontenerze ma znaczenie. Obiekty w sekwencji mogą się pojawiać wielokrotnie.

- Alternatywa (ang. *alternative*, znacznik `rdf:Alt`) służy do reprezentowania grupy obiektów, które są wartościami alternatywnymi (np. w metadanych opisujących plik na serwerze FTP może znaleźć się lista serwerów lustrzanych, na których również znajduje się ten plik).

Należy przy tym zauważyć, że to po stronie aplikacji leży odpowiedzialność za prawidłowe wykorzystanie znaczników opisujących kontenery, gdyż sam RDF definiuje jedynie typy i predykaty, których można użyć do konstrukcji kontenerów przechowujących zasoby.

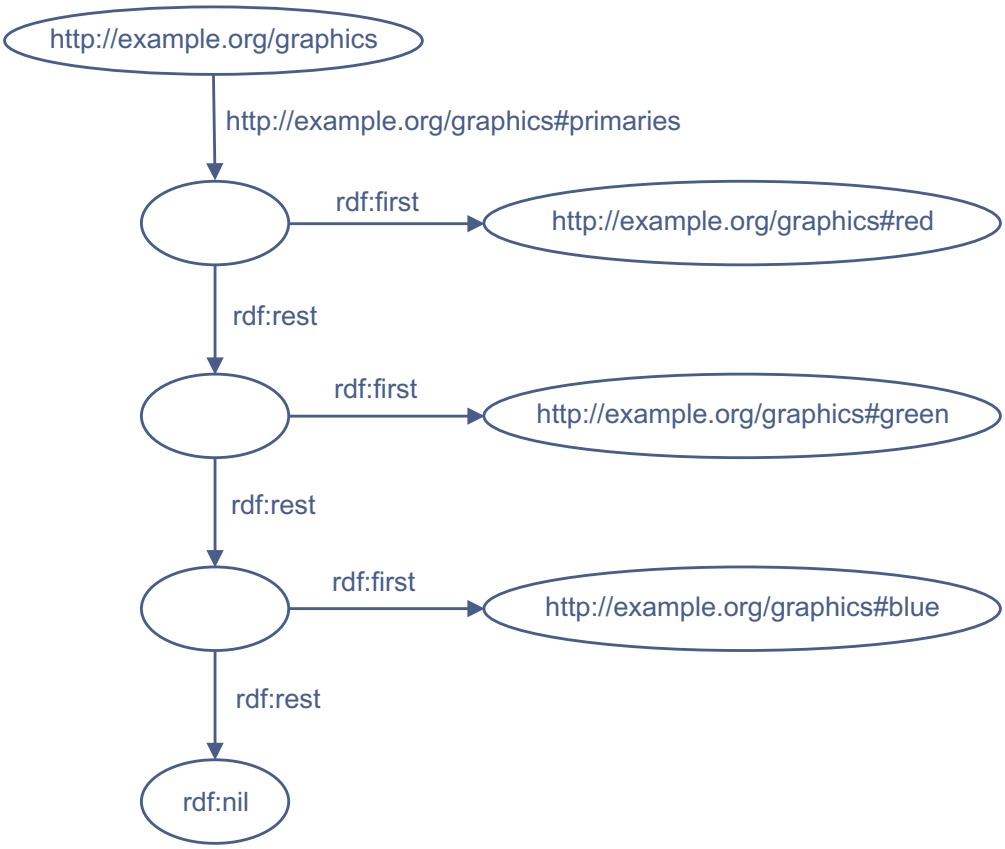
2.3.7 Kolekcje

Opisane w poprzednim punkcie kontenery mają strukturę otwartą tzn. nie można wyspecyfikować pewnej liczby obiektów i stwierdzić, że są to już wszystkie elementy danego kontenera. W szczególności zdania opisujące zawartość kontenera mogą się pojawiać w wielu miejscach grafu. Dlatego RDF udostępnia strukturę kolekcji (ang. *collection*).

Budowę kolekcji omówię na przykładzie. Rozważmy sytuację, w której opisujemy pojęcia związane z grafiką. Chcemy wyspecyfikować trzy barwy jako podstawowe: czerwoną, zieloną i niebieską. Możemy do tego celu użyć kolekcji przedstawionej na rysunku 5 i listingu poniżej.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/graphics">
  <rdf:Description rdf:about="http://example.org/graphics">
    <ex:primaries rdf:parseType="Collection">
      <rdf:Description rdf:about="http://example.org/graphics#red"/>
      <rdf:Description rdf:about="http://example.org/graphics#green"/>
      <rdf:Description rdf:about="http://example.org/graphics#blue"/>
    </ex:primaries>
  </rdf:Description>
</rdf:RDF>
```

Kolekcja jest reprezentowana w RDF jako lista z nazewnictwem i strukturą zaczerpniętymi z języka programowania Lisp. Węzły tej listy są puste, z każdego wychodzą dwie krawędzie. Jedna, etykietowana `rdf:first` wskazuje na węzeł w którym przechowywany jest klucz (czyli element kolekcji). Druga,



Rysunek 5: Przykładowa kolekcja

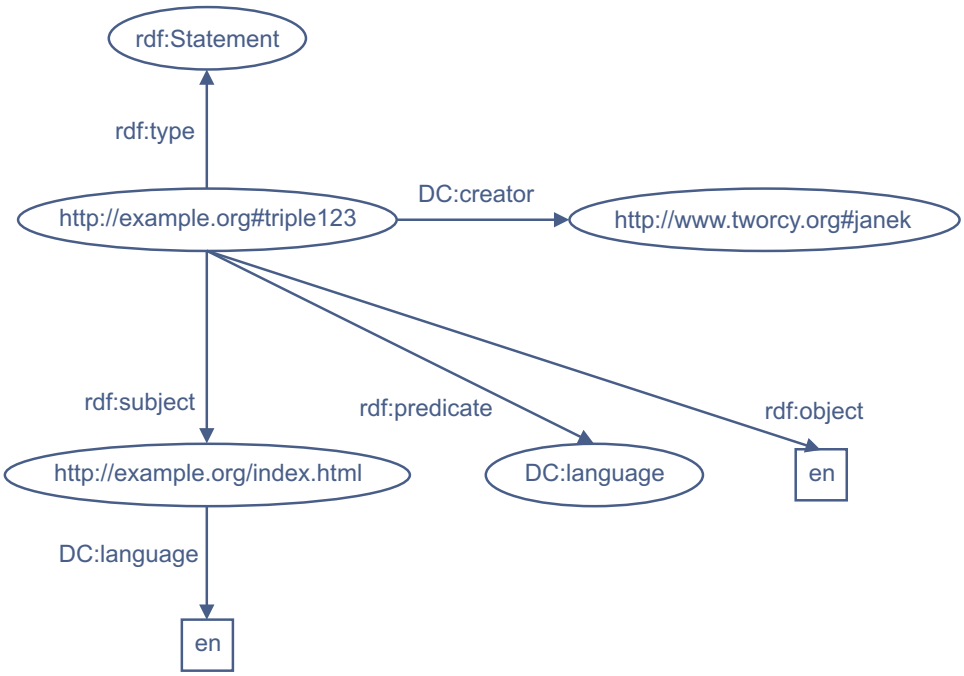
etykietowana `rdf:rest` pełni rolę wskaźnika "next" w klasycznej implementacji listy, czyli pozwala na przejście do następnego elementu. Ostatni pusty węzeł ma wskaźnik wskazujący na specjalny element etykietowany `rdf:nil`, czyli kończący listę. RDF/XML upraszcza w dużym stopniu tę skomplikowaną strukturę ukrywając szczegóły przed użytkownikiem.

2.3.8 Reifikacje

W języku RDF całe zdania mogą być również traktowane jako zasoby. Dzięki funkcjonalności określanej mianem reifikacji (ang. *reification*) można konstruować stwierdzenia, których podmiotami są inne stwierdzenia. Jest to bardzo użyteczne w wielu sytuacjach np. gdy chcemy zapisać kto jest autorem danego stwierdzenia. RDF posiada specjalne elementy słownictwa, których można użyć do opisu stwierdzenia (czyli właśnie reifikacji): `rdf:Statement`, `rdf:subject`, `rdf:predicate`, `rdf:object`. Dla przykładu rozważmy pojedyncze zdanie w RDF opisujące język w jakim jest napisany dokument o adresie `http://example.org/index.html`.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:DC="http://purl.org/dc/elements/1.1/"
 >
  <rdf:Description rdf:about="http://example.org/index.html">
    <DC:language>en</DC:language>
  </rdf:Description>
</rdf:RDF>
```

Załóżmy teraz, że chcemy zapisać kto jest autorem tego stwierdzenia. W tym celu możemy zastosować reifikację i przetransformować to zdanie do grafu przedstawionego na rysunku 6. Kluczowe jest wprowadzenie dodatkowego węzła o URI `http://example.org#triple123`, pełniącego rolę identyfikatora stwierdzenia (informuje o tym zdanie z predykatem `rdf:type` i obiektem `rdf:Statement`). Trzy zdania z predykatami `rdf:subject`, `rdf:predicate` i `rdf:object` specyfikują elementy opisywanego zdania: podmiot, orzeczenie i obiekt. Graf jest rozszerzony również o zdanie z identyfikatorem trójki użytym bezpośrednio jako podmiotem. Mówi ono, że osoba identyfikowana jako `http://www.tworcy.org#janek` była autorem stwierdzenia. Bezpośredni zapis tego grafu miałby postać jak na listingu poniżej:



Rysunek 6: Graf zdania zapisanego z użyciem reifikacji

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:DC="http://purl.org/dc/elements/1.1/"
         xml:base="http://example.org"
 >
  <rdf:Statement rdf:ID="triple123">
    <rdf:subject rdf:resource="http://example.org/index.html"/>
    <rdf:predicate
rdf:resource="http://purl.org/dc/elements/1.1/language"/>
    <rdf:object>en</rdf:object>
    <DC:creator rdf:resource="http://www.tworcy.org#janek"/>
  </rdf:Statement>

  <rdf:Description rdf:about="http://example.org/index.html">
    <DC:language>en</DC:language>
  </rdf:Description>
</rdf:RDF>
```

Zapis w XML pozwala jednak na duże uproszczenie i zmniejszenie liczby zdań wpisywanych explicite dzięki wykorzystaniu atrybutu `rdf:ID`. Zamiast specyfikować elementy trójki za pomocą predykatów `rdf:subject`, `rdf:predicate` i `rdf:object` wystarczy nadać zdaniu identyfikator (wpi-

sując go jako atrybut `rdf:ID`) i odwoływać się do niego dalej tak jak do ID zwykłego zasobu:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:DC="http://purl.org/dc/elements/1.1/"
         xml:base="http://example.org"
 >
  <rdf:Description rdf:about="http://example.org/index.html">
    <DC:language rdf:ID="triple123">en</DC:language>
  </rdf:Description>

  <rdf:Description rdf:about="#triple123">
    <DC:creator rdf:resource="http://www.tworcy.org#janek"/>
  </rdf:Description>
</rdf:RDF>
```

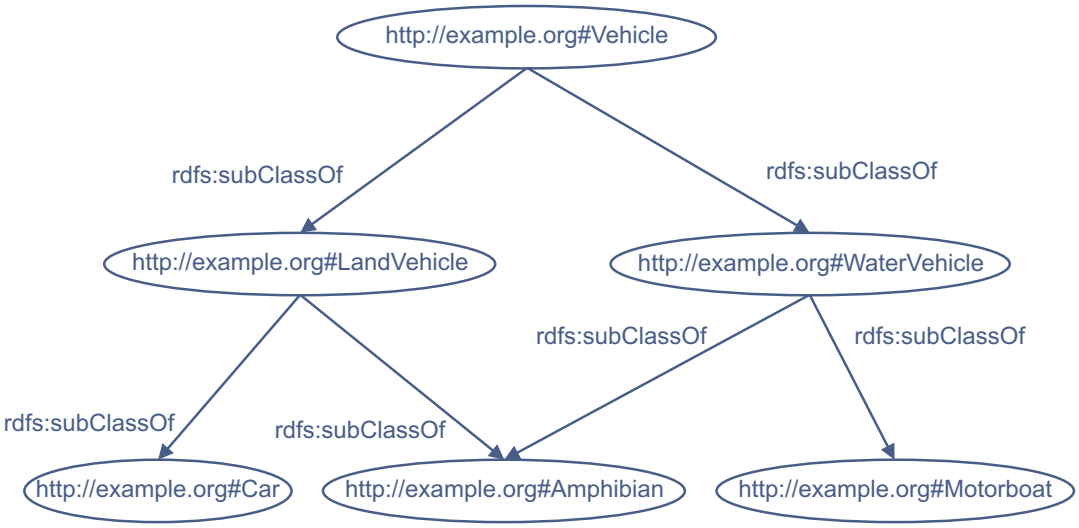
2.4 Schematy RDF

Schematy RDF (ang. *RDF Schema*, RDFS) pełnią wobec RDF podobną rolę jak schematy XML wobec czystego XML. Sam RDF dostarcza jedynie podstawowych struktur do opisu abstrakcyjnych elementów zwanych zasobami. Brakuje jednak bardziej złożonych struktur służących do "specyfikowania konceptualizacji". Nie można chociażby określić, że predykat powinien być używany jedynie do pewnego typu zasobów. RDFS umożliwia jednak modelowanie pojęć bardzo zbliżone do używanego przez programistów modelowania obiektowego. Wprowadza bowiem takie pojęcia jak klasa, podklasa, właściwość.

2.4.1 Klasy i podklasy w RDFS

RDFS pozwala na definiowanie klas obiektów przy użyciu znacznika `rdfs:Class`. Podobnie jak w obiektowych językach programowania w RDFS można stosować dziedziczenie i wprowadzać klasy potomne przy użyciu znacznika `rdfs:subClassOf`. Dopuszczalne jest wielokrotne dziedziczenie. Przykładowy graf przedstawiony na rysunku 7 opisuje prostą hierarchię typów pojazdów. Amfibia jest zaliczana zarówno do pojazdów lądowych jak i wodnych.

Reprezentacja tego grafu w RDF/XML jest bardzo prosta. Dla zilustrowania możliwości języka RDF umieściłem tutaj dwa możliwe sposoby deklarowania klas:



Rysunek 7: Prosta hierarchia

1. przez wykorzystanie znacznika `rdfs:Class`;
2. przez bezpośrednią specyfikację typu zasobu z użyciem znacznika `rdfs:type` (deklaracje klas `Vehicle` i `LandVehicle` w przykładzie).

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org"
 >

<rdf:Description rdf:ID="Vehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="LandVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Vehicle"/>
</rdf:Description>

<rdfs:Class rdf:ID="WaterVehicle">
  <rdfs:subClassOf rdf:resource="#Vehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Car">
  <rdfs:subClassOf rdf:resource="#LandVehicle"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="Motorboat">
  <rdfs:subClassOf rdf:resource="#WaterVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Amphibian">
  <rdfs:subClassOf rdf:resource="#WaterVehicle"/>
  <rdfs:subClassOf rdf:resource="#LandVehicle"/>
</rdfs:Class>

</rdf:RDF>
```

Równie proste jest tworzenie instancji obiektów - używamy zdefiniowanego przez nas znacznika i nadajemy obiektowi identyfikator:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org#"
  xml:base="http://example.org"
 >
  <ex:Car rdf:ID="Porsche"/>
</rdf:RDF>
```

2.4.2 Właściwości

Każdy obiekt chcielibyśmy jakoś scharakteryzować przez opis jego cech szczególnych. RDFS posiada funkcjonalność pozwalającą na definiowanie właściwości (ang. *properties*) obiektów. Właściwości są w RDF instancjami standardowej klasy `rdf:Property` i mogą nie być przypisane do obiektów żadnej klasy, mogą również nie posiadać żadnego typu. Jest to jednak rzadko wykorzystywane, najczęściej dążymy do stworzenia pełnego opisu obiektu przez wyspecyfikowanie zbioru opisujących go typowanych właściwości. Przyporządkowywanie właściwości klasom jest realizowane przez znacznik `rdf:domain`, natomiast typ danych jaki jest dopuszczalny dla danej właściwości definiujemy przy użyciu znacznika `rdf:range`. Na listingu poniżej jest przedstawiona definicja właściwości `http://example.org#maximalSpeed`, czyli maksymalnej prędkości, jaką może osiągnąć pojazd (kontynuacja przykładu z poprzedniego punktu). Prędkość wyrażamy w liczbach całkowitych (`integer` zdefiniowany przez XSD).

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<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:ex="http://example.org#"
```

```
  xml:base="http://example.org"
 >
  <rdf:Property rdf:ID="maximalSpeed">
    <rdfs:domain rdf:resource="#Vehicle"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
  </rdf:Property>
</rdf:RDF>
```

Należy przy tym wspomnieć o pewnych niuansach związanych z właściwościami, które mogą nie być intuicyjne. Jeżeli przy definicji jednej właściwości pojawi się kilka zdań opisujących jej domenę to oznacza, że właściwość ta dotyczy obiektów, które są instancjami wszystkich klas wymienionych przez zdania z `rdfs:domain`. W ten sposób można łatwo zdefiniować np. czas potrzebny do przetransformowania wehikułu lądowego w wodny. Jeżeli określimy domenę tej właściwości zarówno na `LandVehicle` i `WaterVehicle` to będzie ją można stosować do amfibii ale nie do samochodu i motorówki.

Jak wspomniałem `rdf:Property` jest klasą, można więc również stosować do właściwości dziedziczenie i definiować właściwości potomne (ang. *subproperties*), służy do tego znacznik `rdfs:subPropertyOf`.

2.5 RDF Query Language

Przeszukiwanie grafów RDF może być uproszczone dzięki zastosowaniu języka zapytań RDQL (ang. *RDF Query Language*). Zapytania w RDQL polegają na specyfikowaniu wzorców grafów z pewną ilością zmiennych. Program wykonujący zapytanie RDQL przeszukuje graf modelu w poszukiwaniu podgrafów pasujących do wzorca. Wynikiem zapytania jest lista przyporządkowań, każde takie przyporządkowanie jest zbiorem par: nazwa zmiennej i wartość (dla każdej zmiennej zdefiniowanej w zapytaniu). Składnia RDQL jest wzorowana w dużym stopniu na języku SQL. Na listingu poniżej załączyłem przykładowe zapytanie używające ontologii restauracji. Kwerenda ma na celu uzyskanie nazw i adresów restauracji warszawskich, które pozwalają na organizację prywatnych przyjęć na co najmniej 40 osób i w których podawane są dania kuchni francuskiej.

```
SELECT ?name, ?address
WHERE
   (?id rdf:type res:Restaurant)
   (?id res:title ?name)
   (?id res:streetAddress ?address)
   (?id loc:city "Warszawa")
   (?id res:capacity ?seats)
```

```
   (?id res:feature <http://www.agentlab.net/schemas/restaurant#PrivateParties>)
   (?id res:cuisine <http://www.agentlab.net/schemas/restaurant#FrenchCuisine>)
AND
   ?seats >= 40
USING
   res FOR <http://www.agentlab.net/schemas/restaurant#>
   loc FOR <http://www.agentlab.net/schemas/location#>
   mon FOR <http://www.agentlab.net/schemas/money#>
   ost FOR <http://www.agentlab.net/schemas/ontoStorage#>
```

W części `SELECT` specyfikuje się zmienne reprezentujące informacje, które chcemy wydobyć z grafu RDF. W przykładzie są to zmienne `?name` (nazwa restauracji) i `?address` (adres restauracji). W sekcji `WHERE` podajemy listę opisującą wzorzec grafu RDF w postaci trójek `(podmiot predykat obiekt)`. Możemy tutaj używać elementów ontologii (adresów URI zasobów i właściwości a także literałów), zmiennych zdefiniowanych w sekcji `SELECT` oraz dodatkowych zmiennych pomocniczych (nieuwzględnianych w raportowaniu wyników, takimi zmiennymi są w przykładzie `?id` i `?seats`). Zastosowanie klauzuli `AND` pozwala nałożyć dodatkowe ograniczenia na zmienne z użyciem operatorów porównania, logicznych itp. Dzięki temu możemy określić, że interesują nas restauracje z minimalną liczbą miejsc równą 40 (zmienna `?seats`). Opcjonalna sekcja `USING` zawiera definicje aliasów dla przestrzeni nazw, których można używać w sekcjach `WHERE` i `AND`.

2.6 Ontologie

Ontologia jest pojęciem zaczerpniętym z filozofii, gdzie między innymi oznacza analizę pojęć i idei w celu ustalenia co istnieje oraz jakie są związki pomiędzy istniejącymi elementami. W informatyce tego pojęcia używa się w bardzo podobnym kontekście, oznacza ono formalną specyfikację konceptualizacji pewnego obszaru wiedzy czy opisu elementów rzeczywistości ([22]); oczywiście w formie łatwej do wykorzystania przez oprogramowanie. Przy projektowaniu ontologii wykorzystywane są metody kategoryzacji i hierarchizacji. Pewnym pojęciom abstrakcyjnym i grupom obiektów posiadającym wspólne cechy przyporządkowywane są nazwy, tworzone są więc klasy obiektów (kategoryzacja). Tak utworzone klasy są następnie umieszczane w strukturze hierarchicznej. Przy tworzeniu ontologii można użyć różnych języków modelowania. Najczęściej wykorzystywany jest RDF/RDFS lub bardziej wyrafinowane języki zbudowane w oparciu o te technologie.

Do najczęściej stosowanych języków, w których opisywane są ontologie należą:

- DAML+OIL (*DARPA Agent Markup Language + Ontology Inference Layer*) jest językiem, nad którym prace rozpoczęły się w roku 2000 w amerykańskim Departamencie Obrony. Jest zbudowany w oparciu o RDF ale ma znacznie więcej możliwości, z których najważniejsze to:
  - definiowanie ograniczeń na wartości, jakie mogą przyjąć właściwości;
  - wprowadzenie jawnego odróżnienia właściwości, których obiektami mają być zasoby i literały;
  - możliwość definiowania właściwości o przeciwstawnym znaczeniu;
  - właściwości mogą być określone jako przechodnie;
  - dopuszczenie cykli w hierarchii klas;
  - możliwość określenia krotności właściwości;
  - wprowadzenie możliwości zdefiniowania jednoznacznej właściwości, identyfikującej zasób, pełniącej podobną rolę jak klucz pierwotny w relacyjnych bazach danych.
- OWL (ang. *Web Ontology Language*) jest językiem opracowanym przez W3C Consortium i w dużym stopniu bazowanym na DAML+OIL. Istnieją trzy dialekty OWL, istotnie różniące się jeżeli chodzi o zaawansowane możliwości:
  - OWL Lite pozwala na tworzenie hierarchii klas z prostymi ograniczeniami, narzędzia przeznaczone dla OWL Lite są łatwe w implementacji.
  - OWL DL (ang. *OWL Description Logic*) powstał z myślą o bardziej zaawansowanych zastosowaniach. OWL DL jest bardziej ekspresywny niż OWL Lite. Szczególny nacisk położono w nim na wsparcie dla wnioskowania (dowolne wnioskowanie jest obliczalne w skończonym czasie). Na użycie konstrukcji językowych nałożono w związku z tym bardzo dużo ograniczeń.
  - OWL Full można traktować jak mniej restrykcyjną wersję OWL pod względem ograniczeń w użyciu konstrukcji językowych niż OWL DL, których można używać dużo swobodniej. Ale w związku z tym nie ma gwarancji związanych z obliczalnością wnioskowania.

  Dlatego w specyfikacji OWL opracowanej przez W3C [33] znajduje się stwierdzenie, że OWL Lite można traktować jak rozszerzenie RDF a OWL DL i OWL Full jak rozszerzenia RDF z restrykcjami.

2.7 Jena

Jena ([28]) jest najpopularniejszą biblioteką używaną przez aplikacje wykorzystujące dane zapisane w RDF. Prace nad systemem Jena rozpoczęły się w dziale badawczym firmy Hewlett-Packard, obecnie jest to oprogramowanie klasy Open Source. Jenę napisano w Javie i zapewnia ona w pełni obiektowy interfejs do dokonywania manipulacji na grafach RDF (nazywanych przez dokumentację modelami). Modele mogą być nie tylko przechowywane w pamięci podczas działania programu; mogą one być również zapisywane do pliku. Jena umożliwia także tworzenie tzw. modeli trwałych (ang. *persistent models*), w których trójki RDF są zapisywane w relacyjnej bazie danych na serwerze SQL takim jak MySQL, PostgreSQL czy Oracle. Ponadto Jena akceptuje ontologie stworzone przy użyciu RDFS, OWL i DAML+OIL (także ze wsparciem dla wnioskowania). Ma również klasy odpowiedzialne za wykonywanie zapytań RDQL.

3 Systemy wieloagentowe

3.1 Czym jest agent?

Wśród badaczy nie ma zgodności co do formalnej definicji pojęcia agenta. Stan Franklin i Art Graesser w pracy [14] dotyczącej systematyki agentów zebrali kilka najczęściej używanych rozwinięć tego terminu. Przytoczę te definicje i spróbuję wychwycić kilka cech charakteryzujących agentów, odróżniających ich od zwykłego programu. Tłustym drukiem podaję nazwę pod jaką dana definicja jest wymieniana w literaturze.

- **Agent MuBot**: *Termin agent jest używany do reprezentowania dwóch pojęć. Pierwszym jest zdolność agenta do autonomicznej pracy. Drugą jest zdolność agenta do przeprowadzania wnioskowania w określonej dziedzinie.*
  (Według Sankara Virdhagriswarana, [53]).

- **Agent AIMA**: *Za agenta można uznać cokolwiek, co postrzega swoje otoczenie przez czujniki i oddziałuje na to otoczenie przez efektory.*
  (Definicja zaczerpnięta z książki Stuarta Russella i Petera Norviga "Artificial Intelligence: A Modern Approach" [45]).

- **Agent Maes**: *Agenci to systemy znajdujące się w złożonym, dynamicznym środowisku, odbierające z niego bodźce i działające w nim autonomicznie. Agenci dążą do realizacji celów lub wypełniania zadań, do których zostali zaprojektowani.*

(Definicja autorstwa Pattie Maes, jednej z pionierek badań nad systemami wieloagentowymi, pochodząca z pracy [31]).

- **Agent KidSim**: *Agenta definiujemy jako trwałą jednostkę programową stworzoną do szczególnych zastosowań. Słowo "trwały" odróżnia agentów od podprogramów. Agenci mają swoje własne założenia jak realizować zadania, swoje własne zbiory procedur. "Szczegółowe zastosowania" odróżniają agentów od całych, wielofunkcyjnych aplikacji; agenci są zazwyczaj dużo mniejsi.*
  (Definicja z pracy [48], w której autorzy rozważają środowisko wizualnego programowania agentów przeznaczone dla dzieci. KidSim to skrót od "Kids' Simulations").

- **Agent Hayes-Roth**: *Inteligentni agenci w sposób ciągły wykonują trzy czynności:*

  - *Obserwują dynamicznie zachodzące zmiany w środowisku.*
  - *W zależności od zmieniających się warunków podejmują akcje wpływające również na otoczenie.*
  - *Przeprowadzają wnioskowanie, aby zinterpretować obserwacje, rozwiązać problemy i określić akcje jakie należy wykonać.*

  (Definicja Barbary Hayes-Roth z Uniwersytetu Stanford, pochodząca z pracy [24]).

- **Agent IBM**: *Inteligentni agenci są jednostkami programowymi wykonującymi pewne zbiory operacji w imieniu użytkownika lub innego programu. Są w pewnym stopniu niezależni czy też autonomiczni i wykorzystują wiedzę o celach i życzeniach użytkownika.*
  (Wizja agentów według IBM, umieszczona w białej księdze [42] w całości poświęconej systemom wieloagentowym)

- **Agent Wooldridge i Jennings**: *Sprzętowy lub częściej programowy system komputerowy mający następujące właściwości:*

  - *autonomia: agenci działają bez bezpośredniej interwencji użytkownika i mają pewien stopień kontroli nad swoimi akcjami i wewnętrznym stanem*;
  - *zdolności do pracy w grupie: agenci wzajemnie na siebie oddziaływują przy pomocy pewnego języka komunikacji*;

– *reaktywność: agenci obserwują swoje otoczenie (którym może być świat fizyczny, użytkownik widziany przez pryzmat graficznego interfejsu, zbiór innych agentów, Internet i inne) i reagują w odpowiedni sposób na zmiany, które w nim zachodzą.*

– *aktywność: agenci nie odpowiadają tylko na bodźce ze środowiska ale są również zdolni do wykazywania inicjatywy w dążeniu do celu* (Definicja z [54]).

W każdej z tych prób uchwycenia znaczenia (a czasami także zdefiniowania) pojęcia agenta programowego akcent jest położony na inne aspekty jego charakterystyki. Można jednak wyróżnić szereg często wymienianych, wspólnych cech. Agent nie jest luźnym fragmentem kodu lecz pewną jednostką oprogramowania (ang. *software artifact*) o trwałym charakterze. Dąży do celu, zleconego przez użytkownika lub inny program. Zadania jakie realizuje pojedynczy agent są wysoko wyspecjalizowane (nie chcemy, żeby agent był przesadnie skomplikowany). Zleceniodawca określa cel działania agenta, natomiast to w jaki sposób agent wypełni zadanie, zależy już wyłącznie od niego. Agent działa bowiem autonomicznie, sam podejmuje decyzje o tym, jakie akcje w danej chwili powinien wykonać. Wpływ na ten werdykt może mieć stan otoczenia w jakim się znajduje (niezależnie od tego, czy mówimy o otoczeniu fizycznym, czy programowym). Interakcja ze środowiskiem nie jest ograniczona do jednostronnego reagowania na bodźce. Agent może również wykonywać czynności zmieniające stan swojego otoczenia. W szczególności agenci mogą ze sobą współpracować, komunikować się, wymieniać informacjami i wzajemnie zlecać sobie zadania do wykonania.

3.2 Standardy FIPA

Dotychczasowy opis mówił jedynie o dość abstrakcyjnej idei agenta programowego. Natomiast aby możliwa była szybka implementacja systemów wieloagentowych potrzebna jest pewna platforma, udostępniająca szereg mechanizmów wspierających działanie agentów. Na przykład: specjalizacja agentów pociąga za sobą konieczność ich współpracy. Agenci muszą więc mieć możliwość opisania, jakie usługi potrafią realizować, potrzebny jest także mechanizm lokalizacji agentów wykonujących daną usługę. Agenci mogą być tworzeni przez różnych programistów, różne instytucje. Zapewnienie między nimi łączności wymaga ustandaryzowania protokołów komunikacyjnych.

Organizacją opracowującą specyfikacje i wyznaczającą standardy dla systemów wieloagentowych jest istniejące od roku 1996 stowarzyszenie FI-

PA (ang. *Foundation for Intelligent Physical Agents*, [49]). W czerwcu 2005 FIPA została zaakceptowana przez IEEE jako oficjalny, jedenasty opiekun standardów w ramach IEEE. W roku 2002 został przez FIPA wydany zestaw dokumentów opisujących platformę do tworzenia systemów wieloagentowych. Standard ten wymienia komponenty, jakie powinna posiadać platforma i jakie usługi powinny być przez nie udostępniane. Opis dotyczy tego, jak powinny zachowywać się poszczególne składniki ale nie definiuje ich wewnętrznej implementacji (standard FIPA pełni więc podobną rolę jak specyfikacja J2EE dla serwerów aplikacji). Usługi podzielone są na dwie grupy: normatywne i opcjonalne. Do wymaganych elementów platformy należą:

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

Do funkcji opcjonalnych należą:

- Integracja agentów z innym oprogramowaniem;
- Usługi związane z ontologiami;
- Interakcja między człowiekiem a agentem programowym.

Wprowadzony jest również język ACL (ang. *Agent Communication Language*), służący do wymiany informacji między agentami. ACL jest zbudowany w oparciu o teorię aktów mowy. Zdefiniowane są 22 podstawowe cele komunikacji, takie jak informowanie (INFORM), żądanie (REQUEST) czy propozycja (PROPOSE). Opracowano również rozbudowane protokoły interakcji (ang. *interaction protocols*) pełniące rolę wzorców postępowania dla

agentów chcących wykonać popularne akcje takie jak na przykład zlecenie zadania do wykonania. Struktura wiadomości jest dokładnie zdefiniowana i pozwala na identyfikację nadawcy i odbiorcy, zawartości wiadomości i jej własności takich jak np. użyte kodowanie czy ontologia.

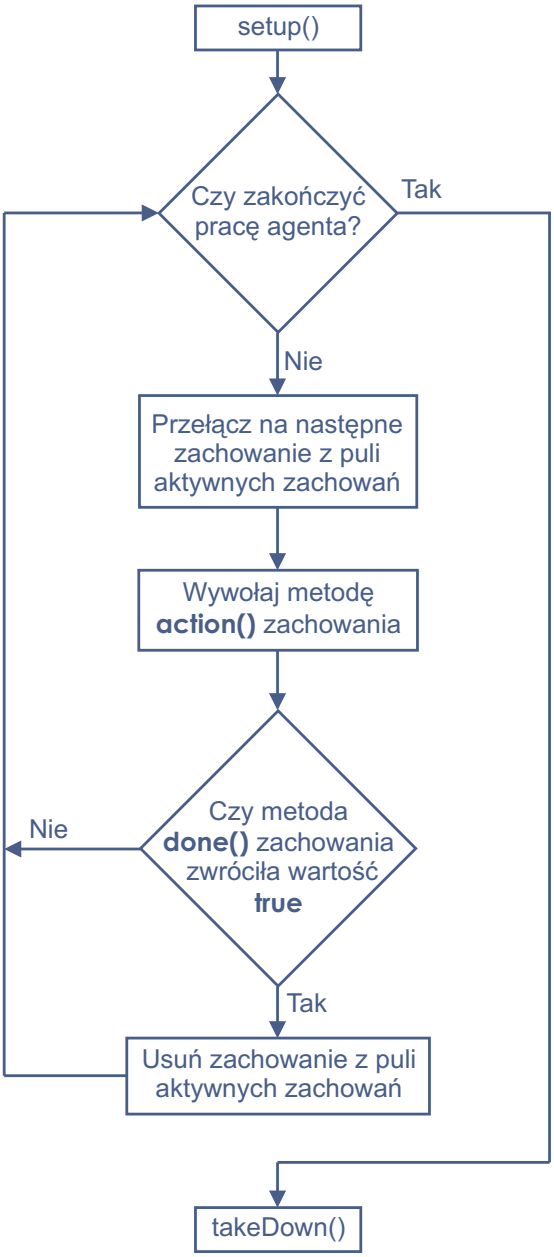
3.3 Java Agent Development Framework

Jedną z najpopularniejszych obecnie platform do tworzenia systemów wieloagentowych jest JADE (ang. *Java Agent Development Framework*, [27]). Środowisko to jest w pełni zgodne ze standardami FIPA. Językiem użytym do implementacji jest Java, co zapewnia temu systemowi bardzo dużą przenośność. Twórcy zadbali także o to, aby możliwe było wykorzystanie dowolnej z wersji platformy Javy: J2SE, J2EE a także J2ME. Elementami JADE są nie tylko biblioteki umożliwiające implementację agentów ale również kontener, czyli środowisko służące do uruchamiania agentów i zapewniające im usługi wymienione w specyfikacjach FIPA. Dzięki kontenerowi agenci są odizolowani od systemu operacyjnego i architektury sieci w której działają, komunikacja między nimi jest zapewniana przez usługę ACC (ang. *Agent Communication Channel*). Agenci mogą być również uruchamiani na urządzeniach przenośnych wyposażonych w J2ME. Jest to możliwe dzięki modułowej konstrukcji (niewykorzystywane przez dany system usługi mogą być wyłączone) i opcji tworzenia kontenera rozdzielonego. Na urządzeniu przenośnym jest uruchamiany jedynie bardzo lekki front-end, natomiast wszystkie zaawansowane usługi działają na komputerze bogatszym w zasoby.

3.3.1 Realizacja agenta w JADE

Realizacja agentów przy użyciu JADE jest bardzo prosta. Programista tworzy nową klasę dziedziczącą z klasy `Agent`, w której do zaimplementowania są dwie metody: `setup()` i `takeDown()`. Metoda `setup()` jest wywoływana przez kontener bezpośrednio po utworzeniu agenta w celu wykonania inicjalizacji specyficznych właściwości danego agenta. Symetrycznie, metoda `takeDown()` wywoływana jest po zakończeniu pracy przez agenta, który może np. zwolnić przydzielone zasoby. Kontener uruchamia każdego agenta w osobnym wątku. Nadawanie globalnego identyfikatora, rejestracja w usłudze *White Pages* i inicjacja kolejki, w której przechowywane będą nadchodzące wiadomości są wykonywane automatycznie.

Wymienione metody służą do inicjacji i finalizacji pracy agenta. Gdzie natomiast jest miejsce na wykonywaną przez niego pracę? Otóż w JADE każda praca jaką wykonuje agent powinna być podzielona na rozłączne zadania. Kod realizujący każde zadanie powinien być przez programistę umiesz-



Rysunek 8: Cykl życia agenta

czony w klasie potomnej klasy `jade.core.behaviours.Behaviour`. Każdy agent może mieć określonych wiele zachowań, które może wykonywać quasi-równocześnie tzn. w danej chwili może być realizowane przez wątek agenta tylko jedno zachowanie ale programista może spowodować przełączanie się na inne zachowanie. Może użyć metody `block()` do przeniesienia zachowania do puli nieaktywnych (uśpionych) zachowań (z opcjonalnym parametrem określającym na jaki czas ma być wstrzymane wykonywanie danego zachowania). Symetrycznie można użyć metody `restart()` do przeniesienia zachowania do puli zachowań aktywnych. Agent przełącza się automatycznie między zachowaniami oznaczonymi jako aktywne ale tylko wtedy, gdy obecnie wykonywane zachowanie zakończy swoje działanie lub odda kontrolę agentowi przez wywołanie metody `block()` (zachowania nie podlegają wywłaszczeniu). Istnieje także przypadek, w którym wszystkie zachowania z puli zachowań zablokowanych są przenoszone do aktywnych - przy nadejściu wiadomości adresowanej do danego agenta. Ponadto zachowania mogą być swobodnie dodawane i usuwane podczas działania agenta. Schematycznie cały cykl życia agenta jest przedstawiony na rysunku 8.

Klasy dziedziczące z `Behaviour` powinny implementować dwie metody: `boolean done()` i `action()`. Pierwsza z nich zwraca wartość boolowską określającą czy dane zachowanie się skończyło, natomiast druga metoda `action()` jest wywoływana za każdym razem, gdy zachowanie jest wykonywane przez agenta. Programista tworząc zachowanie nie musi dziedziczyć bezpośrednio z klasy `Behaviour`, może skorzystać z klasy `OneShotBehaviour` (zachowanie jest wykonywane jednokrotnie, metoda `done()` zawsze zwraca `true`) lub `CyclicBehaviour` (metoda `action()` tego zachowania jest wywoływana cyklicznie, ponieważ `done()` zawsze zwraca `false`). Ponadto w bibliotekach JADE dostępnych jest szereg klas ułatwiających tworzenie zachowań bardziej wyspecjalizowanych, posiadających własne zachowania potomne (podzachowania):

- `SequentialBehaviour`: podzachowania wywoływane są sekwencyjnie, zachowanie nadrzędne kończy się tylko wtedy, gdy skończą się wszystkie potomne.

- `ParallellBehaviour`: działa na podobnej zasadzie jak `SequentialBehaviour` ale do zakończenia zachowania nadrzędnego wystarcza, aby jedno zachowanie potomne zostało ukończone (lub w ogólniejszym przypadku N takich zachowań). Jest to przydatne do reprezentowania zadań, które można wyrazić jako kolekcję równoległych, alternatywnych operacji.

- `FSMBehaviour`: Podzachowania są wywoływane w kolejności określonej

przez automat skończony zdefiniowany przez programistę. Stany automatu oznaczone jako terminalne odpowiadają zachowaniom, po których wykonaniu zostanie zakończone całe zadanie nadrzędne.

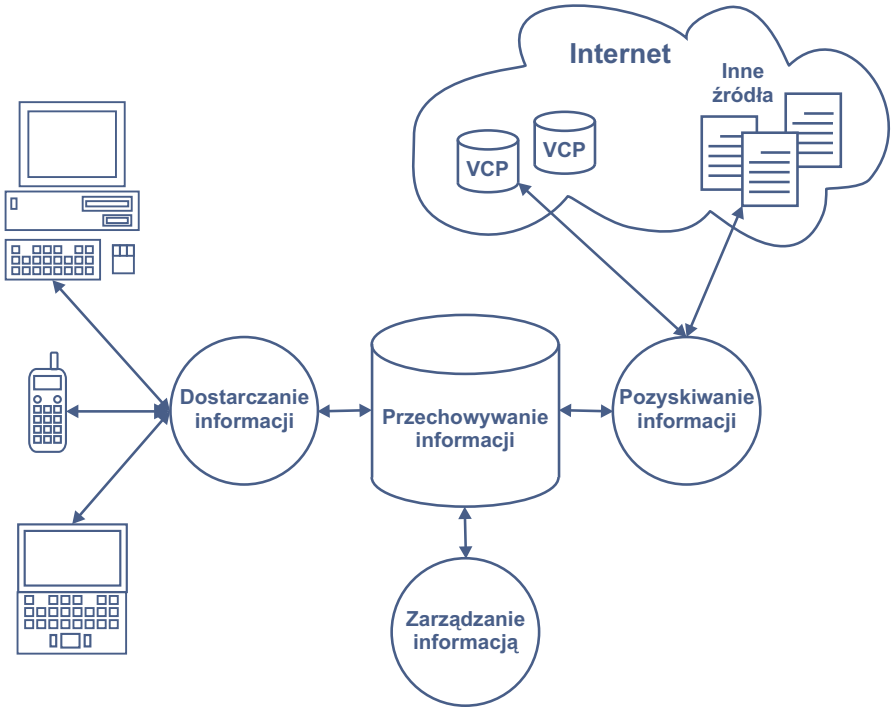
3.3.2 Komunikacja między agentami

Porozumiewanie się agentów uruchomionych na platformie JADE jest bardzo proste. Do wysłania wiadomości wystarczy znajomość identyfikatora agenta (AID), który można uzyskać przez usługi *White Pages* i *Yellow Pages*. Wszystkie wiadomości są przesyłane w języku ACL. Podstawowym językiem używanym w JADE do kodowania wiadomości jest FIPA SL (ang. *FIPA Semantic Language*). Wiadomości zakodowane w FIPA SL są łańcuchami znaków w formie możliwej do odczytania przez człowieka. Wadą tego kodowania jest brak możliwości przesyłania ciągów bajtów (np. nie można przesłać obiektu poddanego serializacji). Drugi kodek został opracowany na potrzeby LEAP (ang. *Light Extensible Agent Platform*), czyli rozszerzenia JADE umożliwiającego pracę z agentami urządzeniom przenośnym. Kodek LEAP zużywa bardzo mało zasobów i koduje wiadomości do postaci binarnej. O ile użycie FIPA SL pozwala potencjalnie na komunikację z agentami innymi niż JADE to LEAP jest specyficzny dla tej platformy. Najciekawszą funkcjonalnością ACL jest jednak możliwość stworzenia własnej ontologii, w której będzie zapisywana treść wiadomości.

Przy nadejściu wiadomości do agenta pozostaje jeszcze do rozwiązania kwestia, które zachowanie powinno ją obsłużyć. Otrzymana wiadomość jest umieszczana w kolejce, w której oczekują nadesłane wiadomości. Następnie wszystkie zachowania są oznaczane jako aktywne. Każde, które spodziewa się nadejścia wiadomości powinno sprawdzić, czy jest ona przeznaczona dla niego. JADE w znacznym stopniu automatyzuje ten proces. Programista używając klasy `MessageTemplate` może stworzyć wzorce wiadomości, jakie będą akceptowane przez dane zachowanie.

4 System wspomagania planowania podróży

W poprzednich rozdziałach omówiłem najważniejsze technologie związane z Siecią Semantyczną oraz systemami wieloagentowymi. Teraz zajmę się pokazaniem korzyści płynących z ich zastosowania, rozważając już konkretny system - PTIS (ang. *Personalized Traveler Information System*). PTIS (opisany szczegółowo w publikacjach [4], [20]) jest w zamierzeniu jego projektantów systemem mającym w sposób całościowy wspierać użytkownika przy



Rysunek 9: System Wspomagania Planowania Podróży

planowaniu podróży. Kompleksowość programu należy rozumieć dwojako. Przede wszystkim w systemie mają uwzględnione być wszystkie aspekty podróżowania (planowanie samej trasy i dobór środków transportu, rezerwacja miejsc w hotelu, organizacja czasu wolnego, wyszukiwanie lokalnych atrakcji, itp.). Równie ważnym elementem jest personalizacja - budowa indywidualnego profilu użytkownika (w maksymalnym stopniu zautomatyzowana). Taki profil jest wykorzystywany w celu lepszego przystosowania informacji prezentowanych użytkownikowi do jego preferencji (zagadnienia związane z personalizacją są szczegółowo omówione w artykule [15]). Informacje przechowywane w systemie są uporządkowane z wykorzystaniem ontologii, zaś wszystkie operacje na nich wykonują wyspecjalizowani agenci. Źródłem danych umieszczanych w systemie jest Internet.

4.1 Podsystemy PTIS

Na rysunku 9 przedstawiony jest całościowy schemat omawianego systemu i wyróżnione są składające się nań podstawowe podsystemy.

4.1.1 Podsystem pozyskiwania informacji

Podsystem pozyskiwania informacji (*Content Collection Subsytem*, CCS) jest odpowiedzialny za komunikację ze źródłami danych i wykonywanie poleceń przeszukiwania tych źródeł. Wyróżniamy dwa podstawowe typy dostawców:

- Zweryfikowanych Dostawców Informacji (ang. *Verified Content Providers*, VCP). Są to źródła danych, które dostarczają rzetelnych, spójnych informacji, są stale dostępne i nie zmieniają swojego formatu danych bez ostrzeżenia. Do tej grupy należą głównie duże portale o ustalonej, stałej strukturze stron WWW oraz źródła udostępniające dane w RDF lub XML (sytuacja idealna).

- Pozostałe, uzupełniające źródła. Informacje przez nie dostarczane często bywają niepełne, format nie jest ściśle określony i przestrzegany. W tej grupie ogół stanowią strony WWW, z których informacje są wydobywane za pomocą specjalnie stworzonych parserów.

Warto zauważyć, że status VCP nie jest przyznawany na stałe, jeżeli jakieś źródło przestaje dostarczać pewnych informacji to konsekwencją jest utrata zaufania i uznania za źródło zweryfikowane. Analogicznie wraz ze wzrostem jakości danych źródło może awansować na VCP. Ponieważ w chwili obecnej dane zapisane w RDF występują w Internecie wciąż bardzo rzadko, więc większość informacji musi być pozyskiwana przez analizę stron WWW i konwersję wyekstrahowanych danych do RDF z uwzględnieniem ontologii wdrożonej w systemie.

4.1.2 Podsystem przechowywania informacji

Wszystkie informacje w systemie są uporządkowane według ontologii i zapisane w formacie RDF. Do przechowywania danych wykorzystywane są modele Jena, z punktu widzenia programisty operacje są więc wykonywane przy pomocy obiektowego API. Pierwotnie system miał jedynie indeksować informacje ale w obecnym podejściu wszystkie dane są pobierane z Internetu i przechowywane lokalnie. Zwiększa to zapotrzebowanie na pamięć potrzebną do składowania danych ale przyspiesza wyszukiwanie (przy każdym zapytaniu nie trzeba odwoływać się do dostawców informacji i przeprowadzać kosztownego procesu łączenia danych). Troska o aktualność danych jest jednak przerzucona na system, konieczne staje się regularne łączenie z dostawcami w celu odświeżenia zawartości.

4.1.3 Podsystem zarządzania informacjami

Główną funkcją podsystemu zarządzania informacjami jest dbanie o aktualność i pełność danych umieszczonych w modelu. Z danymi związane są znaczniki czasowe ostatniej aktualizacji. Jeżeli system stwierdzi, że od ostatniej aktualizacji minęło zbyt wiele czasu to uruchamiany jest proces ponownego ściągania tych informacji od dostawców. Ponadto z obiektami w modelu można związać znaczniki kompletności danych. Nie o każdym obiekcie można zawsze uzyskać pełne informacje posługując się jednym tylko źródłem danych. System wykrywa takie niekompletne opisy i uruchamia dodatkowe procesy wyszukiwania w celu ich uzupełnienia.

4.1.4 Podsystem dostarczania informacji

Rola tego podsystemu wykracza poza proste wykonywanie zapytań i prezentowanie ich wyników. Celem systemu jest bowiem dostarczanie informacji spersonalizowanych. Klienci łączący się z systemem mogą używać różnych urządzeń (komputery klasy PC z przeglądarkami obsługującymi HTML i XHTML, czy też różnego rodzaju PDA i telefony komórkowe obsługujące tylko WAP). Konieczne jest więc dostosowanie wyników do możliwości prezentacyjnych urządzenia. Dzięki zapisowi RDF/XML i wykorzystaniu XSL(T) można w łatwy sposób dokonać przekształcenia do formatu prezentacji akceptowanego przez dane urządzenie.

5 Ontologiczny system katalogowania informacji

Funkcjonalność zaimplementowanego przez mnie systemu obejmuje elementy podsystemów zbierania, gromadzenia i zarządzania informacjami. Przykładową, wdrożoną ontologią jest ontologia obiektów gastronomicznych. Ontologia, opisana w pracach [19] i [18] została opracowana w ramach projektu PTIS [3].

5.1 Reprezentacja danych

Podczas działania programu dane są przechowywane w modelu Jena. W RDF podstawową logiczną jednostką informacji jest zdanie (trójka, ang. *triple*). Natomiast większość algorytmów wewnątrz mojego systemu operuje na opisach zdań składających się z czterech elementów, zwanych dalej quadami.

W skład każdego quada poza elementami zdania RDF (podmiotem, predykatem i obiektem) wchodzi jeszcze identyfikator źródła danych, z którego pochodzi dane stwierdzenie. Fizycznie w modelu Jena quad jest reprezentowany przez dwa zdania. Pierwsze zawiera konkretne informacje o restauracji a drugie jest konstruowane z użyciem reifikacji i służy jedynie jako znacznik do zapamiętania od jakiego dostawcy została ta konkretna informacja uzyskana. Ponadto z opisem każdego obiektu (w tym przypadku restauracji) są związane dodatkowe metadane:

- Lista adresów URL z których uzyskano informacje o danym obiekcie;
- Znacznik czasowy ostatniej aktualizacji;
- Drugi znacznik czasowy oznaczający termin przedawnienia się informacji, po którym należy ponownie odpytać źródła danych i ewentualnie zaktualizować opis obiektu.

5.2 Agenci działający w podsystemie OntoStorage

System składa się z wielu współpracujących agentów. Niektóre ich typy występują jednokrotnie ale część jest uruchamiana w kilku instancjach (są to agenci najczęściej wykorzystywani, których pracę można zrównoleglić). I tak w systemie można wyróżnić następujące typy agentów:

- **Agent Bazodanowy** (`DatabaseAgent`, DBA) odpowiada za wszelkie operacje odczytu i zapisu wykonywane na modelu Jena. W systemie uruchamiany jest jeden agent bazodanowy zajmujący się zapisem do modelu i kilku agentów przeznaczonych do odczytu danych i wykonywania zapytań RDQL. Szerzej agent bazodanowy jest omówiony w punkcie 5.2.1.
- **Agent Nadzorca Spójności Danych** (`DataSupervisonAgent`, DSA) jest agentem odpowiedzialnym za utrzymanie spójności danych. Na podstawie znaczników czasowych związanych z obiektami w modelu agent ustala czy opis tego obiektu jest już nieaktualny. Jeżeli dane zostaną uznane za wymagające sprawdzenia, agent inicjuje i nadzoruje proces aktualizacji (szerzej omówiony w punkcie 5.2.2).
- **Agent Indeksujący** (`IndexingAgent`, IA) porównuje zbiory stwierdzeń i ustala czy reprezentują one ten sam obiekt. Jeżeli tak, to przeprowadza proces łączenia danych z wielu źródeł i rozwiązuje mogące wystąpić konflikty. Szczegóły tego procesu opisane są w paragrafie 5.2.3

- **Agent Koordynator Wyszukiwania Danych** (`SearchCoordinatorAgent`, SCA) kolejkuje zamówienia na wyszukiwanie informacji, utrzymuje pulę współpracujących z nim agentów pracujących bezpośrednio ze źródłami danych. Opisany jest on w punkcie 5.2.4.

- **Agent Dostawca Danych** (`WrapperAgent`, WA) jest agentem odpowiedzialnym za ekstrakcję informacji ze stron WWW dostawców danych. W obecnej implementacji wdrożone są dwa parsery stron: ChefMoz i restauracje.com. W systemie działa jednocześnie kilku agentów WA, ich praca jest koordynowana przez SCA. Więcej informacji o dostawcach danych znajduje się w punkcie 5.2.5.

- **Główny Agent Nadzorca** (`ManagerAgent`, MA) jest agentem nadzorującym pracę całego systemu. MA przy uruchamianiu systemu inicjuje pozostałych agentów a także rozsyła żądania zakończenia pracy w momencie, gdy cały system jest zamykany. Przyjmuje polecenia od agenta GUI i rozsyła je do odpowiednich agentów. Zakres zadań realizowanych przez MA przedstawiony jest w punkcie 5.2.6.

- **Agenci Współpracy z Interfejsem Użytkownika** - w systemie znajdują się dwaj tacy agenci, pełniący rolę pośrednika między interfejsem graficznym, którym posługuje się użytkownik a systemem wieloagentowym. Z jednej strony reagują na zdarzenia związane z wybraniem opcji w menu czy na pasku narzędziowym i wysyłają odpowiednie komunikaty do MA. Z drugiej strony odbierają wiadomości od MA i odpowiednio do ich zawartości zmieniają stan GUI. Sam interfejs jest stworzony przy pomocy biblioteki SWT (ang. *Standard Widget Toolkit*, [51]), wyprodukowanej na potrzeby środowiska eclipse.

5.2.1 Agent bazodanowy (DBA)

Agent bazodanowy jest jedynym agentem mającym bezpośredni dostęp do modelu Jena. Wszelkie operacje zapisu lub odczytu danych muszą się odbywać za jego pośrednictwem. Dzięki temu reszta agentów jest odizolowana od fizycznej formy zapisu RDF. W systemie działają dwa typy agentów bazodanowych - pojedyncza instancja agenta wykonującego operacje zapisu i kilka instancji agentów wykonujących jedynie operacje wymagające odczytu danych (wyszukiwanie pasującego obiektu, ekstrakcja informacji o konkretnym obiekcie, wykonywanie kwerend RDQL, ekstrakcja opisów wymagających uaktualnienia).

Zapis. Agent zlecający DBA zapis danych wysyła po prostu cały opis obiektu w postaci listy quadów. DBA sam decyduje, czy otrzymany obiekt jest nowy i należy go w całości dodać do modelu, czy też wystarczy jedynie zaktualizować umieszczone wcześniej informacje. Po otrzymaniu zlecenia agent najpierw wyszukuje w modelu pasujący obiekt (czyli zasób reprezentowany przez odpowiednie URI) i jeżeli go znajdzie to następuje aktualizacja. Agent sprawdza, które zdania należy usunąć a które dodać i wykonuje minimalną liczbę operacji konieczną do wykonania aktualizacji. Jeżeli jest to nowy opis nie mający odpowiednika w grafie to wszystkie quady są dodawane do modelu.

Ekstrakcja informacji o konkretnym obiekcie. Parametrem jest przy tej operacji URI reprezentujące zasób. DBA najpierw wyciąga wszystkie stwierdzenia, w których dany zasób jest użyty jako podmiot a następnie bada reifikacje związane z tymi trójkami aby powiązać dostawców informacji ze zdaniami. DBA wykonuje konwersję tych informacji do postaci quadów i odsyła tak skonstruowaną listę do agenta, który zlecił zapytanie.

Wyszukiwanie pasującego obiektu. Agent otrzymuje listę quadów opisujących obiekt i stara się znaleźć w modelu zasób, którego opis odpowiada zdefiniowanemu wzorcowi. Dopasowywanie odbywa się w sposób zależny od ontologicznej klasy, której instancją jest wyszukiwany obiekt. Dla każdej klasy w ontologii muszą istnieć dwie klasy w Javie implementujące odpowiednie interfejsy:

- `IObjectInfoTag`. System zakłada, że każdy obiekt posiada "znacznik obiektu" czyli jedną lub kilka charakterystycznych właściwości jednoznacznie go identyfikujących (w przypadku restauracji taką rolę pełni zestaw: nazwa, miasto, adres i kod pocztowy). Interfejs deklaruje dwie metody:

  ```
  public interface IObjectInfoTag
  {
    public IObjectInfoTag getTag (QuadsList quads);

    public boolean isComplete ();
  }
  ```

  Metoda `getTag` służy do ekstrakcji takiego znacznika z całego opisu. Aby wyszukiwanie było możliwe ten znacznik musi być kompletny, natomiast metoda `isComplete` weryfikuje właśnie jego kompletność.

- `IObjectInfoSearcherByTag`. Realizuje proces wyszukiwania dla pojedynczej klasy ontologicznej na podstawie dostarczonego znacznika obiektu. Definiuje tylko jedną metodę `find` służącą do tego celu:

  ```
  public interface IObjectInfoSearcherByTag
  {
    public QuadsList findObject (IOntModelWrapper wrapper,
       IObjectInfoTag tag) throws OntPropertyNotFoundException;
  }
  ```

  Dzięki dedykowanej implementacji dla każdej klasy ontologicznej uzyskiwana jest większa elastyczność (np. w ten sposób można uwzględnić, że ten sam adres może być reprezentowany przez różne łańcuchy znaków).

W zależności od tego, czy wyszukiwanie zakończyło się sukcesem czy porażką odsyłany jest komunikat o błędzie, komunikat o nie znalezieniu pasującego obiektu lub pełny opis dopasowanego obiektu w postaci listy quadów.

Wykonywanie kwerend RDQL. Wykonywanie zapytań RDQL jest bardzo proste, gdyż praktycznie cała praca jest wykonywana przez dedykowane do tego celu klasy Jena. Rezultaty kwerendy są również formatowane przy użyciu standardowego formatera dostarczonego przez Jena i odsyłane do zleceniodawcy.

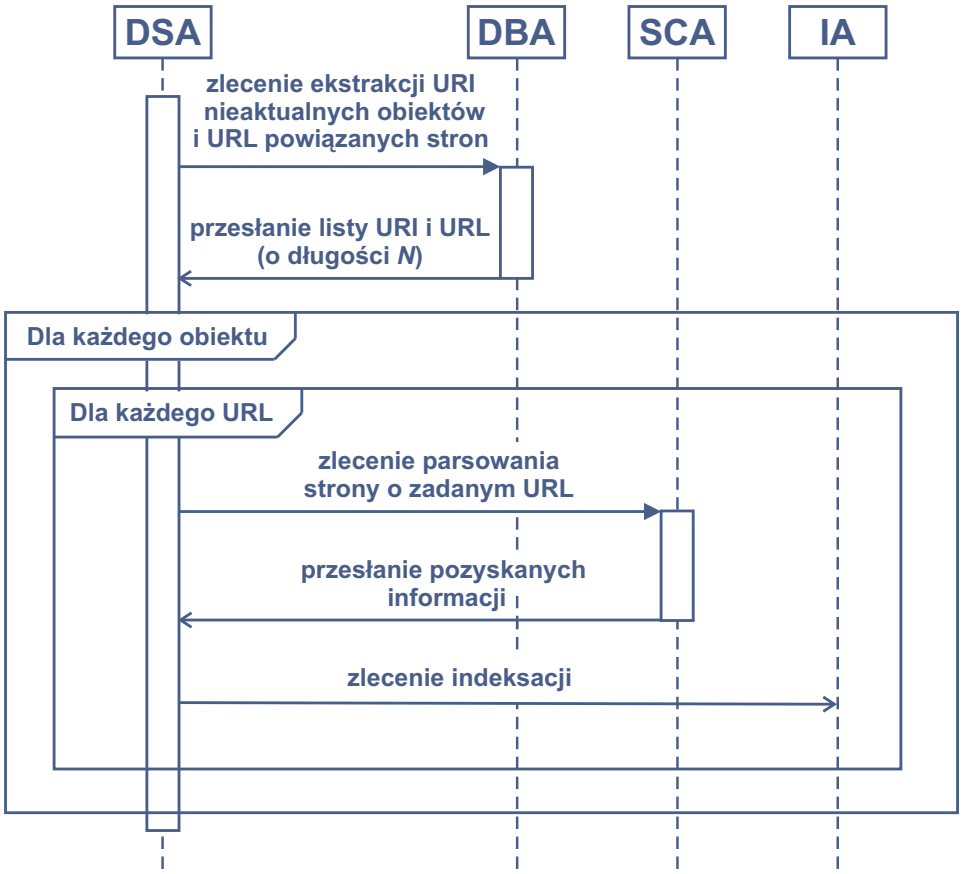
Ekstrakcja opisów wymagających uaktualnienia. Agent sprawdza, dla których obiektów znaczniki utraty aktualności są wcześniejsze niż chwila obecna. Dla każdego nieaktualnego obiektu wyciąga URL stron, z których pochodziły informacje składające się na jego opis. Lista tych adresów jest następnie odsyłana do agenta, który zlecił wyszukiwanie.

5.2.2 Agent Nadzorca Spójności Danych (DSA)

DSA jest odpowiedzialny za utrzymywanie spójności danych. Jest kilka aspektów tej spójności:

Aktualność danych. Jak już wspomniałem, z opisem każdego obiektu umieszczonego w modelu związane są dwa znaczniki czasowe (data ostatniej aktualizacji i wyznaczony termin ważności informacji). Po otrzymaniu od MA polecenia rozpoczęcia aktualizacji [1] DSA inicjuje ten proces. Schemat

[1] aktualizacja uruchamiana jest z poziomu GUI aby można było łatwo zaobserwować zachodzące zmiany ale w pełnej implementacji DSA sam okresowo inicjowałby ten proces



Rysunek 10: Diagram interakcji w procesie aktualizacji informacji

interakcji w procesie aktualizacji jest przedstawiony na diagramie 10. DSA wysyła do DBA polecenie wyekstrahowania URI identyfikujących obiekty, które są nieaktualne (tzn. znacznik terminu ważności informacji wskazuje na datę wcześniejszą niż obecna). Następnie zleca SCA ponowne połączenie z dostawcami informacji w celu sparsowania stron WWW, z których wcześniej uzyskano informacje na temat tych obiektów. Po otrzymaniu świeżych informacji są one wysyłane do indeksacji. Zaimplementowany jest również prosty sposób adaptacji przedziału czasowego między kolejnymi aktualizacjami danych. Jeżeli dane nie zmieniły się od czasu ostatniego sprawdzenia, to przedział jest wydłużany dwukrotnie, w przeciwnym przypadku skracany o połowę. Działanie to ma na celu dostosowanie poszukiwania nowych danych do zmian zachodzących w repozytoriach z których owe dane pochodzą. Ponieważ każdorazowe odpytanie repozytorium jest zasobochłonne, a ilość takich odpytań wzrasta lawinowo wraz z ilością przechowywanych w systemie danych, więc celem proponowanego tutaj rozwiązania jest adaptacyjna

minimalizacja liczby wykonywanych zapytań.

Kompletność danych. Używane przez system ontologie mogą umożliwiać bardzo szczegółowe charakteryzowanie zasobów. Sytuacja, w której wykorzystanie tylko jednego źródła danych pozwoli na skonstruowanie kompletnego opisu będzie więc stosunkowo rzadka. Przy większej ilości źródeł danych celowe byłoby zastosowanie specjalnych znaczników wskazujących na kompletność lub częściowość opisów. Aktualizacja statusu kompletności następowałaby przy każdej zmianie zdań charakteryzujących zasób. Znacznik byłby wykorzystywany w sposób analogiczny jak znacznik czasowy aktualizacji. DBA dokonywałby ekstrakcji obiektów oznaczonych jako niekompletne a DSA sprawdzałby (za pośrednictwem SCA i WA), czy inne źródła danych posiadają jakieś informacje o konkretnym obiekcie i w razie ich znalezienia wysyłałby dane do indeksacji.

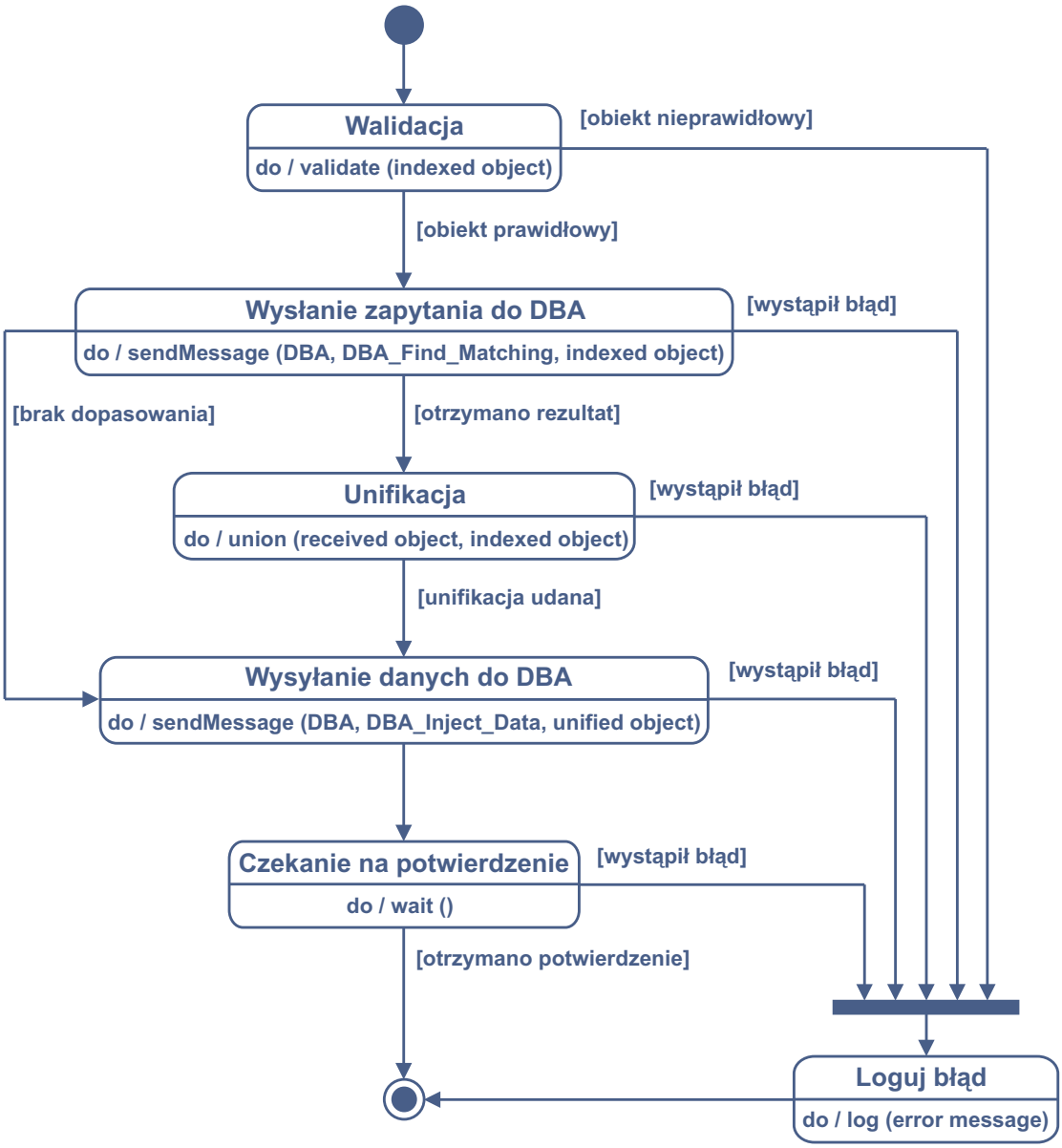
5.2.3 Agent Indeksujący (IA)

W systemie działa tylko jedna instancja IA (w przeciwnym przypadku mogłoby dojść do sytuacji, w której ta sama restauracja byłaby dodana wielokrotnie do modelu). W ramach tego agenta działają dwa zachowania cykliczne, pełniące rolę planistów i od zera do N zachowań przeprowadzających proces indeksowania, gdzie N jest określane przez administratora systemu w pliku konfiguracyjnym w znaczniku `NumberMaxIndexingBehaviours`. Zachowania planistyczne to:

- `ServiceIABehaviour` - oczekuje na zlecenia na przeprowadzenie indeksowania. Otrzymawszy takie zlecenie umieszcza je na końcu kolejki FIFO i budzi wątek planisty zadań.
- `SchedulerBehaviour` - rozplanowuje zadania indeksowania. Okresowo sprawdza, czy w kolejce są jakieś zadania do wykonania (planista może być także zbudzony przez inne zachowanie). Jeżeli liczba wykonywanych w danej chwili procesów indeksowania nie przekracza wyznaczonego maksimum, to dodaje nowe zachowanie `IndexingBehaviour` odpowiedzialne za wykonanie zadania.

Najważniejszym zachowaniem jest `IndexingBehaviour`, które przeprowadza proces indeksacji pojedynczego obiektu. Schemat tego procesu jest przedstawiony diagramie 11, omówię teraz jego najważniejsze etapy.

Walidacja. Najpierw wykonywane jest sprawdzenie, czy dostarczony opis obiektu jest ważny. Dla każdej klasy w ontologii powinien być zdefiniowany



Rysunek 11: Diagram stanu zachowania IndexingBehaviour

zestaw zasad określający, które jej instancje należy uznać za ważne. Zasady te administrator umieszcza w podkatalogu `/xml` w pliku XML o nazwie takiej samej jak nazwa klasy. Fragment przykładowego pliku z zasadami walidacji jest przedstawiony na listingu poniżej:

```
<ValidationRules>
  <Rule property="http://www.agentlab.net/schemas/location#city">
    <Content significance="essential" plurality="singular"/>
  </Rule>
</ValidationRules>
```

Korzeniem dokumentu jest węzeł `ValidationRules`. Każdej właściwości odpowiada jeden węzeł `Rule` z atrybutem `property` zawierającym URI identyfikującym tę właściwość w ontologii. Węzeł `Content`, potomny względem `Rule` ma dwa atrybuty:

- `significance` określa jak ważna jest dana właściwość ze względu na kompletność opisu obiektu. Może przyjąć jedną z trzech wartości:
  - `essential` - obiekt musi mieć określoną wartość dla tej właściwości, bo w przeciwnym przypadku zostanie uznany za niekompletny w stopniu uniemożliwiającym jego sensowne użycie (dla restauracji takimi właściwościami są nazwa, adres, miasto i kod pocztowy - podstawowe właściwości umożliwiające identyfikację lokalu).
  - `required` - obiekt powinien mieć określone wartości dla wszystkich właściwości oznaczonych jako `required` aby system mógł uznać opis obiektu za kompletny. Obiekty mające określone wszystkie właściwości oznaczone jako `essential` ale niekompletne będą normalnie dodane do bazy i używane przy raportowaniu wyników zapytań. Informacja o niekompletności mogłaby być wykorzystywana do uruchomiania procesu wyszukania brakujących danych.
  - `optional` - właściwości opcjonalne, nie mają wpływu na to, czy opis obiektu zostanie uznany za kompletny.
- `plurality` określa dozwoloną krotność danej właściwości w opisie obiektu. Rozróżnienie jest najprostsze z możliwych tzn. wyróżniamy właściwości, które mogą wystąpić co najwyżej raz (`singular`) i takie, których krotność jest dowolna (`multiple`). Walidacja obiektu zakończy się niepowodzeniem w przypadku, gdy instancja będzie miała wielokrotnie użytą właściwość oznaczoną jako `singular`.

Unifikacja. Pod tym pojęciem kryje się proces łączenia dwóch opisów tego samego obiektu z rozwiązywaniem mogących wystąpić konfliktów. Przy projektowaniu fragmentu oprogramowania realizującego to zadanie dążyłem do stworzenia algorytmu, który byłby niezależny od używanej ontologii restauracji, aby możliwe było dodawanie kolejnych ontologii małym kosztem. Większość procesu jest sterowana przez odpowiedni plik XML ale okazało się, że niektóre aspekty rozwiązywania konfliktów wymagają dedykowanego kodu. Tym niemniej odwołania do tego kodu są jasno określone w pliku XML. Przystąpię więc do omawiania jego struktury i dostępnych funkcji. Analogicznie jak w przypadku pliku z zasadami walidacji, plik XML powinien nazywać się tak samo jak nazwa klasy w ontologii i dla każdej właściwości powinien być zdefiniowany element `Rule` z odpowiednim atrybutem `property` (korzeniem jest tutaj element o nazwie `MergingRules`). Węzeł `Rule` ma również atrybuty `significance` i `singular`, których znaczenie jest dokładnie takie samo, jak w przypadku atrybutów węzła `Content`. Węzły potomne węzła `Rule` sterują sposobem łączenia danych z wielu źródeł. Dla każdej właściwości w zależności od jej charakteru możemy mieć do czynienia z jednym z opisanych dalej scenariuszy.

Właściwości, które muszą mieć takie same wartości. W ontologii restauracji zaliczają się do nich nazwa restauracji, jej adres, miasto i kod pocztowy. Przykładowo zasada dla miasta, w którym znajduje się restauracja jest przedstawiona na listingu poniżej:

```
<Rule property="http://www.agentlab.net/schemas/location#city"
        significance="essential" plurality="singular"
 >
  <RequireMatch/>
  <ProviderSignificance strict="false">
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#CM"/>
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#RC"/>
  </ProviderSignificance>
</Rule>
```

Miasto, w którym zlokalizowana jest restauracja jest informacją wymaganą (`significance="essential"`). Restauracja nie może się jednocześnie znajdować w wielu miejscowościach, więc należy wymusić jednokrotne wystąpienie właściwości `city` przez ustawienie `plurality="singular"`. Umieszczenie węzła potomnego `RequireMatch` powoduje, że algorytm łączący będzie wymagał, aby oba łańcuchy znakowe reprezentujące nazwę miasta były identyczne.

Ponieważ z każdym zdaniem w modelu związany jest tylko jeden znacznik o dostawcy informacji, więc konieczne jest rozstrzygnięcie znacznikiem

którego dostawcy będzie ostemplowane każde zdanie. Do określenia hierarchii ważności dostawców informacji dla danej właściwości służy znacznik `ProviderSignificance`. Węzłami potomnymi `ProviderSignificance` są węzły o nazwie `Provider` z atrybutem `id` równym URI identyfikującym dostawcę informacji. Kolejność w jakiej umieszczone są węzły potomne odpowiada ważności w hierarchii. W tym przykładzie informacja z ChefMoz jest ważniejsza niż informacja z restauracje.com, więc nazwa miejscowości będzie oznaczona jako uzyskana z ChefMoza.

W przypadku adresu okazało się, że wykonanie prostego porównania łańcuchów znakowych nie jest wystarczające, gdyż ten sam adres może być reprezentowany przez wiele różnych napisów. Dlatego na listingu poniżej węzeł `RequireMatch` ma dodatkowy atrybut `class`.

```
<Rule property="http://www.agentlab.net/schemas/location#streetAddress"
      significance="essential" plurality="singular">
  <RequireMatch class="logrus.ontmodel.restaurant.tools.AddressMatcher"/>
  <ProviderSignificance strict="false">
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#CM"/>
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#RC"/>
  </ProviderSignificance>
</Rule>
```

Atrybut `class` to w pełni kwalifikowana nazwa klasy Javy, która będzie realizowała porównanie adresów. Program tworzy instancję tej klasy z użyciem metod refleksji Javy i zakłada, że jest to klasa implementująca interfejs `IPropertyMatcher`:

```
public interface IPropertyMatcher
{
  public boolean match (String str1, String str2);
}
```

Jedyna jego metoda `match` służy właśnie do specyficznego porównywania właściwości (w tym przypadku adresu).

Właściwości z wielokrotnymi wartościami. Niektóre właściwości (takie jak rodzaj kuchni, do jakiej należą dania podawane w restauracji) mogą występować w opisie obiektu wielokrotnie. Oprogramowanie powinno połączyć więc dwie listy pochodzące od dwóch dostawców w jedną. Listing poniżej przedstawia definicję zasady łączenia dla takiego przypadku.

```
<Rule property="http://www.agentlab.net/schemas/restaurant#cuisine"
      significance="required" plurality="multiple"
 >
```

```
  <MergeMultiple/>
  <ProviderSignificance strict="false">
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#RC"/>
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#CM"/>
  </ProviderSignificance>
</Rule>
```

Pojawia się nowy znacznik `MergeMultiple` mówiący oprogramowaniu, że ma dokonać łączenia (ma to oczywiście sens jeżeli `plurality="multiple"`). Kolejność określana przez `ProviderSignificance` rozstrzyga niejasność w przypadku, gdy taka sama wartość właściwości występuje u obu dostawców.

Właściwości wyjątkowe. Dla niektórych właściwości istnieje tylko jeden dostawca informacji, można to oznaczyć w pliku XML przy pomocy znacznika `ExclusiveProvider` z atrybutem `id` ustawionym na URI identyfikujące dostawcę:

```
<Rule property="http://www.agentlab.net/schemas/restaurant#capacity"
      significance="optional" plurality="singular">
  <ExclusiveProvider id="http://www.agentlab.net/schemas/ontoStorage#RC"/>
</Rule>
```

Właściwości o wartościach ogólnych i szczegółowych. Niektóre właściwości (jak np. stopień dostępności restauracji dla osób niepełnosprawnych) mogą przybierać wartości o znaczeniu ogólnym (np. `res:SomeAccessibility` oznacza, że restauracja jest w jakimś bliżej nieokreślonym stopniu dostępna dla niepełnosprawnych) i szczegółowym (np. `res:CompletelyAccesible` - pełna dostępność czy `res:PartiallyAccessible` - częściowa dostępność). Chcemy aby system brał pod uwagę ogólne informacje tylko w przypadku, gdy nie są dostępne informacje szczegółowe. Służy do tego znacznik `LowSignificanceData`. Elementami potomnymi tego węzła są znaczniki `Object` z atrybutem `value` ustawionym na URI reprezentujące obiekt, który chcemy uznać za mało ważny tzn. taki, który nie zostanie uwzględniony, jeżeli będą dostępne informacje szczegółowe.

```
<Rule property="http://www.agentlab.net/schemas/restaurant#accessibility"
      significance="required" plurality="singular">
  <ProviderSignificance strict="false">
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#CM"/>
    <Provider id="http://www.agentlab.net/schemas/ontoStorage#RC"/>
  </ProviderSignificance>
  <LowSignificanceData>
    <Object
value="http://www.agentlab.net/schemas/restaurant#UnknownAccessible"/>
```

```
    <Object
value="http://www.agentlab.net/schemas/restaurant#SomeAccessibility"/>
  </LowSignificanceData>
</Rule>
```

Właściwości dekonfliktowane przez dedykowany kod. Wymienione wcześniej proste zasady sterujące są w większości przypadków wystarczające do opisania sposobu łączenia informacji. Jeżeli nie uda się skonstruować zasady przy pomocy zdefiniowanych znaczników XML, można zaimplementować ten proces dla danej właściwości i odnotować to w pliku w następujący sposób:

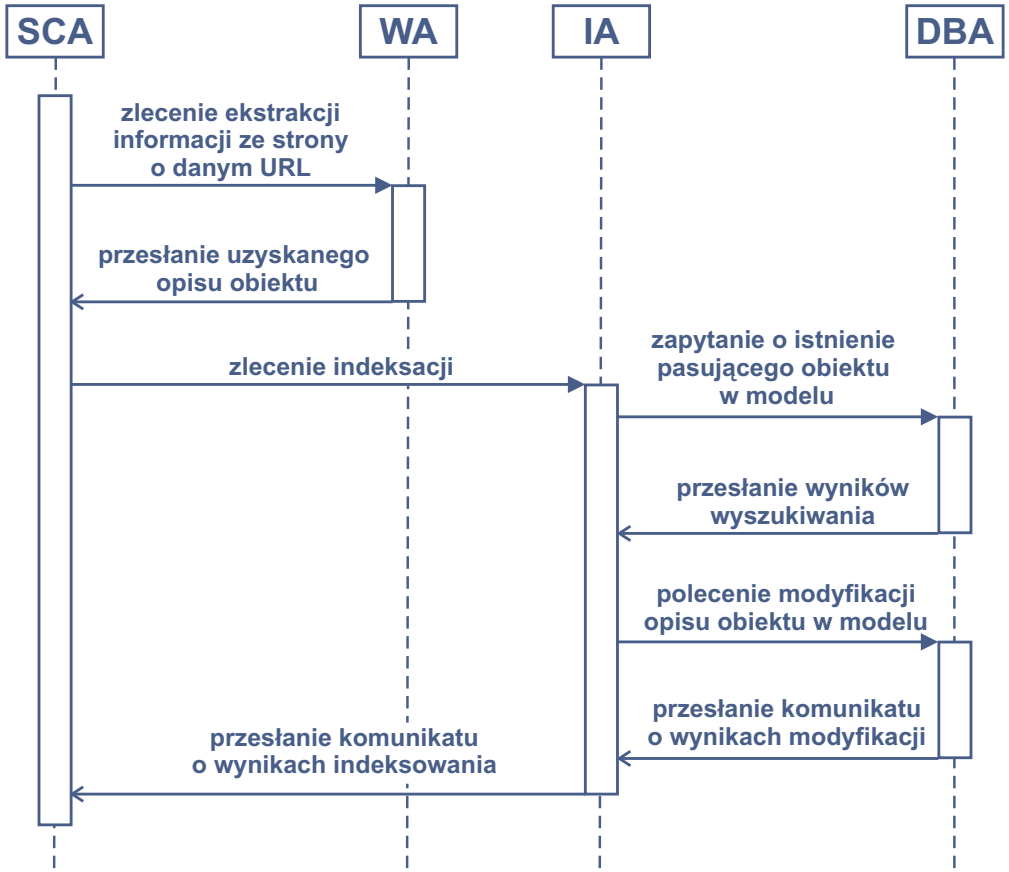
```
<Rule property="http://www.agentlab.net/schemas/restaurant#parking"
      significance="required" plurality="singular"
 >
  <JavaClassRule
class="logrus.ontmodel.restaurant.tools.ParkingMergingRule"/>
</Rule>
```

Znacznik `JavaClassRule` oznacza, że cały proces łączenia będzie przeprowadzony przez instancję klasy o pełnej, kwalifikowanej nazwie wziętej z atrybutu `class`. Instancja będzie utworzona przy pomocy metod refleksji Javy i program zakłada, że implementuje ona interfejs `ICustomMerging`.

Takie łączenie jest przeprowadzane dla właściwości określającej akceptowalne formy płatności. Kody określające formy płatności są zarówno ogólne (różne znaczniki określające możliwości płacenia poszczególnymi typami kart: bankowymi, kredytowymi itp.) jak i szczegółowe (można płacić kartą konkretnego banku). Zawieranie się poszczególnych klas ogólnych i występowanie przypadków szczególnych sprawiają, że zdefiniowanie reguł łączenia przy pomocy przedstawionego zestawu znaczników jest bardzo trudne. Wykorzystałem więc w tym przypadku znacznik `JavaClassRule` i zaimplementowałem cały proces w klasie `PaymentMergingRule`.

5.2.4 Agent Koordynator Wyszukiwania Danych (SCA)

Wszystkie polecenia wyszukania danych u dostawców informacji lub ich ekstrakcji z konkretnego URL są w systemie kierowane do SCA. Ten kolejkuje zamówienia w prostej kolejce FIFO i rozdziela je do puli współpracujących z nim WA. Maksymalna ilość agentów WA jednocześnie działających w systemie jest określana w pliku konfiguracyjnym w znaczniku `NumberWA`. SCA obsługuje dwa typy zadań do wykonania: zlecenie na masowe wyciąganie informacji od dostawców (tryb "pająka") i wyciąganie informacji z pojedynczej strony WWW.



Rysunek 12: Diagram interakcji w procesie parsowania strony WWW

Tryb pająka. W pierwszym etapie SCA zleca WA ekstrakcję samych łącz do indywidualnych stron z informacjami o restauracjach. Po otrzymaniu listy łącz SCA przystępuje do zrównoleglenia procesu zbierania informacji. Parsowanie stron WWW może odbywać się już jednocześnie. Schematycznie proces nadzoru nad parsowaniem pojedynczej strony i wymianą wiadomości między agentami jest przedstawiony na diagramie 12. Szczegóły procesu indeksacji są opisane w punkcie 5.2.3 a operacji wykonywanych przez agenta bazodanowego w punkcie 5.2.1.

Parsowanie pojedynczej strony WWW. Nadzór nad tą operacją ze strony SCA jest analogiczny jak na diagramie 12.

5.2.5 Agent Dostawca Danych (WA)

Agenci WA pełnią rolę pośrednika między systemem wieloagentowym a dostawcami informacji. W mojej implementacji to pośrednictwo polega na parsowaniu stron WWW ChefMoz i restauracje.com oraz transformacja uzyskanych informacji do RDF. WA obsługuje dwa polecenia: parsowanie konkretnej strony WWW w celu uzyskania informacji o pojedynczej restauracji oraz przeszukania podstron zawierających indeksy w celu ekstrakcji łącz do indywidualnych stron restauracji.

Parsowanie konkretnej strony WWW. WA otrzymuje od SCA polecenia parsowania strony WWW i nazwę powiązanego z nią dostawcy. Następnie przystępuje do próby ekstrakcji informacji i jeżeli zakończyła się ona sukcesem to odsyła wynik w postaci kolekcji quadów. Przyczyny niepowodzenia działania parsera mogą być różne:

- Niedostępność strony WWW;
- Błąd parsera spowodowany zmianą wyglądu strony i w konsekwencji innego rozmieszczenia elementów w drzewie HTML;
- Odrzucenie uzyskanego opisu ze względu na niekompletność danych w stopniu uniemożliwiającym ich wykorzystanie;
- Błędy w danych lub ich nieznana postać.

W razie napotkania błędu informacja o nim jest zapisywana do logów i WA informuje SCA o nieudanym parsowaniu.

Najczęstszą przyczyną odrzucenia danych w przypadku parserów stron związanych z restauracjami są braki w adresie (brak kodu pocztowego, nazwy miasta lub adresu ulicy). Ponadto na www.restauracje.com adresy nie są weryfikowane przez edytorów tylko wpisywane bezpośrednio przez użytkowników i w związku z tym liczba formatów, w jakich są one przechowywane jest bardzo duża. Parser stara się odgadnąć, które nazwy odpowiadają poszczególnym elementom adresu ale nie zawsze jest to możliwe (na ilustracji 13 brakuje po prostu kluczowych elementów jak ulica i kod pocztowy, parser zostaje też "zaskoczony" informacją o gminie).

Randomizacja danych. W praktyce przetestowanie funkcji aktualizacji danych było trudne, gdyż informacje o restauracjach ulegają zmianie bardzo rzadko. Dlatego w celu przetestowania działania systemu zaimplementowana została specjalna funkcja dzięki której agenci WA mogą dokonywać losowych

Rysunek 13: Przykład nieakceptowalnego adresu

zmian w danych uzyskanych od dostawców. To czy informacje będą zmodyfikowane zależy od administratora i wartości flagi `UseDataRandomization` w pliku konfiguracyjnym aplikacji (konfiguracja opisana jest w paragrafie 5.3.1).

5.2.6 Główny Agent Nadzorca (MA)

MA jest agentem koordynującym pracę całego systemu i odpowiedzialnym za jego uruchamianie. Pierwszym jego zadaniem po uruchomieniu jest podniesienie reszty systemu. MA tworzy pozostałych agentów (poza agentami związanymi z GUI) i dodaje ich do kontenera JADE. Następnie czeka na potwierdzenia o pomyślnej inicjalizacji i w momencie kiedy nadejdą one od wszystkich agentów wysyła do agenta GUI komunikat o pomyślnym uruchomieniu systemu. Dzięki zastosowaniu MA jako pośrednika między agentami współpracującymi bezpośrednio z GUI a resztą agentów możemy wprowadzać niezależnie zmiany zarówno w GUI jak i w agentach pracujących z danymi w RDF (o ile nie zmienia się sposób komunikacji między MA a agentami GUI).

5.3 Korzystanie z systemu

5.3.1 Plik konfiguracyjny

Administrator ma możliwość ustalenia szeregu parametrów systemu przez wyspecyfikowanie ich w pliku konfiguracyjnym. Przykładowy plik konfiguracyjny przedstawiony jest na listingu poniżej:

```
<?xml version="1.0" encoding="windows-1250"?>
<Settings>
   <NumberReadDBA value="2"/>
   <NumberWA value="2"/>
   <NumberMaxIndexingBehaviours value="2"/>
   <InitialTermOfValidity value="1"/> <!-- In minutes -->
   <UseDataRandomization value="true"/>
```

```
</Settings>
```

Znaczenie poszczególnych elementów jest następujące:

- `NumberReadDBA` - Atrybut `value` określa liczbę jednocześnie uruchomionych agentów bazodanowych zajmujących się odczytem danych z modelu (ponadto zawsze uruchamiany jest jeszcze jeden DBA zajmujący się tylko operacjami zapisu i modyfikacji danych).
- `NumberWA` - Atrybut `value` określa liczbę jednocześnie uruchomionych agentów WA zajmujących się uzyskiwaniem informacji od dostawców (czyli parsowaniem stron WWW).
- `NumberMaxIndexingBehaviours` - Atrybut `value` określa maksymalną liczbę aktywnych zachowań `IndexingBehaviour` dla agenta IA.
- `InitialTermOfValidity` - Atrybut `value` określa początkowy czas między aktualizacjami danych ("okres ważności" danych). Okres ten jest wyrażony w minutach.
- `UseDataRandomization` - Atrybut `value` przyjmuje wartości boolowskie i określa, czy agenci WA mają "zakłócać" dane aby zasymulować zmiany zachodzące z czasem w źródłach danych.

5.3.2 Interfejs użytkownika

Po uruchomieniu programu użytkownik wskazuje plik RDF, z którego będzie wczytany model. Jeżeli nie zostanie wybrany żaden plik to system wczyta trójki z pliku "default.rdf", zawierającego jedynie stwierdzenia opisujące ontologię restauracji. Po inicjalizacji wszystkich agentów uaktywnia się szereg opcji w menu i na pasku narzędziowym. Większość akcji inicjowanych z poziomu GUI ma charakter asynchroniczny. Agent GUI wysyła zlecenie do odpowiedniego, wyspecjalizowanego agenta, który po zakończeniu operacji wysyła informację zwrotną do GUI (pośredniczy w tym agent MA). Użytkownik jest informowany o wynikach działań w wyświetlanym oknie dialogowym.
Interfejs użytkownika pozwala na wykonanie następujących operacji na systemie:

- Pozyskiwanie danych od dostawców informacji. Jest realizowane po wybraniu opcji z podmenu `Action` → `Crawl`. Uruchamiane jest przeszukiwanie podstron znajdujących się w ChefMozie i restauracje.com, które zawierają informacje o Warszawie lub Krakowie (przeszukiwanie WWW i indeksacja pozyskanych informacji trwa kilkanaście minut).

- Wyświetlenie podstawowych informacji o modelu. Po wybraniu opcji `Tools` → `Send build report request` inicjowany jest proces zbierania informacji o liczbie opisów restauracji w modelu, ile z nich zostało zbudowanych w oparciu o łączone dane i ile ogółem znajduje się trójek w modelu. Po nadejściu raportu wyświetlany jest on w oknie dialogowym.

- Zapisanie modelu do pliku. Po wybraniu `Tools` → `Save model` wyświetlane jest standardowe okno dialogowe zapisu pliku, użytkownik wybiera gdzie ma być zapisany zrzut aktualnego stanu modelu Jena (w postaci możliwego do załadowania potem dokumentu RDF/XML).

- Wysyłanie zapytań w języku RDQL. Opcja `Tools` → `RDQL query` uruchamia dodatkowe okno dialogowe, w którym można wpisać zapytanie RDQL (lub wczytać je z wcześniej przygotowanego pliku). Po wysłaniu zapytania i jego przetworzeniu wyniki są wyświetlane w formie tabeli z przyporządkowaniami wyników do poszczególnych zmiennych występujących w zapytaniu.

- Inicjalizacja aktualizacji. Wybór opcji w menu `Tools` → `Initiate update process` powoduje wysyłanie do DSA żądania rozpoczęcia procesu aktualizacji opisanego w punkcie 5.2.2.

5.4 Podsumowanie

Zaimplementowany przeze mnie fragment systemu PTIS działał w sposób zadowalający przy ponad 750 opisach restauracji, z czego niemal 60 było skonstruowanych przez połączenie informacji uzyskanych z ChefMoza i restauracje.com. Łączna liczba trójek RDF wynosiła około 130000. Stosunkowo długi był jedynie czas inicjalizacji systemu, wymuszony przez konstrukcję modelu Jena ze stwierdzeń RDF wczytywanych z pliku. Najbardziej czasochłonną i jednocześnie najbardziej wrażliwą częścią systemu są parsery stron WWW. Każda zmiana wyglądu strony pociąga bowiem za sobą konieczność gruntownej przebudowy analizatora HTML. Ze względu na niekompletne dane lub ich nieuporządkowany format ze strony restauracje.com mogło być wykorzystane tylko około 50% opisów restauracji. Utylizacja danych z Chefmoza była znacznie wyższa i potencjalnie opisy mogły być bardziej szczegółowe ale znalazłem tylko jedną restaurację, której opis wykorzystywał w pełni ontologię zdefiniowaną przez Chefmoza.

Wydaje się, że praktyczne zastosowanie systemu nie będzie możliwe dopóki RDF nie zostanie rozpowszechniony i szeroko wykorzystywany a dane w

tym formacie upublicznione. Nawet zastosowanie przez dostawców informacji różnych ontologii nie stanowi dużego problemu. Dzięki zapisowi RDF/XML można np. wykorzystać XSLT do szybkiej transformacji danych bez konieczności pisania dedykowanego kodu. Jeżeli jednak zajdzie potrzeba stworzenia pewnych procedur, to będzie ich nieporównywalnie mniej niż w przypadku całych parserów stron i wyraźnie wzrośnie przejrzystość kodu.

6 Bibliografia

Literatura

[1] van Aart Chris, Pels Ruurd *Creating and Using Ontologies in Agent Communication*, EXP, 2002, volume 2, issue 3.

[2] Aciar Silvana, López Herrera Josefina *A Multi-Agent System to select information sources based on intrinsic characteristics*

[3] AgentLab - Travel Support Project: http://www.agentlab.net/projects/e-Travel/

[4] Angryk Rafał, Galant Violetta, Gordon Minor *Travel Support System - an Agent-Based Framework*, Proceedings of the International Conference on Internet Computing, 2002.

[5] Bellifemine F., Caire G., Poggi A., Rimassa G. *JADE - A White Paper*, EXP, 2003, volume 3, issue 3, pp 6-19.

[6] Bellifemine Fabio, Caire Giovanni, Trucco Tiziana, Rimassa Giovanni *JADE programmers's guide*, 2005 (http://jade.cselt.it/doc/programmersguide.pdf).

[7] Berners-Lee T., Hendler J., Lassila O. *The Semantic Web*, Scientific American, 2001, pp. 28-37.

[8] Camacho David, Aler Ricardo, Cuadrado Juan *Rule-Based Parsing for Web Data Extraction*, 2004, Intelligent Agents for Data Mining and Information Retrieval, Idea Group Publishing.

[9] Carlson Christopher N. (2003) *Information overload, retrieval strategies and Internet user empowerment.* Proceedings of The Good, the Bad and the Irrelevant, 2003, pp. 169-173.

[10] ChefMoz website: http://chefmoz.org/

[11] CyberNeko HTML parser website: http://people.apache.org/∼andyc/neko/doc/html/index.html

[12] Daconta Michael C., ObrstLeo J., Smith Kevin T. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, 2003, John Wiley & Sons.

[13] DARPA Agent Markup Language website: http://www.daml.org/

[14] Franklin Stan, Graesser Art *Is it an Agent, or just a Program - A Taxonomy for Autonomous Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[15] Galant Violetta, Paprzycki Marcin *Information Personalization in an Internet Based Travel Support System*, Proceedings of the BIS'2002 Conference, Poznań University of Economics Press, 2002, pp. 191-202.

[16] Gawinecki Maciej, Gordon Minor, Nguyen Ngoc Thanh, Paprzycki Marcin, Szymczak Michał *RDF Demarcated Resources in an Agent Based Travel Support System*, 2005, Proceedings of the 17 th Mountain Conference of the Polish Information Processing Society.

[17] Gawinecki Maciej, Gordon Minor, Paprzycki Marcin, Szymczak Michał, Vetulani Zygmunt, Wright Jimmy *Enabling Semantic Referencing of Selected Travel Related Resources*, 2005, Proceedings of the 8 th International Conference on Business Information Systems (BIS 2005), Poznań University of Economics Press, pp. 271-288.

[18] Gawinecki Maciej, Kruszyk Mateusz, Paprzycki Marcin *Ontology-based Stereotyping in a Travel Support System*, 2005, Proceedings of the XXI Fall Meeting of Polish Information Processing Society, PTI Press, pp. 73-85

[19] Gordon Minor, Kowalski Aleksander, Paprzycki Marcin, Pełech Tomasz, Szymczak Michał, Wasowicz Tomasz *Ontologies in a Travel Support System*, 2005, In: D. J. Bem et. al. (eds.) Internet 2005, Technical University of Wroclaw Press, pp. 285-300.

[20] Gordon Minor, Paprzycki Marcin *Designing Agent Based Travel Support System*, Proceedings of the ISPDC 2005 Conference, IEEE Computer Society Press, 2005, pp. 207-214.

[21] Griss Martin L. *My Agent Will Call Your Agent... But Will It Respond?*, http://martin.griss.com/pubs/tr-hpl-1999-159.pdf

[22] Gruber Thomas R. *A Translation Approach to Portable Ontology Specifications*, Knowledge Acquisition, 1993, volume 5, issue 2, pp. 199-220.

[23] Harrington Patrick, Gordon Minor, Nauli Andy, Paprzycki Marcin, Williams Steve, Wright Jimmy *Using Software Agents to Index Data in E-Travel System*, 2003, Electronic Proceedings of the 7 th SCI Conference, Orlando

[24] Hayes-Roth B.,*An Architecture for Adaptive Intelligent Systems*, Artificial Intelligence: Special Issue on Agents and Interactivity, 1995, volume 72, pp. 329-365.

[25] Helin Heikki *Bibliography on Software Agents*, http://www.cs.helsinki.fi/u/hhelin/agents/agent-bib.html

[26] Hendler James *Agents and the Semantic Web*, 2001, IEEE Intelligent Systems, volume 16, issue 2, pp. 30 - 37.

[27] JADE website: http://jade.cselt.it/

[28] Jena website: http://jena.sourceforge.net/

[29] Koivunen Marja-Riitta, Miller Eric *W3C Semantic Web Activity*, Proceedings of the Semantic Web Kick-off Seminar in Finland, listopad 2001.

[30] Lerman Kristina, Getoor Lise, Minton Steven, Knoblock Craig *Using the Structure of Web Sites for Automatic Segmentation of Tables*, 2004, Proceedings of ACM SIG on Management of Data (SIGMOD-2004)

[31] Maes Pattie *Artificial Life Meets Entertainment: Life like Autonomous Agents*, Communications of the ACM, 1995, volume 38, issue 11, pp. 108-114.

[32] Manola Frank, Miller Eric *RDF Primer, W3C Recommendation*, 2004.

[33] McGuinness Deborah L., van Harmelen Frank *OWL Web Ontology Language Overview*, W3C Recommendation, 2004.

[34] Ben-Ameur H., Bédard F., Vaucher S., Kropf P., Chaib-draa B., Gérin-Lajoie R., *NADIM-Travel: A Multiagent Platform for Travel Services Aggregation*, 2004, 11th International Conference on Information Technology Tourism (ENTER 2004), Springer Verlag.

[35] Nwana Hyacinth S., Ndumu Divine T. *A Perspective on Software Agents Research* The Knowledge Engineering Review, 1999, volume 14, issue 2, pp. 1-18.

[36] OpenTravel Alliance webiste: http://www.opentravel.org/

[37] Palmer Sean B. *The Semantic Web: An Introduction*, 2001.

[38] Paprzycki Marcin *Agenci programowi jako metodologia tworzenia oprogramowania* Problemy i Metody Inżynierii Oprogramowania, WNT, 2003, http://www.e-informatyka.pl/article/show/422

[39] Paprzycki Marcin, Gilbert Austin, Nauli Andy, Gordon Minor, Williams Steve, Wright Jimmy *Indexing agent gathered data in an e-travel system*, 2004, Informatica, volume 28, issue 1, pp. 69-78.

[40] Paprzycki Marcin, Kaczmarek Paweł, Gawinecki Maciej, Vetulani Zygmunt *Interakcja Użytkownik - Agentowy System Wspomagania Podróży*, 2005, Proceedings of the 17th Mountain Conference of the Polish Information Society.

[41] Powers Shelley *Practical RDF*, 2003, O'Reilly.

[42] Praca zbiorowa, *IBM Intelligent Agent Strategy*, IBM Corporation, 1995.

[43] Praca zbiorowa *Towards The Semantic Web - Ontology-driven Knowledge Management*, 2003, John Wiley & Sons;

[44] Przewodnik internetowy restauracje.com: http://www.restauracje.com/

[45] Russell Stuart, Norvig Peter *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2002.

[46] Seaborne Andy *RDQL - A Query Language for RDF*, 2004.

[47] Semantic Web Community website: http://www.semanticweb.org/

[48] SmithD. C., CypherA., Spohrer J. *KidSim: Programming Agents Without a Programming Language*, Communications of the ACM, 1994, volume 37 issue 7, pp. 55-67.

[49] Specyfikacje FIPA: http://www.fipa.org/repository/standardspecs.html

[50] Swartz Aaron *The Semantic Web In Breadth*, http://logicerror.com/semanticWeb-long

[51] Standard Widget Toolkit website: http://www.eclipse.org/swt/

[52] Toffler Alvin *Future Shock*, Random House, 1970.

[53] Virdhagriswarana Sankar *MuBot Agent*, http://www.crystaliz.com/logicware/mubot.html

[54] Wooldridge M., Jennings Nicholas R. *Agent Theories, Architectures, and Languages: a Survey* Intelligent Agents, Springer-Verlag, 1995, pp. 1-22.