# Internetowy System Wspomagania Podróży – Tworzenie Podsystemu Odpowiedzialnego za Gromadzenie Danych

**Szymon PISAREK, Marcin PAPRZYCKI**

**Streszczenie:** Niniejszy artykuł opisuje, odpowiedzialny za gromadzenie danych, fragment internetowego systemu wspomagania podróży. System ten oparty jest na technologii agentowej, a jego częścią centralną jest zbiornica danych semantycznie „opisanych" w języku RDF. W tekście przedstawiamy ogólny schemat systemu: podsystemy odpowiedzialne za interakcję z użytkownikiem, zarządzanie danymi i gromadzenie danych. Następnie omówieni zostają agenci gromadzących dane ze źródeł internetowych i przekształcający je do formy wymaganej w systemie.

## 1. Wprowadzenie

Ciągły przyrost ilości informacji dostępnej w Internecie ma zarówno pozytywne jak i negatywne skutki. Z jednej strony znajdują się tam „prawie wszystkie" istotne informacje z prawie każdej dziedziny, z drugiej zaś strony ich ilość jest tak wielka, iż bardzo często mamy problem ze znalezieniem tych danych, które są nam „naprawdę" potrzebne. Wielu naukowców, w tym P. Maes [21], twierdzi, że inteligentni agenci programowi mogą być odpowiedzią na problem zalewu informacją. Jeśli rzeczywiście ma się tak stać to koniecznym jest przekształcenie informacji dostępnej w Internecie w postać, która będzie czytelna dla tychże agentów. Jedną z możliwych dróg osiągnięcia tego celu jest Semantic Web [28], czyli Internet Semantyczny, w którym informacja jest opisana ontologicznie. W tym celu możemy posłużyć się językami opisu zasobów, takimi jak: Resource Description Framework (RDF) [27] czy Ontology Web Language (OWL) [25]. Niestety, w chwili obecnej praktycznie nie ma dostępnych danych w formacie RDF/OWL. Jedno z niewielu większych repozytoriów to ChefMoz, jednakże korzystanie z tych danych bywa bardzo kłopotliwe [10].

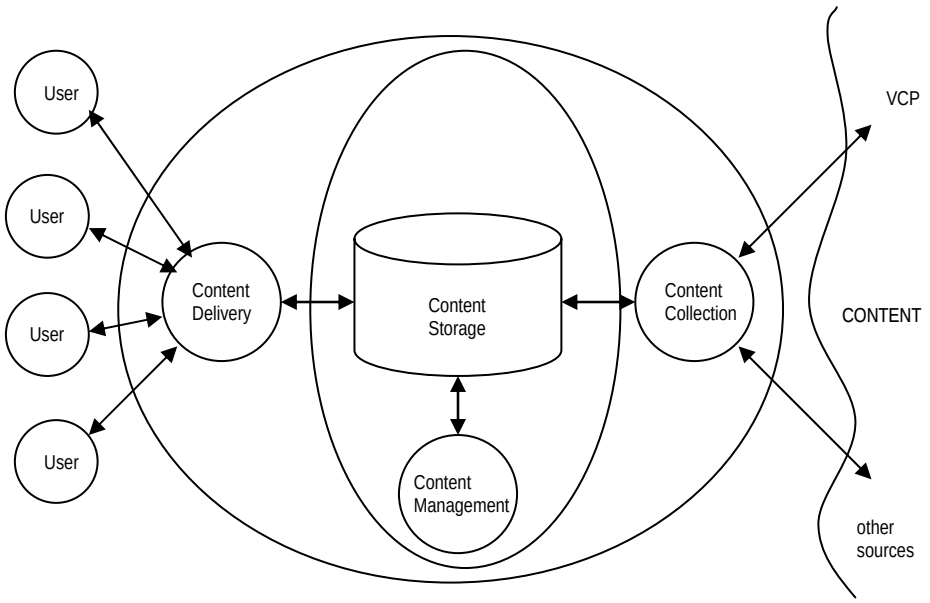
Aby przetestować hipotezę P. Maes, o skuteczności agentów programowych postanowiliśmy stworzyć agentowy system wspomagania podróży, w którym dane opisane będą językiem RDF. Celem niniejszej pracy jest omówienie rozwiązań zastosowanych przy gromadzeniu danych dla systemu. W następnej sekcji przedstawiamy całościowy obraz systemu. Następnie, w Sekcji 3, omówimy podsystem gromadzący dane, a jego działanie zilustrujemy w Sekcji 4.

## 2. Struktura Agentowego Systemu Wspomagania Podróży

Zarys architektury systemu przedstawiony został na Rysunku 1 (jego szczegółowy opis działania znajduje się w [11]). Pozostaliśmy przy nazwach angielskich, aby utrzymać zgodność z nazwami występującymi w bibliografii. System składa się z:

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

*Content Delivery Subsystem* (*CDS*) jest odpowiedzialny za odpowiedzi na zapytania przesłane przez *User*-ów. Tutaj inteligentni agenci, na podstawie zapytania użytkownika, tworzą zbiór personalizowanych danych wynikowych.




Rysunek 1. Schemat Systemu

*Content Management Subsystem* (*CMS*) odpowiada za całość funkcji związanych z zarządzaniem danymi zgromadzonymi w centralnym repozytorium. Agenci tego podsystemu odpowiedzialni są m.in. za nadzór, aby dane w systemie były aktualne. Załóżmy, iż co piątek zmienia się repertuar kina. *CMS* informuje *CCS*, iż odpowiednie dane powinny zostać zaktualizowane. Innym zadaniem *CMS* jest uzupełnianie niekompletnych danych. Załóżmy, że posiadamy dane na temat hotelu: jego nazwę i adres, ale nie mamy numeru telefonu. W tej sytuacji *CMS* informuje *CCS,* że należy znaleźć brakujące informacje.

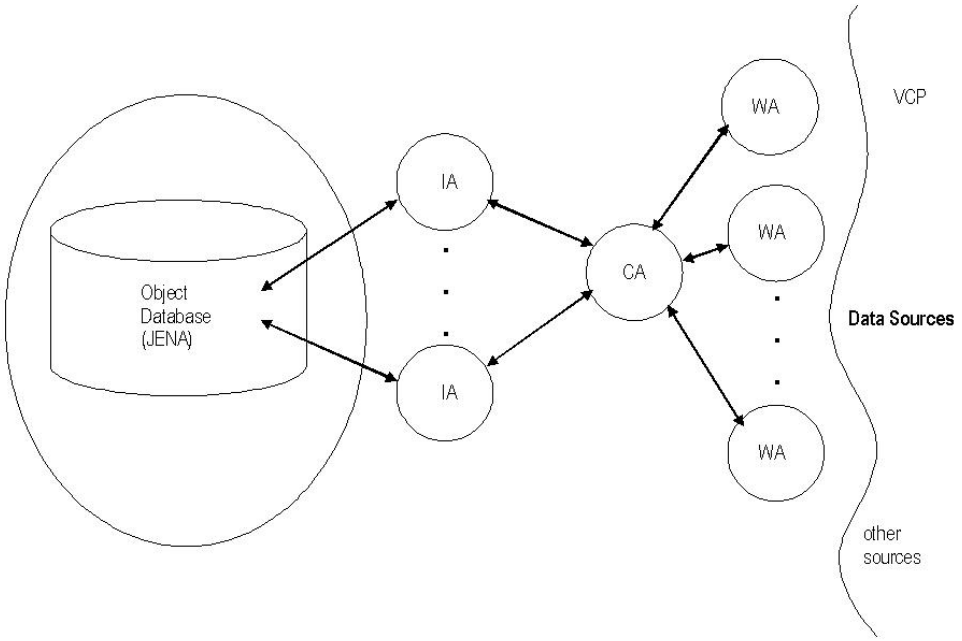
*Content Collection Subsystem* (*CCS*) stanowi przedmiot niniejszej pracy i odpowiedzialny jest za gromadzenie danych (pochodzących z różnych źródeł) i przekształcanie ich do postaci wymaganej przez system (opisanej językiem RDF i odpowiadających zakładanej przez system ontologii podróży [10]).

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

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

## 3. Podsystem Gromadzenia Informacji – *Content Collection Subsystem* (*CCS*)

Architektura podsystemu *CCS* przedstawiona jest na Rysunku 2. Przyjrzyjmy się teraz dokładnie poszczególnym jego elementom.




Rysunek 2. Podsystem Gromadzenia Informacji (*CCS*)

***Źródła danych*** Bazy danych, Web Services, strony Internetowe, pliki mogą stać się źródłami danych. Jedno źródło może składać się np. ze stron WWW oraz Web Service – takie kombinacje będą częste, ponieważ trudno znaleźć w jednym miejscu wszystkie interesujące nas dane. Obecnie korzystamy tylko ze stron internetowych, jednakże architektura systemu pozwala korzystać z innych źródeł. Wystarczy stworzyć odpowiednie „opakowanie" – *Wrapper Agent* – albowiem dla *CCS* nie ma znaczenia czy pobieramy dane ze strony WWW czy z bazy danych.

***Wrapper Agents*** (*WA*) W chwili obecnej praktycznie nie ma w Internecie danych opisanych semantycznie np. w języku RDF. Z tego właśnie powodu jesteśmy zmuszeni stworzyć agentów typu *WA*. Agent *WA* może być rozumiany jako „opakowanie" źródła danych. Jest on odpowiedzialny za wyłuskanie interesujących nas danych, stworzenie na ich podstawie tokenu kodującego dane jako trójki w języku RDF. Ponadto token zawiera dane administracyjne: informację na temat agenta, który go stworzył, datę stworzenia oraz priorytet (niski, średni lub wysoki). Następnie agent *WA* przesyła stworzony token do koordynatora (*Coordinator Agent*), komunikacja między nimi odbywa się w języku ACL [1].

Warto zauważyć, iż w bieżącej wersji *CCS* dane pobierane są ze stron internetowych i tworzenie odpowiednich agentów jest bardzo czasochłonne, a równocześnie nawet drobna zmiana w prezentowaniu (formacie) danych może

spowodować, iż agent przestanie pracować prawidłowo – parsowanie strony zakończy się błędami. Mamy nadzieję, że rozwiązanie to jest tylko tymczasowe i że w przyszłości idea Semantic Web stanie rozpowszechni się na tyle, że dane prezentowane na stronach WWW będą również dostępne w formacie RDF lub w innej reprezentacji ontologicznej.

***Coordinator Agent*** (*CA*) uruchamiany jest przy starcie systemu i istnieje dopóty, dopóki system działa. Poza koordynacją działań *Wrapper-ów* – uruchamiania ich, służy on jako tunel komunikacyjny między *Wrapper-ami* a agentami odpowiedzialnymi za zapisywanie znalezionych informacji w repozytorium (*Index Agent*). Działanie koordynatora zorganizowane jest w stylu agentowym poprzez wdrożenie odpowiednich zachowań (*Behaviour*). Z jednej strony mamy zachowanie odpowiedzialne za komunikację z *Wrapper-ami* – to tutaj odbierany jest token od *WA* i zapisany w wewnętrznej kolejce priorytetowej koordynatora. Z drugiej zaś strony, kolejne zachowanie pobiera z kolejki token o najwyższym priorytecie i przekazuje go do jednego z agentów indeksujących (*IA*) (przy użyciu protokołu FIPA-ConrtactNetProtocol [9]).

Zauważmy, że kolejka priorytetowa pozwala nam w trybie natychmiastowym obsługiwać tokeny z wysokim priorytetem. Konieczność takiego podejścia ilustruje następujący scenariusz. W pewnym momencie w systemie działa bardzo dużo *Wrapper-ów*, jedni ściągają najnowsze repertuary kin, inni pobierają najnowszy rozkład jazdy pociągów. W tym momencie jeden z użytkowników zainteresowany jest repertuarem teatrów w Warszawie. Niestety dostępne dane nie są już aktualne – to właśnie dzisiaj zmienił się repertuar. Koordynator dowiaduje się o tym od podsystemu zarządzającego danymi i uruchamia odpowiedniego *Wrapper-a*, aby ten pobrał najświeższe dane. *Wrapper* zbiera dane, tworzy odpowiedni token, przekazuje go do koordynatora. Token ten musi być opracowany w pierwszej możliwej kolejności, aby móc obsłużyć zapytanie użytkownika.

W chwili obecnej informacje na temat kiedy uruchomić dowolnego agenta *WA* czytane są przy starcie systemu (z pliku konfiguracyjnego). Jednakże w kolejnych wersjach *CCS* informacje te będą mogły zostać przekazane koordynatorowi przy pomocy odpowiedniej wiadomości w języku ACL.

W aktualnej wersji systemu koordynator służy jako „skrzynka” w procesie komunikacji między agentami *WA* a *IA*. Można by się zastanowić czy w pewnym momencie nie stanie się on „wąskim gardłem” podsystemu *CCS*. Jeśli duża liczba *Wrapper-ów* będzie pracowała w tym samym czasie, może to być rzeczywiście problem. W tej sytuacji możliwym jest zastosowanie puli agentów *CA* w miejsce pojedynczego koordynatora. W tym przypadku *WA* kontaktowałby się z pulą agentów *CA* przy pomocy protokołu FIPA-ContractNetProtocol [9] i otrzymywałby od każdego z nich informację jak bardzo jest zajęty (np. ile tokenów o danym priorytecie ma do obsłużenia w kolejce), następnie *Wrapper* wybierałby „najmniej zajętego” *CA* i przesyłałby mu token. Warto podkreślić, iż przejście z jednego do puli koordynatorów wymaga bardzo mało zmian w systemie.

***Indexing Agents*** (*IA*) Przy starcie podsystemu *CCS* uruchamiana jest pula agentów typu *Indexing Agent*. Agenci ci istnieją przez cały czas działania systemu. Agent *IA* zaszyte ma w sobie trzy zachowania. Pierwsze odpowiedzialne jest za

otrzymanie tokenu od koordynatora (*CA*). Drugie sprawdza kompletność danych w otrzymanym tokenie, np. w informacji na temat lotniska może brakować jego kodu ICAO [14]. Ostatnie zaś odpowiedzialne jest za zapisanie danych zawartych w tokenie do repozytorium. Zauważmy, że tylko agenci *IA* wiedzą, gdzie znajduje się baza danych, jaki jest jej typ: centralna czy też rozproszona, czy jest ona relacyjna czy może obiektowa. Dzięki temu wszelkie zmiany związane z repozytorium powodują modyfikację wyłącznie agentów *IA*, reszta systemu pozostaje nienaruszona. Spójrzmy teraz na fragment przykładowych danych, zapisanych w repozytorium, opisujących Warszawskie lotnisko (ontologia służąca do opisu lotniska znajduje się w [2]).

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/airports/WAW">
<admin:createdBy>airportInfo_wrapper@syme:1099/JADE
      </admin:createdBy>
<admin:createdDate>Fri Apr 15 14:15:21 CEST
      2005</admin:createdDate>
<admin:modificationTryBy>airportInfo_wrapper@syme:1099/JADE</
      admin:modificationTryBy>
<admin:modificationTryDate>Fri Apr 22 20:19:11 CEST
      2005</admin:modificationTryDate>
<admin:modifiedBy>airportInfo_wrapper@syme:1099/JADE</admin:modified
      By>
<admin:modifiedDate>Fri Apr 29 17:39:19 CEST
      2005</admin:modifiedDate>
<admin:incompleteData>true</admin:incompleteData>
<airport:iataCode>WAW</airport:iataCode>
<airport:location>Warsaw, Poland</airport:location>
<airport:name>Rafael Nunez</airport:name>
</rdf:Description>
```

Zapisane dane administracyjne to (*zasób* = *warszawskie lotnisko*):

1. `admin:createdBy` – informacja na temat agenta, który jako pierwszy zebrał dane służące do opisu pewnego zasobu
2. `admin:createdDate` – data stworzenia trójek RDF opisujących zasób
3. `admin:modificationTryBy` – dane agenta, który próbował uaktualnić dane odnośnie pewnego zasobu, jednakże przesłane przez niego dane były takie same jak w repozytorium
4. `admin:modificationTryDate` – data próby uaktualnienia danych opisujących pewien zasób przesłane dane były identyczne jak w repozytorium
5. `admin:modifiedBy` – informacja na temat agenta, który przesłał najbardziej aktualne dane odnośnie zasobu
6. `admin:modifiedDate` – data modyfikacji trójek RDF opisujących zasób
7. `admin:incompleteData` – flaga informująca, czy dane opisujące zasób są kompletne czy nie, innymi słowy czy do opisu zasobu wykorzystujemy wszystkie właściwości ontologii wybranej do opisu pewnej klasy zasobów (w tym przypadku lotnisk)

### 3.1 Dodatkowe uwagi na temat *Wrapper-ów*

Proponowany system odpowiedzialny za gromadzenie danych jest na tyle uniwersalny, iż można go traktować jako framework do budowy systemów typu *CCS*. Jedyne elementy, które będą zmieniały się to agenci *WA*. Zauważmy tutaj, że jeśli powstaną w Internecie repozytoria danych opisanych semantycznie, to wystarczy wówczas stworzyć *WA* odczytujących tak opisane dane i tworzących odpowiednie tokeny. Konieczność tworzenia kolejnych *WA* jest słabym punktem przedstawionego rozwiązania. Zastanówmy się w jaki sposób można by zmniejszyć ilość pracy potrzebnej do tworzenia agentów tego typu.

W chwili obecnej dla każdej strony internetowej musimy stworzyć „ręcznie" agenta *WA*, który rozumie/zna prezentowany tam format informacji. Nawet niewielka zmiana formatu powoduje konieczność przebudowy agenta.

Pierwszym pomysłem jest system, który automatycznie generowałby agentów *Wrapper Agent* dla zadanej strony WWW. System taki, np. przy wykorzystaniu sieci neuronowych, byłby w stanie rozpoznawać strukturę strony, tabele, zwykły tekst, itp.. Czy jednak budowa takiego systemu nie byłaby trudniejsza od budowy naszego podsystemu *CCS*? Problem ten omawiany jest w [4, 18, 19, 20, 23].

Inne rozwiązanie koncentruje się wokół aplikacji typu *Web Crawler* (*WC*), a bardziej szczegółowo wokół rozwiązań typu *Focused Crawler* (*FC*). Zamiast agenta *WA* przydzielonego do konkretnego źródła danych mamy tutaj agentów typu *Focused Crawler Agent* (*FCA*) odwiedzających zasoby, „czytających" informacje, próbujących je „zrozumieć" i przekształcić w trójki RDF. Największą trudnością w tym podejściu jest zbudowanie agenta, który zrozumie dane nieustruktualizowane. Więcej informacji o *Web Crawler* i *Focused Crawler* można znaleźć w [5, 6, 22, 24, 26].

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

## 4. Przykład działania podsystemu *CSS*

Przedstawimy teraz działanie jednego z zaimplementowanych *WA*. Jest to *MarriottHotelsWrapperAgent (MHWA)*. Agent ten zbiera informacje na temat wszystkich hoteli Marriott na świecie. Jako źródło danych wykorzystujemy strony z portalu firmy Marriott (http://marriott.com). Dane jakich potrzebujemy opisane są poprzez ontologię *Hotel Ontology* zaproponowaną w [10].

Portal firmy Marriott umożliwia przeszukiwanie hoteli według państw oraz według stanów dla USA (Rysunek 3.) Pierwszym zadaniem agenta *MHWA* jest znalezienie wszystkich państw i stanów (dla USA) gdzie znajdują się hotele Marriott. Strona WWW znajdująca się na Rysunku 3 jest pierwszą stroną odwiedzaną przez *Wrapper-a*. Poczynione zostało tutaj założenie, że w Polsce (query string: `?country=PL`) istniej co najmniej jeden hotel Marriott. Po prawej stronie znajduje się lista rozwijana z państwami/stanami (dla USA) w których występują hotele (na rysunku lista ze stanami jest zasłonięta przez rozwiniętą listę państw). *MHWA* „wyłuskuje" wszystkie kody państw oraz stanów USA z

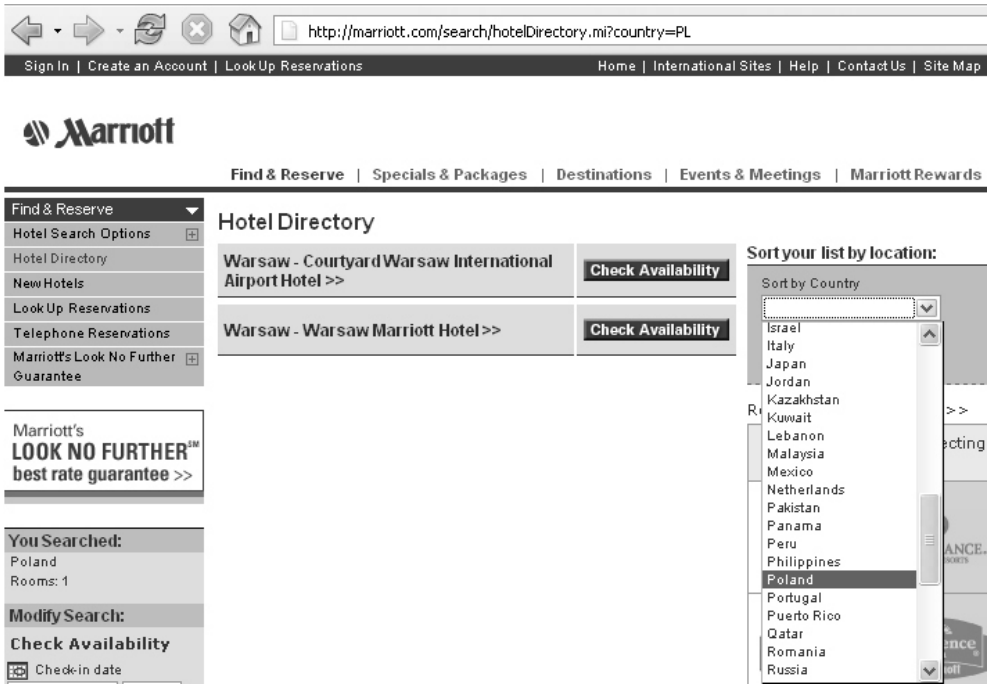
rozwijanych list i dla każdego kodu odwiedza stronę zmieniając odpowiednio „query string”. Dla kodów państw „query string” jest następujący:

`?country=[kod państwa]`, np. dla Japonii jest to: `?country=JP`,

dla stanów USA:

`?state=[kod stanu]`, np. dla stanu Kalifornia jest to: `?state=CA`.

Z odwiedzonych stron agent odczytuje kody wszystkich hoteli występujących w danym państwie/stanie (środkowa część strony na Rysunku 3.).




Rysunek 3. Hotele Marriott (strona główna)

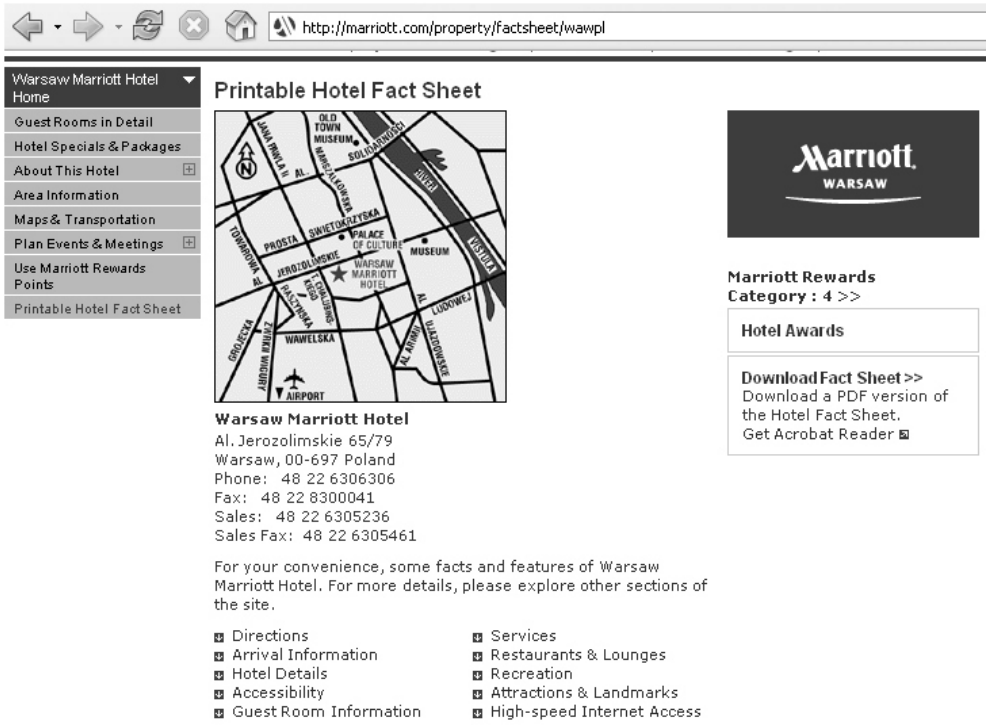
Mając kody hoteli, można pobrać szczegóły. Adres strony z detalami hoteli ma następujący format:

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

Dla każdego kodu hotelu, *MHWA* odwiedza stronę z detalami podmieniając w powyżej podanym adresie URL posiadany już kod hotelu. Przykładowa strona, dla warszawskiego hotelu Marriott, znajduje się na Rysunku 4. Po załadowaniu strony z detalami, agent wyłuskuje interesujące go dane (te wymagane przez ontologię hotelu zaproponowaną w [10]) i tworzy trójki RDF opisujące ten hotel. Przyjrzyjmy się fragmentowi kodu w HTML reprezentującego adres hotelu:

```
<div id="hotelAddressSub">
<h2>Warsaw Marriott Hotel</h2>
       <ul>
              <li>Al. Jerozolimskie 65/79</li>
              <li>Warsaw, 00-697 Poland</li>
```

```
            <li>Phone:   48 22 6306306</li>
            <li>Fax:    48 22 8300041</li>
            <li>Sales:    48 22 6305236</li>
            <li>Sales Fax:  48 22 6305461</li>
      </ul>
</div>
```




Rysunek 4. Szczegóły hotelu Marriott w Warszawie (strona WWW)

Z powyższego kody HTML agent *MHWA* tworzy następujące trójki RDF:

```
<rdf:Description
      rdf:about="http://www.agentlab.net/travel/hotels/Marriott/WAW
            PL">
      <location:streetAddress>Al. Jerozolimskie
            65/79</location:streetAddress>
      <location:city>Warsaw</location:city>
      <location:zip>00-697</location:zip>
      <location:country>Poland</location:country>
      <location:phone>48 22 6306306</location:phone>
      <location:fax>48 22 8300041</location:fax>
</rdf:Description>
```

Po zbudowaniu (kompletnego lub niekompletnego) opisu hotelu, agent tworzy token, zawierający trójki RDF oraz dane administracyjne, i wysyła go do *CA*. *CA* będzie postępował z otrzymanym tokenem w sposób opisany w Sekcji 3.1.

## 5. Uwagi końcowe

W niniejszej pracy przedstawiliśmy ogólną architekturę Agentowego Systemu Wspomagania Podróży, szczególną uwagę przywiązując do podsystemu gromadzenia informacji (*CSS*). Omówiliśmy agentów *CCS*: *Coordinator Agent*, *Wrapper Agent*, *Indexing Agent*. Rozważyliśmy również możliwości ulepszenia podsystemu poprzez automatyczne generowanie *Wrapper-ów* oraz stworzenie nowego typu agentów dostarczających dane do systemu: *Focused Crawler Agent*. Następnie przedstawiliśmy przykład działania *Wrapper-a* zbierającego dane na temat hoteli Marriott.

W chwili obecnej zaimplementowane zostały dwa *Wrapper-y*. *AirportsWrapperAgent* – odpowiedzialny za zbieranie informacji na temat lotnisk na świecie i korzystający z portalu: http://www.world-airport-codes.com (kody lotnisk IATA [13]) oraz strony http://www.airrouting.com/scripts/airportloc.asp – dla podanego kodu IATA otrzymuje dane na temat lotniska wykorzystywane w ontologii *Airport* [2]. Drugim agentem jest zaprezentowany powyżej *MarriottHotelsWrapperAgent* –odpowiedzialny za zbieranie informacji na temat wszystkich hoteli Marriott na świecie.

Podsystem *CCS* został zbudowany przy wykorzystaniu JADE [15] – otoczenia do tworzenia aplikacji agentowych. Podczas „wyłuskiwania" danych ze stron WWW wykorzystywany jest CyberNeko HTML Parser [7] – parser HTML, który dla zadanej strony tworzy reprezentujący ją obiekt DOM [8] oraz Jaxen [16] – XPath engine [30]. Strony ładowane są do pamięci przy użyciu biblioteki HTTP Client [12].

Obecnie jesteśmy w trakcie budowania kolejnych *Wrapper-ów* i integrowania ich z systemem. O postępach będziemy informować w kolejnych raportach.

## Literatura


1. Agent Communication Language: http://www.fipa.org/repository/aclspecs.html
2. Airport Ontology: http://www.daml.org/ontologies/323, http://www.daml.org/2001/10/html/airport-ont
3. Angryk Rafał, Galant Violetta, Gordon Minor, Paprzycki Marcin (2002) Travel Support System - an Agent-Based Framework. In: H. R. Arabnia, Y. Mun (ed.), *Proceedings of the International Conference on Internet Computing* (IC'02), CSREA Press, Las Vegas, NV, 719-725
4. Ashish Naveen, Knoblock Craig; Semi-automaticWrapper Generation for Internet Information Sources; In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*, Kiawah Island, SC, 1997
5. Bradshaw Shannon, Menczer Filippo, Pant Gautam; Search Engine-Crawler Symbiosis: Adapting to Community Interests; *Proc. 7th European Conference on Research and Advance d Technology for Digital Libraries* (ECDL 2003)
6. Chau Michael, Chen Hsinchun; Personalized and Focused Web Spiders; in *N. Zhong, J. Liu, Y. Yao (Eds), Web Intelligence*, Springer-Verlag, February 2003, pp. 197-217
7. CyberNeko HTML Parser: http://people.apache.org/~andyc/neko/doc/html/index.html
8. Document Object Model: http://www.w3.org/DOM/
9. FIPA Contract Net Interaction Protocol: http://www.fipa.org/specs/fipa00029/XC00029F.html

10. Gawinecki M., Gordon M., Paprzycki M., Szymczak M., Vetulani Z., Wright J.; (2005) Enabling Semantic Referencing of Selected Travel Related Resources. w: W. Abramowicz (ed.) *Proceedings of the 8th International Conference on Business Information Systems* (BIS 2005), Poznań University of Economics Press, Poznań, Poland, 271-288
11. Gordon M., Paprzycki M.; Designing Agent Based Travel Support System, w przygotowaniu
12. HTTP Client: http://jakarta.apache.org/commons/httpclient/
13. IATA: http://www.iata.org
14. ICAO: http://www.icao.int
15. JADE: http://jade.tilab.com/
16. Jaxen: http://jaxen.org/
17. Jena: http://www.hpl.hp.com/semweb/jena.htm
18. Knoblock Craig, Lerman Kristina, Minton Steven, Muslea Ion; Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach; *IEEE Data Engineering Bulletin*, 23(4):33--41, December 2000
19. Lerman Kristina, Gazen Cenk, Minton Steven, Knoblock Craig; Populating the SemanticWeb; In *Proceedings of the AAAI 2004 Workshop on Advances in Text Extraction and Mining*, 2004
20. Lerman Kristina, Getoor Lise, Minton Steven, Knoblock Craig; Using the Structure of Web Sites for Automatic Segmentation of Tables; In *Proceedings of ACM SIG on Management of Data (SIGMOD-2004)*, 2004
21. Maes P.; Agents that Reduce Work and Information Overload; *Communications of the ACM*, 37, 7, (1994) 31-40
22. Marc Ehrig; Ontology-Focused Crawling of Documents and Relational Metadata; Master's Thesis
23. Muslea Ion, Minton Steven, Knoblock Craig; Hierarchical Wrapper Induction for Semistructured Information Sources; *Autonomous Agents and Multi-Agent Systems*, 4(1/2), March 2001
24. Novak Blaž; A Survey of Focused Web Crawling Algorithms; *Conference on Data Mining and Warehouses* (SiKDD 2004)
25. Ontology Web Language: http://www.w3.org/TR/owl-features/
26. Pant Gautam, Srinivasan Padmini, Menczer Filippo; Crawling the Web; in *M. Levene and A. Poulovassilis, editors: Web Dynamics*, Springer-Verlag, 2003
27. Resource Description Framework: http://www.w3.org/RDF/
28. Semantic Web: http://www.w3.org/2001/sw/
29. Travel Support Project: http://www.agentlab.net/projects/e-Travel/
30. XPath: http://www.w3.org/TR/xpath

Szymon Pisarek
Wydział Matematyki i Nauk Informacyjnych, Politechnika Warszawska
Pl. Politechniki 1, 00-661 Warszawa
*e-mail:* szymekpi@o2.pl

Marcin Paprzycki
Instytut Informatyki, SWPS
ul. Chodakowska 18/31, 03-815 Warszawa
*e-mail:* Marcin.Paprzycki@swps.edu.pl