Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

Praca magisterska

Mapowanie ontologii na przykładzie CYC i Słownika Semantycznego Języka Polskiego

Aleksander Pohl

Kierunek: Informatyka

Nr albumu: 112020

Promotor
prof. dr hab. Wiesław Lubaszewski

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Kraków 2006

Oświadczenie autora

Ja, niżej podpisany Aleksander Pohl oświadczam, że praca ta została napisana samodzielnie i wykorzystywała (poza zdobytą na studiach wiedzą) jedynie wyniki prac zamieszczonych w spisie literatury.

. . . . . .
(Podpis autora)

Oświadczenie promotora

Oświadczam, że praca spełnia wymogi stawiane pracom magisterskim.

. . . . . .
(Podpis promotora)

Spis treści

Część I

Wprowadzenie

Rozdział 1

Cel pracy

1.1. Wstęp

Celem niniejszej pracy jest rozwiązanie problemu mapowania Słownika Semantycznego Języka Polskiego[1] na ontologię Cyc[2]. Każda z wymienionych struktur abstrakcyjnych odzwierciedla, na swój sposób, relacje jakie zachodzą z jednej strony – pomiędzy słowami wewnątrz jakiegoś języka naturalnego (w przypadku Słownika Semantycznego – języka polskiego, w przypadku ontologii Cyc – języka angielskiego), z drugiej zaś – pomiędzy słowami, a rzeczywistością pozajęzykową.

W niniejszym rozdziale przedstawimy jedynie przybliżone definicje *ontologii* oraz *słownika semantycznego* aby wstępnie sformułować cel niniejszej pracy. Pełniejsza ekspozycja tych pojęć nastąpi w rozdziałach późniejszych.

Ontologia jest specyfikacją konceptualizacji wybranej dziedziny rzeczywistości, natomiast *słownik semantyczny* to słownik języka naturalnego, w którym definicje poszczególnych słów wyrażone są w postaci relacji semantycznych w jakie wchodzą definiowane słowa z innymi słowami.

Zarówno koncepty występujące w ontologiach, jak i słowa definiowane w słownikach semantycznych reprezentują pewne obiekty z rzeczywistości pozajęzykowej. Zatem mapowanie jednej struktury na drugą polega na rozpoznaniu w drugiej strukturze elementów, które reprezentują te same (przynajmniej pod pewnym względem) obiekty, co obiekty reprezentowane przez poszczególne elementy struktury pierwszej oraz zachowaniu tej informacji w sposób dogodny dla dalszego wykorzystania.

Rozpoznanie odpowiadających sobie elementów poszczególnych struktur może być dokonane przez człowieka (mamy wtedy do czynienia z tzw. *ręcznym* mapowaniem) bądź przez komputer wyposażony w odpowiedni program (jest to mapowanie *automatyczne*). W prak-

[1] http://wierzba.wzks.uj.edu.pl/smd
[2] http://www.cyc.com

tyce stosuje się również podejście mieszane, w którym komputer dostarcza ograniczony zestaw propozycji mapowania, z których człowiek wybiera poprawne (jest to mapowanie *półautomatyczne*).

1.2. Przyczyny mapowania ontologii

Ontologie oraz słowniki semantyczne, o których mowa w poprzednim punkcie, znajdują coraz więcej zastosowań, m.in. w problemach z zakresu przetwarzania języka naturlanego (NLP). W idealnym świecie mielibyśmy do czynienia z jedną ontologią lub jednym słownikiem semantycznym, które w sposób doskonały opisywałyby rzeczywistość. Rzeczywisty świat wygląda jednak zupełnie inaczej – powstaje coraz więcej niekompatybilnych ontologii oraz słowników, które nie mogą być w łatwy sposób dostosowane do siebie nawzajem.

Oprogramowanie przetwarzające wiedzę, które bazuje na jednej ontologii, nie może skorzystać bezpośrednio z wiedzy zgromadzonej w innym systemie bazującym na innej ontologii. Ponieważ jednak coraz częściej zachodzi konieczność wymiany wiedzy pomiędzy takimi heterogenicznymi systemami (np. w środowiskach agentowych), pożądane stało się stworzenie algorytmów, które pozwoliłyby na mapowanie niezależnie powstałych ontologii, dzięki któremu możliwe byłoby szersze wykorzystanie zgromadzonej wiedzy [6].

Z kolei w systemach przetwarzania języka naturalnego często przydatne jest wykorzystanie wiedzy zgromadzonej w ontologiach, np. w algorytmach tłumaczenia maszynowego opartych o wiedzę (knowledge based machine translation – KBMT)[23]. Dodatkowo, ontologie często dysponują wydajnymi silnikami wnioskującymi, które mogłyby również zostać wykorzystywane przy przetwarzaniu języka naturalnego. Dlatego właśnie zachodzi potrzeba znalezienia mapowania pomiędzy poszczególnymi elementami ontologii a słowami występującymi w słownikach semantycznych.

1.3. Efekty osiągnięte dzięki mapowaniu

Prace nad niniejszym projektem motywowane były w ogólności chęcią stworzenia bazy wiedzy przydatnej przy przetwarzaniu tekstów w języku polskim. Poszczególne efekty, które dzięki realizacji tego projektu chcemy osiągnąć, są przedstawione poniżej.

1.3.1. Uzupełnienie struktury Słownika

Słownik Semantyczny Języka Polskiego jest przedsięwzięciem w początkowej fazie swojego rozwoju. Ontologia Cyc powstawała przez przeszło 20 lat. Dzięki porównaniu struktury oraz zawartości tych źródeł wiedzy, można będzie wyciągnąć wnioski dotyczące kierunku dalszego rozwoju słownika oraz niezbędnych uzupełnień, które muszą w nim zostać dokonane.

1.3.2. Polski leksykon w Cyc

W przypadku pełnego zmapowania Słownika na ontologię Cyc uzyskalibyśmy polski leksykon dla tej ontologii, dzięki czemu można by rozwijać bazujące na niej algorytmy przetwarzania języka naturalnego. W tym wypadku szczególnie cenny jest silnik wnioskujący, w który wyposażona jest ontologia Cyc.

1.3.3. Stworzenie narzędzi do przeglądania słownika/ontologii

Przy okazji mapowania słownika konieczne jest stworzenie narzędzi, które pozwalałyby przeglądać Słownik Semantyczny oraz ontologię Cyc. Narzędzia takie ze względu na to, że integrują dwa niezależne źródła wiedzy, mogą okazać się na tyle elastyczne, że będą mogły być wykorzystane również w innych projektach opartych o (być może inne) słowniki i ontologie.

1.3.4. Stworzenie algorytmów parsingu haseł słownikowych

Zarówno Słownik Semantyczny Języka Polskiego jak i ontologia Cyc dostarczają odpowiedni interfejs programistyczny (API) pozwalający manipulować występującymi w nich danymi. Tym niemniej, aby mapowanie mogło być prowadzone w sposób przynajmniej półautomatyczny, konieczne jest wykorzystanie słownika polsko-angielskiego. W obecnej chwili na rynku polskim nie znaleziono żadnego słownika, który dostarczałby odpowiedni interfejs programistyczny. W związku z tym konieczne jest wykorzystanie jednego ze słowników elektronicznych, który nie był tworzony z myślą o wykorzystaniu przez inne programy komputerowe.

Z tego względu konieczne będzie opracowanie algorytmów parsingu haseł słownikowych oraz zaproponowanie struktur danych odpowiednich dla tych haseł. Rozwiązania te mogą w przyszłości zostać wykorzystane w innych projektach np. z zakresu automatycznego tłumaczenia tekstów.

1.4. Stan badań

Problem mapowania ontologii jest obecnie bardzo intensywnie badany. Powstaje szereg prac, w których proponowane są różnego rodzaju algorytmy pozwalające dopasować do siebie niezależnie powstałe ontologie. Natomiast zagadnienie tworzenia leksykonów dla wybranej ontologii w różnych językach naturlanych, nie zostało dotychczas wystarczająco zgłębione. Powodem tego może być fakt, że w systemach ontologicznych różnica pomiędzy warstwą konceptualną a językową często nie jest wystarczająco uwypuklona [4].

Poniżej przedstawiamy kilka prac z zakresu mapowania ontologii, które uznane zostały za szczególnie wartościowe. Nie wszystkie algorytmy w nich zaprezentowane mogłyby zostać zastosowane w niniejszym projekcie, aczkolwiek stanowią dobrą ilustrację kontekstu, w którym jest on realizowany.

1.4.1. GLUE

W pracy [6] zaproponowano algorytm automatycznego mapowania ontologii, bazujący na uczeniu maszynowym. Może on jednak działać tylko w ontologiach posiadających dla każdego mapowanego konceptu wiele przykładów użycia. Na ich podstawie konstruowane są klasyfikatory, które wykorzystuje się do zbadania podobieństwa poszczególnych konceptów. Klasyfikatory te konstruowane są na podstawie tekstowej zawartości przykładów użycia konceptów.

Metoda ta nie może być jednak zastosowana w naszym projekcie z dwóch powodów. Po pierwsze – zarówno ontologia Cyc jak i Słownik Semantyczny Języka Polskiego nie posiadają danych, na podstawie których można by stworzyć odpowiednie klasyfikatory (w przypadku Cyc byłyby to indywidua należące do odpowiednich kolekcji, w przypadku

Słownike Semantycznego – konteksty językowe, w których wybrane słowo wystąpiło w danym znaczeniu). Po drugie zaś, nawet gdybyśmy dysponowali odpowiednimi danymi, to ze względu na to, że wyrażone byłyby one w innych językach naturalnych, konstrukcja odpowiednich klasyfikatorów byłaby pozbawiona sensu.

1.4.2. OntoMorph

System OntoMorph [5] został zaprojektowany do przekształcania, czyli tłumaczenia ontologii. Nie jest to system, który w ścisłym sensie służy do ich mapowania. Aczkolwiek w pracy jemu poświęconej można natknąć się na problemy podobne do problemów z zakresu mapowania ontologii. System składa się z dwóch części – jednej odpowiedzialnej za przekształcanie składni języków opisu ontologii, drugiej odpowiedzialnej za przekształcanie ich semantyki.

Każda z tych części posługuje się pewnym językiem przekształceń. W przypadku pierwszej jest on podobny do XSLT 2.0 [17], gdyż służy głównie do przekształcania drzewa rozbioru syntaktycznego poszczególnych sentencji. Język używany do przekształcania semantyki jest bardziej skomplikowany – w szczególności w poszczególnych regułach przekształceń można korzystać z silnika wnioskującego, który pozwala na wykonywanie skomplikowanych przekształceń.

Podstawowa wada zaproponowanego systemu polega na tym, że wszystkie reguły przekształceń (zarówno syntaktycznych jak i semantycznych) muszą zostać zaprojektowane „ręcznie”, zatem automatyzacja w zasadzie ogranicza się do przekształcania wybranych formalizmów na podstawie zasad wykrytych przez inżyniera ontologii.

1.4.3. CtxMatch

Algorytm *CtxMatch* [3] wykorzystuje trzy rodzaje informacji zawartej w mapowanych strukturach: leksykalną, dziedzinową oraz strukturalną. W przeciwieństwie do innych algorytmów mapowania ontologii, nie wykorzystuje on uczenia maszynowego, lecz przekształca zagadnienie mapowania w dobrze znany problem SAT (spełnialność formuły logicznej).

Algorytm ten jest szczególnie interesujący z naszego punktu widzenia, gdyż, dzięki temu że nie korzysta z informacji o przykładach użycia poszczególnych konceptów, mógłby być zastosowany w kontekście Cyc i Słownika Semantycznego Języka Polskiego. Ponadto, dzięki temu że wykorzystuje informację leksykalną, z powodzeniem mógłby zostać zastosowany dla struktur, które opisane są za pomocą różnych języków naturalnych. Algorytm ten bowiem w prosty sposób może zostać zaadoptowany tak, by korzystał z leksykonu bilingwalnego.

Część II

Podstawy teoretyczne

Rozdział 2

Model danych

2.1. Teoretyczny model języka

Ponieważ Słownik Semantyczny Języka Polskiego i ontologia Cyc odzwierciedlają, na swój sposób, strukturę języka, konieczne jest przedstawienie najbardziej ogólnego spojrzenia na język, jakie przyjęte jest we współczesnym językoznawstwie.

2.1.1. Język a komunikacja

Uważa się obecnie, że podstawową funkcją języka jest komunikowanie, czyli przekazywanie informacji. Typowy obraz tego zjawiska zawiera następujące elementy statyczne: nadawcę komunikatu, kanał komunikacyjny i odbiorcę komunikatu oraz dynamiczne: operacje kodowania i dekodowania oraz sam komunikat. Nadawca koduje informację do postaci komunikatu, wykorzystując możliwości jakie daje mu kanał komunikacyjny. Odbiorca dekoduje ją, na podstawie komunikatu, który dotarł do niego poprzez kanał.

Jako przykład można podać sytuację komunikowania pewnego stanu rzeczy, o którym nadawca sądzi, że zachodzi, za pomocą mówienia: nadawca koduje swoją wiedzę wydając dźwięki (wprawiając w drgania cząsteczki powietrza, poprzez wydmuchiwanie powietrza z płuc, modulując ich częstotliwość za pomocą strun głosowych). Poszczególne dźwięki tworzą składniki komunikatu. Odbiorca odbiera te ruchy powietrza za pomocą narządu słuchu, rekonstruuje komunikat i dekoduje zawartą w nim informację.

2.1.2. Syntaktyka, semantyka i pragmatyka

Badania nad poszczególnymi elementami biorącymi udział w procesie komunikacji prowadzone są w ramach tradycyjnie wyróżnionych dziedzin:

1. **Syntaktyka** – zajmuje się elementami, które tworzą strukturę komunikatu oraz abstrakcyjnymi relacjami pomiędzy nimi.

2. **Semantyka** – zajmuje się relacjami, jakie zachodzą pomiędzy elementami, które tworzą komunikat a informacją zawartą w tym komunikacie, innymi słowy koncentruje się ona na znaczeniu komunikatu.
3. **Pragmatyka** – zajmuje się relacjami i zjawiskami, jakie zachodzą pomiędzy procesem komunikacji a jego uczestnikami, czyli nadawcą i odbiorcą komunikatu.

Ponieważ badane struktury abstrakcyjne starają się opisać relacje jakie zachodzą pomiędzy językiem a światem, wchodzą one naturalnie w zakres zainteresowania semantyki i pragmatyki. Tym niemniej, niemożliwe jest całkowie abstrahowanie od syntaktyki, gdyż dziedziny te zachodzą na siebie w pewnym stopniu.

2.2. Teoretyczny model znaczenia

Niezwykle ważnym, wypracowanym przez semantykę, narzędziem przydatnym do opisu badanych struktur jest *trójkąt semiotyczny*. Jego źródła należy szukać u Arystotelesa, który opisywał relacje jakie zachodzą z jednej strony: pomiędzy formą słowa (w terminologii teorii komunikacji – składnikiem komunikatu) a jego znaczeniem, z drugiej zaś: pomiędzy znaczeniem słowa a tym do czego słowo to odnosi się w rzeczywistości pozajęzykowej. Pomysł ten rozwijany był przez scholastyków, później zaś znalazł dosyć ścisłe opracowanie w pismach C. S. Peirce'a [30] oraz C. K. Ogdena i I. A. Richardsa [29].

Idea trójkąta semiotycznego jest następująca – wszelkie symbole służące do komunikowani, innymi słowy, wszystko co posiada znaczenie, składa się z trzech elementów:

1. formy
2. znaczenia
3. denotacji

Forma, w przedstawionym schemacie, reprezentuje materialny nośnik symbolu, którym mogą być dźwięki mowy, litery zapisane na kartce, gesty ręką w języku migowym, etc. Innymi słowy, jest to jakieś fizyczne zjawisko, które występuje w kanale komunikacyjny i może być zaobserwowane przez odbiorcę komunikatu. Jest to składnik komunikatu, który jest podstawą jego interpretacji.

Najistotniejszą własnością formy jest to, że choć wyrażana jest ona poprzez różne tokeny (instancje), które nie są ze sobą tożsame, np. dźwięki słyszane przy czytaniu słowa *kot*, przez dwie różne osoby; to sama forma może zostać utożsamiona. Gdyby utożsamienie to nie następowało, wszelka komunikacja byłaby niemożliwa.

Należy zwrócić tutaj uwagę, że w istocie forma jest odpowiednikiem językowego wyrazu, w znaczeniu leksemu[1]. Zatem pod jedną formę symboliczną podpadają zarówno różne reprezentacje fizyczne danego symbolu, jak i różne formy fleksyjne (o ile takowe występują).

Znaczenie jest najbardziej problematycznym elementem przedstawionego schematu. Z jednej strony wiadomo, że nie ma bezpośredniej relacji pomiędzy formą symboliczną a denotacją, dlatego musi występować element pośredniczący w relacji pomiędzy nimi. Z drugiej jednak strony natura owego elementu jest trudna do przeniknięcia – jest on przedmiotem badań filozoficznych od stuleci, mimo to nie osiągnięto konsensusu co do jego statusu ontologicznego. Nie wiadomo bowiem czym właściwie jest znaczenie – wybieg utożsamiający je z pewnym stanem umysłu użytkownika języka jest nieakceptowalny dla tych, którzy poddają w wątpliwość istnienie umysłu.

[1] W językach fleksyjnych formy fleksyjne danego wyrazu (takie jak *kot, kota, kotu,...*) tworzą jedną formę symboliczną, czyli jeden leksem (zapisywany zwykle dużymi literami: KOT). [22, s. 222, 444]

Nie będziemy jednak wchodzić w to zagadnienie, przyjmując po prostu, że znaczenie jest elementem pośredniczącym w badanej relacji forma-denotacja, który wchodzi również w relacje ze znaczeniami innych symboli. W szczególności będą nas interesowały te drugie relacje, gdyż badane struktury abstrakcyjne (Słownik Semantyczny Języka Polskiego oraz Ontologia Cyc) zawierają ich odpowiedniki.

Denotacja symbolu, to coś, do czego symbol ten się odnosi. Zwykle jest to jakiś obiekt z rzeczywistości pozajęzykowej, aczkolwiek języki naturalne zawierają zwykle swój metajęzyk, tzn. język który pozwala opisać te języki. Dlatego też denotacją wyrazu „wyraz" w zdaniu „Wyraz «krótki» jest krótki" jest obiekt z rzeczywistości językowej – wyraz „krótki". Ponadto w języku można odnosić się do wielu obiektów, które w ogóle nie występują w rzeczywistości, np. w zdaniu „Hamlet był mężczyzną" denotacja wyrazu „Hamlet" jest niemożliwa (lub trudna) do identyfikacji. Można też odnosić się do obiektów, których status ontologiczny jest problematyczny – np. dobro, piękno, etc.

Problemy te, choć niezwykle ważne w sporach filozoficznych, nie będą również brane pod uwagę w badaniach tutaj prowadzonych. Niezależnie od tego, czy dany wyraz, w jakimkolwiek użyciu, posiada denotację, to jeśli można zidentyfikować relacje semantyczne pomiędzy nim, a innymi wyrazami, będzie on przedmiotem naszych badań, gdyż uznawany jest wtedy za element języka. Takie podejście prezentowane jest zarówno w Słowniku Semantycznym Języka Polskiego jak i ontologii Cyc.

2.3. Relacje semantyczne

W poprzednim punkcie uznaliśmy, że najistotniejszym czynnikiem, który decyduje o tym, czy dany symbol jest elementem języka, jest występowanie relacji semantycznych pomiędzy znaczeniem tego symbolu, a znaczeniem innych symboli. Określenie to byłoby jednak puste, gdybyśmy nie wskazali czym są relacje semantyczne. Poniżej wyjaśniamy o jakie relacje tutaj chodzi i jak można je identyfikować.

2.3.1. Pojęcie dystrybucji

Pojęcie dystrybucji zdefiniowane jest w [22, s. 84]:

> Każda jednostka językowa [...] podlega w większym lub mniejszym stopniu ograniczeniom, co do kontekstów, w jakich może występować. Fakt ten wyraża się w stwierdzeniu, że każda jednostka językowa poniżej poziomu zdania ma swą charakterystyczną *dystrybucję*[2].

W odniesieniu do słów – dystrybucja słowa to możliwość jego wystąpienie w otoczeniu innych słów, czyli w określonym kontekście słownym.

Jeśli porównujemy dystrybucje dwóch słów, to możemy uzyskać jeden z czterech przypadków ich wzajemnej dystrybucji:

1. równoważną
2. komplementarną
3. inkluzywną
4. zazębiającą się

[2] [Podkreślenie moje – A.P.]

Dystrybucja *równoważna* słów A i B zachodzi wtedy, gdy słowo A może wystąpić we wszystkich kontekstach w których występuję słowo B oraz słowo B może wystąpić we wszystkich kontekstach, w których występuje słowo A. Dobrym przykładem są tutaj synonimy, ale nie tylko one posiadają równoważną dystrybucję. Możliwość występowania dwóch słów w danym kontekście nie odnosi się bowiem do tożsamości znaczenia zdań, w których osadzone są odnośne konteksty. Warunek ten wymaga jedynie aby zdania te były sensowne.

Dystrybucja *komplementarna* występuje pomiędzy słowania A i B wtedy, gdy ani słowo A nie może wystąpić w kontekście w którym występuje słowo B, ani słowo B nie może wystąpić w kontekście, w którym występuje słowo A.

Dystrybucja *inkluzywna* występuje pomiędzy słowami A i B wtedy, gdy słowo B może wystąpić we wszystkich kontekstach, w których występuje słowo A, ale słowo A nie może wystąpić we wszystkich kontekstach, w których występuje słowo B (może wystąpić tylko w niektórych kontekstach, w których występuje słowo B).

Dystrybucja *zazębiająca się* występuje pomiędzy dwoma słowami wtedy, gdy dla obu porównywanych słów można znaleźć konteksty, w których może wystąpić pierwsze i drugie słowo, jak i konteksty, w których występuje tylko pierwsze albo tylko drugie słowo. [22, *ibid.*]

2.3.2. Relacje paradygmatyczne i syntagmatyczne

Relacje pomiędzy znaczeniami poszczególnych symboli mogą zostać podzielone na dwie grupy. Podział odpowiada językoznawczemu podziałowi na relacje:

1. paradygmatyczne
2. syntagmatyczne

Relacje **paradygmatyczne** danej jednostki językoznawczej to relacje, w które wchodzi ta jednoska z innymi jednostkami danego poziomu, dzięki temu, że mogą one występować w tych samych kontekstach. Innymi słowy relacje paradygmatyczne to relacje jakie zachodzą pomiędzy daną jednostką a innymi jednostkami, które posiadają wobec niej dystrybucję równoważną, inkluzywną bądź zazębiającą się. Relacja paradygmatyczna występuje np. pomiędzy słowami „krzesło” i „taboret”, ponieważ mogą one występować zamiennie w następującym zdaniu (które tworzy kontekst ich użycia): „Usiadłem na krześle/taborecie”.

Relacje **syntagmatyczne** to relacje w jakie wchodzi dana jednostka z innymi jednostki danego poziomu, dzięki temu, że tworzą one kontekst jej użycia. Relacja syntagmatyczna występuje na przykład pomiędzy słowem „krzesło” oraz słowom „siadać” dzięki temu, że występują razem w zdaniu: „Usiadłem na krześle”.

Relacje te mogą być wyróżnione w ramach semantyki. Tym niemniej pozajęzykowy kontekst użycia zdania, będący przedmiotem badań pragmatyki, również ma wpływ na relacje jakie występują pomiędzy znaczeniami poszczególnych słów. Wiąże się on ze specyficzną wiedzą, bądź cechami danego użytkownika języka. Jako przykład można podać sytuację, w której osoba wypowiadająca zdanie: „Usiadłem na krześle” jest królem. Wtedy krzesło przyjmuje specyficzną funkcję miejsca z którego sprawowane są rządy i staje się tronem. W takiej sytuacji zdanie: „Usiadłem na tronie” jest akceptowalne, niezależnie od tego czy przedmiot do którego odnosi się słowo „tron” w tym zdaniu, charakteryzuje się np. bogatym zdobnictwem, typowym dla monarszych tronów.

Rozdział 3

Słownik Semantyczny Języka Polskiego

3.1. Słowniki semantyczne

Czym jest *słownik semantyczny*? W ogólności można powiedzieć, że słownik taki to zbiór definicji słów występujących w jakimś języku naturalnym, zawierających informacje dotyczące semantyki tych słów. Tym niemniej definicja taka jest zbyt szeroka, gdyż większość tzw. ogólnych słowników, zawiera jakieś informacje na temat znaczenia poszczególnych słów. Słownik semantyczny musi zatem traktować znaczenie w sposób szczególny – powinno być ono w jednoznaczny sposób interpretowalne nie tylko przez człowieka, ale również przez maszyny.

Aby osiągnąć jednoznaczność interpretacji najlepiej jest wyróżnić pewne relacje semantyczne (patrz p. 2.3), które zachodzą pomiędzy słowami i opisać poszczególne słowa wykorzystując te relacje. Zatem słownik semantyczny byłby to słownik, który wykorzystuje pewną liczbę relacji semantycznych do opisu znaczenia poszczególnych słów.

Definicja 3.1. ***Słownik semantyczny*** *to zbiór słów wybranego języka naturalnego, których semantyka opisana jest za pomocą ograniczonej liczby ściśle zdefiniowanych relacji semantycznych.*

Niektóre słowniki semantyczne jako jednostki podstawowej nie przyjmują słów danego języka (ani nawet homonimów) lecz poszczególne znaczenia tych słów. Podejście to motywowane jest faktem, że większość słów posiada wiele rozmaitych, choć zbliżonych do siebie znaczeń (jest to zjawisko *polisemii* [22, s. 477]). Co więcej, obserwuje się, że niektóre znaczenia danego słowa pokrywają się ze znaczeniami innych słów. Wykorzystując tę obserwację wprowadzono pojęcie *synsetu*, czyli zbioru słów, które spośród wielu znaczeń w jakich mogą zostać użyte, posiadają jedno wspólne znaczenie. Zbiór ten jest wykorzystywany jako reprezentant wspólnego znaczenia słów należących do niego.

Poszczególne słowniki semantyczne będą zatem różniły się przede wszystkim *językiem*, który opisują, liczbą i rodzajem *relacji*, jakie zostały wykorzystane do opisu poszczególnych słów oraz podstawową *jednostką*, która jest opisywana (słowo, jedno ze znaczeń słowa, etc.).

3.1.1. WordNet jako przykład słownika semantycznego

Do najbardziej znanych słowników semantycznych wykorzystywanych na świecie należy z pewnością WordNet [8]. Pojęcie *synsetu* zostało właśnie wprowadzone przez jego twórców. WordNet jest słownikiem semantycznym języka angielskiego. Zawiera on 7 podstawowych relacji semantycznych, które wykorzystywane są do definiowania poszczególnych synsetów.

Ponieważ słownik ten okazał się niezwykle użyteczny, powstało wiele jego odpowiedników dla innych języków. Organizacje, które pracują nad poszczególnymi WordNetami stowarzyszone są w ramach *Global WordNet Association*[1]. Z informacji dostępnych na stronie tej organizacji wynika, że słowniki wzorowane na WordNecie istnieją dla niemal 40 języków. Ponadto, w ramach tej organizacji prowadzone są prace, które mają na celu wyodrębnienie zbioru najbardziej podstawowych synsetów, występujących we wszystkich językach oraz stworzenie indeksu, który zawierałby mapowania pomiędzy nimi.

Relacje

W WordNecie zastosowano 7 relacji do definiowania znaczeń poszczególnych słów:

1. synonimię
2. podobieństwo
3. antonimię
4. holonimię
5. hipernimię
6. hiponimię
7. meronimię

Relacja *synonimii* występuje pomiędzy znaczeniami dwóch słów, jeśli słowa te mogą zastępować siebie nawzajem we wszystkich zdaniach (kontekstach), bez naruszenia warunków prawdziwości tych zdań.

Relacja *podobieństwa* występuje pomiędzy znaczeniami dwóch słów, jeśli słowa te mogą zastępować siebie nazwajem w dużej części zdań (kontekstów), bez naruszenia warunków prawdziwości tych zdań.

Relacja *antonimii* zachodzi pomiędzy znaczeniami dwóch słów jeśli pomiędzy tymi słowami występuje asocjacja, będąca rezultatem ich częstego współwystępowania.

Relacja *holonimi* zachodzi pomiędzy znaczeniami słów X i Y (X jest holonimem Y), wtedy gdy Y nazywa przedmiot, który jest częścią przedmiotu nazywanego X.

Relacja *hipernimii* zachodzi pomiędzy znaczeniami słów X i Y (X jest hipernimem Y), gdy Y jest rodzajem X (X jest terminem bardziej ogólnym niż Y). Odpowiada ona dystrybucji inkluzywnej (patrz p. 2.3.1).

Relacja *hiponimii* zachodzi pomiędzy znaczeniami słów X i Y (X jest hiponimem Y), gdy X jest rodzajem Y (Y jest terminem bardziej ogólnym niż X).

Relacja *meronimii* zachodzi pomiędzy znaczeniami słów X i Y (X jest meronimem Y), gdy X nazywa przedmiot, który jest częścią przedmiotu nazywanego Y.

[1] www.globalwordnet.org

3.1.2. Zastosowania słowników semantycznych

Słowniki semantyczne są niezwykle przydatnym narzędziem wykorzystywanym w budowie algorytmów przetwarzania tekstu w językach naturlanych. Uważa się bowiem obecnie, że znajomość statystycznych własności danego języka nie jest wystarczająca do konstrukcji precyzyjnych algorytmów. Poniżej przedstawione są problemy, których skuteczne rozwiązanie zależy bez wątpienia od wiedzy na temat semantyki, a więc od istnienia odpowiednich słowników semantycznych.

Jednoznaczny wybór znaczenia

Podstawowy problem wszelkich algorytmów przetwarzających w sposób zaawansowany teksty w języku naturlanym, to problem jednoznacznego wyboru znaczenia słów użytych w danym kontekście. Człowiek, kiedy czyta tekst, nie ma żadnego problemu z wyborem właściwego znaczenia. Niestety, algorytmy automatycznego przetwarzania tekstu nie są w stanie wybrać trafnie jednego ze znaczeń, jedynie na podstawie wiedzy statystycznej [26].

Dzięki zawarciu w słownikach semantycznych informacji o relacjach semantycznych jakie zachodzą pomiędzy poszczególnymi słowami, możliwa jest konstrukcja algorytmów, które bazując na tych informacjach, są w stanie w sposób bardziej trafny wybrać właściwe znaczenie.

Tłumaczenie maszynowe

Innym zagadnieniem, związanym z wymienionym wcześniej, jest maszynowe tłumaczenie z jednego języka na inny. Również tutaj bez rozpoznania, które znaczenie wystąpiło w badanym zdaniu oraz bez informacji o tym, jak poszczególne znaczenia mają się do słów w drugim języku, nie jest możliwe uzyskanie adekwatnego tłumaczenia. Jeśli odpowiednie znaczenie nie zostanie rozpoznane w języku macierzystym, cały proces tłumaczenia staje się bezsensownym przekształcaniem jednych napisów w inne, z całkowitym pominięciem sensu, jaki zawarty był w tłumaczonym zdaniu.

W tym wypadku szczególnie cenne byłyby bilingwalne słowniki, w których poszczególne, odpowiadające sobie znaczenia, byłyby w sposób ścisły rozróżnione.

3.2. Cel powstania Słownika Semantycznego Języka Polskiego

Słownik Semantyczny Języka Polskiego, to wspólne przedsięwzięcie Uniwersytetu Jagiellońskiego i Akademii Górniczo-Hutniczej, realizowane pod kierownictwem prof. dr hab. Wiesława Lubaszewskiego. Jego angielska nazwa to Polish Semantic Dictionary, w skrócie PSD.

Słownik Semantyczny Języka Polskiego, w zamierzeniu autorów, będzie podstawowym narzędziem wykorzystywanym w automatycznej analizie semantycznej tekstów w języku polskim. Dotychczasowe próby oparcia analizy tekstu wyłącznie o zależności syntaktyczne (jak to ma miejsce w pracach inspirowanych badaniami Chomskiego), nie przyniosły zadowalających rezultatów. Dlatego też aby skutecznie przetwarzać teksty w języku polskim, konieczne jest stworzenie narzędzia, które opisywałoby semantykę tego języka.

3.3. Przyjęte założenia

Założenia przyjęte przy realizacji Słownika wypływają z refleksji nad podobnymi systemami realizowanymi w ciągu ostatnich lat, jak i ogólnej, filozoficznej refleksji nad rzeczywistością, której historia jest znacznie dłuższa.

3.3.1. Brzytwa Ockhama

Naczelnym założeniem, które przyświecało twórcom słownika jest słynna *brzytwa Ockhama* – zasada głosząca, że „nie należy mnożyć bytów ponad potrzebę". Dlatego wszędzie tam, gdzie pojawiało się pytanie – czy ważniejsza jest prostota systemu, czy też jego precyzja, skłaniano się ku pierwszej opcji. Z tego też względu liczba kategorii do których kwalifikowane są poszczególne słowa jest niewielka, w szczególności w porównaniu z ontologią Cyc.

3.3.2. Zdroworozsądkowy punkt widzenia

Słownik tworzony jest z punktu widzenia przeciętnego użytkownika języka – a nie, jak to ma miejsce w przypadku ontologii – z punktu widzenia naukowca specjalizującego się w dziedzinie, w której wykorzystywane jest opisywane słowo. Słownik, w zamierzeniu twórców, będzie wykorzystywany do analizy tekstów niespecjalistycznych, w których dominuje zdroworozsądkowe spojrzenie na rzeczywistość. Nie chcemy przez to powiedzieć, że będzie on ignorował ustalenia nauk szczegółowych, aczkolwiek nie będą one stanowiły podstawowego kryterium uznania jakiegoś opisu za właściwy. Podstawowym kryterium akceptacji opisu są intuicje użytkowników języka, które znajdują swoje odzwierciedlenie w tworzonych przez nich tekstach i wypowiedziach. Charakterystyczny przykład, który powinien uzmysłowić to założenie jest następujący – w słowniku akceptowalne jest przypisanie *słońcu* akcji: *wschodzić*, *zachodzić*. Odwołując się do wiedzy z zakresy fizyki, można uznać, że opis ten jest nieadekwatny. Praktyka językowa pokazuje jednak, że nawet fizycy, w niespecjalistycznych tekstach i dyskusjach, używają tych określeń. Dlatego też określenia te muszą się znaleźć w Słowniku.

3.3.3. Deskrypcyjna teoria znaczenia

W filozofii dyskutowany jest problem znaczenia słów odnoszących się do rodzajów naturalnych[2], które stanowią jedną z podstawowych grup słów, występujących w Słowniku. Jedno stanowisko, zwane w ogólności deskrypcyjną teorią znaczenia, stwierdza, że znaczenie takich słów jest tożsame z pewną grupą cech charakterystycznych dla przedstawicieli tych rodzajów.

Drugie stanowisko, reprezentowane np. przez Kripkego, odwołuje się do *istoty gatunkowej*, *aktu chrztu* oraz *łańcucha przyczynowo-skutkowego*. Zakłada ono, że każde słowo tego typu, zostało wprowadzona do języka na zasadzie specyficznego aktu chrztu, w którym to akcie po raz pierwszy orzeczono to słowo o jakimś przedstawicielu danego rodzaju naturalnego. Nadto zaś, zakłada się, że rodzaje naturalne faktycznie występują w rzeczywistości, a ich przedstawiciele mają tożsamą *istotę gatunkową*. Na gruncie tych założeń poprawne orzecze-

[2] *Rodzaj* jest rozumiany nie tylko w sensie biologicznym, ale szerzej – jako zbiór obiektów które występują naturalnie w rzeczywistości, tzn. nie zostały wytworzone przez człowieka, a którym odpowiada jedno słowo.

nie pojęcia o przedmiocie, będzie uzależnione od łańcucha przyczynowo-skutkowego, który zapoczątkowany został w akcie chrztu, a który podtrzymywany był przez wspólnotę użytkowników języka. Dane słowo będzie można orzec o wybranym obiekcie tylko wtedy, gdy obiekt ten ma identyczną istotę gatunkową z obiektem, który został ochrzczony tym słowem.

Nie rozstrzygamy tutaj, które teoria jest poprawna. Jednakże stwierdzenie czym jest istota gatunkowa jest dla bardzo wielu rodzajów naturalnych niezwykle problematyczne. Dlatego też wybieramy pierwsze podejście – ponadto ma ono dla nas zdecydowanie większą wartość metodologiczną, gdyż z założenia opisy poszczególnych kategorii czy słów zadane są w postaci zbioru cech. Nie rozstrzygamy czy cechy te są istotne – wystarczy aby były charakterystyczne.[3]

3.4. Realizacja modelu teoretycznego

Słownik Semantyczny Języka Polskiego w następujący sposób realizuje definicję 3.1: poszczególne słowa występujące w języku polskim definiowane są za pomocą szeregu relacji semantycznych, które opisane zostały w punkcie 3.4.2. Ponadto słowa podzielone zostały na szereg kategorii semantycznych – zgodnie z przeświadczeniem twórców słownika, mówiącym, że słowa języka naturalnego przynależą do pewnych grup, co znacząco wpływa na ich semantykę i sposób użycia.

Przykładowe hasła opracowane z wykorzystaniem przedstawionych relacji i kategorii semantycznych znajdują się w dodatku A.

3.4.1. Kategorie

Słowa występujące w Słowniku Semantycznym podzielone zostały na kategorie, zgodnie z ich semantyczną charakterystyką. Wprowadzenie kategoryzacji słów wynika również z faktu, że słowa należące do różnych kategorii, opisywane są za pomocą różnych relacji (por. `HUMAN` oraz `EVENT`).

Przynależność do danej kategorii oznacza, że słowo dziedziczy wszystkie cechy semantyczne, które przypisane są kategorii, do której ono należy. Dane słowo może należeć do kilku kategorii, lecz sytuacja ta jest raczej rzadko spotykana. Wśród relacji, które służą do opisu słów występuje relacja `IS_A_KIND_OF`. Relacja ta posiada podobne własności co relacja przynależności do danej kategorii. Należy jednak zwrócić uwagę na dwie istotne różnice. Po pierwsze: każde słowo musi należeć do co najmniej jednej kategorii semantycznej, natomiast nie jest wymagane aby wchodziło w relację `IS_A_KIND_OF` z jakimkolwiek ze słów. Po drugie zaś: przynależność do kategorii nie jest relację semantyczną. Co prawda, dla niektórych kategorii, jak np. `HUMAN` przynależność jakiegoś słowa do niej implikuje występowanie relacji `IS_A_KIND_OF` pomiędzy tym słowem a słowem „człowiek”. Tym niemniej sytuacja ta nie ma miejsc w przypadku wszystkich kategorii, dlatego też przynależność do kategorii nie może być utożsamiona z relacją `IS_A_KIND_OF`.

Poniżej przedstawiamy pobieżnie kategorie zaproponowane dotychczas przez twórców słownika. List ta nie jest jednak zamknięta i wstępne badania zawartości Słownika wykazały, że będzie ona musiała zostać rozszerzona. Propozycje jej rozszerzenia zostały przedstawione na końcu.

[3] *Cecha charakterystyczna* pozwala odróżnić przedstawicieli różnych rodzajów, *cecha istotna* zaś (o ile istnieje) przysługuje w sposób konieczny wszystkim przedstawicielom rodzaju.

HUMAN

Do kategorii HUMAN należą słowa, których denotacją są wyłącznie przedstwiciele *gatunku ludzkiego*. Słowa te odnoszą się zatem do ludzi, ale nie do ich stanów mentalnych, czy części ich ciał. Przykład: *strażak*.

PLANT

Do kategorii PLANT należą słowa, które odnoszą się zarówno do *roślin* (zwykle samożywnych organizmów fotosyntetyzujących, niezdolnych do poruszania się) jak i *grzybów* (zwykle organizmów cudzożywnych, niefotosyntetyzujących, niezdolnych do poruszania się). Przykład: *róża*.

ANIMAL

Do kategorii ANIMAL należą słowa, które odnoszą się do *zwierząt* (zwykle organizmów cudzożywnych, zdolnych do poruszania się), ale nie do człowieka. Przykład: *koń*.

LOCATION

Do kategorii LOCATION należą słowa, które odnoszą się do obiektów, wobec których w naturalny sposób mogą być *lokalizowane zdarzenia*. Obiekty te posiadają zwykle rozmiary zbliżone do wielkości człowieka lub większe. Przykład: *bar*.

INSTRUMENT

Do kategorii INSTRUMENT należą słowa, które odnoszą się do *narzędzi*, wykorzystywanych przez człowieka do wykonywania określonej pracy. Przykład: *łopata*.

PHYS_OBJ

Do kategorii PHYS_OBJ należą słowa odnoszące się do *obiektów fizycznych* (masywnych obiektów rozciągłych, trwających w czasie), które nie mogą być zakwalifikowane do żadnej z wcześniej wymienionych kategorii (HUMAN, PLANT, ANIMAL, LOCATION i INSTRUMENT). Przykład: *głośnik*.

SET

Do kategorii SET należą słowa, które odnoszą się do *zbiorów* i *kolekcji*. Zbiór od kolekcji rożni się tym, że nie istnieje kryterium przynależności do zbioru – zbiór jest tożsamy ze swoją ekstensją, dlatego utożsamia się zbiory posiadające tę samą ekstensję. Kolekcje natomiast posiadają stowarzyszone kryterium kwalifikacji, dzięki czemu to co stanowi jeden zbiór (organizmy wyposażone w serce i organizmy wyposażone w nerkę) rozpada się na dwie kolekcje. Przykład: *grupa [osób]*.

STRUCTURE

Do kategorii STRUCTURE należą słowa, które odnoszą się do różnego rodzaju *struktur*. Struktury, w odróżnieniu od zbiorów, charakteryzują się tym, że elementy w nich występujące powiązane są identyfikowalnymi relacjami, które istotne są z punktu widzenia struktury, jako całości. Przykład: *drużyna [piłkarska]*.

NUMBER

Do kategorii NUMBER należą słowa, które mogą być użyte do określenia *liczności* jakiegoś zbioru. Naturalnie kwalifikują się do niej liczebniki, ale również słowa, które określają liczność w sposób mniej precyzyjny – np. *mnóstwo*.

TIME

Do kategorii TIME należą słowa, które odnoszą się w pewien sposób do *czasu* – mogą być to np. jego odcinki (*tydzień*) czy punkty (*chwila*).

ABSTRACT_OBJ

Do kategorii ABSTRACT_OBJ należą słowa, które odnoszą się do *obiektów abstrakcyjnych* (przeciwieństwo obiektów fizycznych) nie kwalifikujące się jednak do żadnej z kategorii: SET, STRUCTURE, NUMBER, TIME. Przykład: *wampir*.

EVENT

Do kategorii EVENT należą słowa, która odnoszą się do zdarzeń. Przykład: *kupowanie*.

NAME

Do kategorii NAME należą słowa, które są nazwami własnymi. Przykład: *Hamlet*.

SELF

Do kategorii SELF należą słowa, które trudno jest zakwalifikować do którejkolwiek z pozostałych kategorii. Ponieważ do kategorii tej mogą trafić słowa, które nie mają ze sobą nic wspólnego, z przynależności do tej kategorii nie można wyciągnąć żadnych wniosków na temat semantyki danego słowa (poza wnioskiem, że jest ona dosyć specyficzna).

Nowe kategorie

Wśród proponowanych nowych kategorii można wymienić: STATE, BODY_PART, SELF_ANIMATE, ELEMENT, INSTITUTION, METER, MENTAL_OBJ, RELATION, MENTAL_STATE, PROCESS, METAPHOR. Należy zwrócić uwagę, że kategorie te są na razie tylko proponowane i nie wiadomo, czy wszystkie zostaną uwzględnione w ostatecznej wersji słownika, dlatego też nie podajemy ich definicji.

3.4.2. Relacje

Relacje występujące w Słowniku, służące do opisu zależności występujących pomiędzy znaczeniami poszczególnych słów, podzielone zostały na dwie grupy. Do pierwszej grupy relacji należą:

1. SYNONYMY
2. SIMILAR_TO
3. IS_A_KIND_OF
4. IS_A
5. IS_A_PART_OF
6. CONSISTS_OF

Są to semantyczne relacje dwuargumentowe, spośród których 4 pierwsze, to relacje paradygmatyczne, a 2 ostatnie to relacje syntagmatyczne. Są to ogólne relacje, które mogą być

zaobserwowane w kontekstach zdaniowych, niezależnie od kontekstu pozajęzykowego. Są one zatem opisywane głównie przez semantykę.

Do drugiej grupy relacji należą:

1. SOURCE
2. DESTINATION
3. ROLE
4. ACTION
5. STATE
6. ACTOR
7. OBJECT
8. FROM
9. TO
10. INSTRUMENT
11. TIME
12. PLACE
13. MOOD

Są to relacje pragmatyczne, spośród których większość to relacje syntagmatyczne. Do ich zaobserwowania potrzebny jest zwykle kontekst pozajęzykowy. W ich opisie może występować dodatkowe słowo, sygnalizowane za pomocą słowa kluczowego RELATED_TO, które odnosi się do obiektu, którego wystąpienie w kontekście pozajęzykowym konieczne jest dla zaistnienia badanej relacji. Relacje te zatem mogą być dwu lub trój-argumentowe. Są one głównie przemiotem badania pragmatyki.

Zaproponowany podział nie demarkuje jednak w sposób ścisły grup poszczególnych relacji. W przypadku pierwszych – odnośne konteksty pragmatyczne oddziałują na wymienione relacje, a w drugim – konteksty zdaniowe. Podział ten wskazuje jedynie główną komponentę, która ma zasadniczy wpływ na kształtowanie wymienionych relacji.

Relacje z pierwszej grupy wykorzystywane są do opisu słów, niezależnie od tego, do której kategorii należą. Natomiast do opisu słów należących do wszystkich kategorii, z wyjątkiem kategorii EVENT, wykorzystywane są następujące relacje z grupy drugiej: SOURCE, DESTINATION, ROLE, ACTION, STATE. Do opisu słów należących do kategorii EVENT wykorzystywane są zaś relacje: ACTOR, OBJECT, FROM, TO, INSTRUMENT, TIME, PLACE, MOOD.

SYNONYMY

SYNONYMY (*synonimia*) – relacja paradygmatyczna występująca pomiędzy słowami, których zamiana w dowolnym kontekście zdaniowym nie powoduje zmiany znaczenia zdania, z którego pochodzi ów kontekst. Przykład: *krew* SYNONYMY: *jucha*.

SIMILAR_TO

SIMILAR_TO (*podobieństwo znaczeń*) – relacja paradygmatyczna, podobna do synonimii, różniąca się od niej tym, że zamiana (bez zmiany znaczenia zdania) jednego słowa na inne, którego znaczenie jest podobne, nie jest możliwa we wszystkich kontekstach zdaniowych, lecz jedynie w dużej ich części. Przykład: *obuwie* SIMILAR_TO: *buty*.

IS_A_KIND_OF

`IS_A_KIND_OF` (*hiponimia*) – relacja paradygmatyczna. Dane słowo występuje w tej relacji z innym słowem, jeśli desygnat tego słowa może być opisany drugim słowem, którego znaczenie jest ogólniejsze niż znaczenie danego słowa. Ekstensja danego słowa jest podzbiorem właściwym ekstensji słowa drugiego. Przykład: *szpital* `IS_A_KIND_OF`: *budynek*.

IS_A

`IS_A` (*hiperonimii*) – relacja paradygmatyczna, będąca (w pewnym stopniu) relacją symetryczną względem relacji `IS_A_KIND_OF`. Dane słowo występuje w tej relacji z innym słowem (*resp.* wyrażeniem), jeśli desygnat tego słowa może być opisany drugim słowem (*resp.* wyrażeniem), którego znaczenie jest węższe niż znaczenie badanego słowa. Ekstensja tego słowa jest nadzbiorem właściwym ekstensji słowa (*resp.* wyrażenia) drugiego. Przykład: *statek* `IS_A`: *parostatek*.

IS_A_PART_OF

`IS_A_PART_OF` (*meronimia*) – relacja syntagmatyczna, występująca pomiędzy słowem, które odnosi się do obiektu, który jest częścią innego obiektu, a słowem które odnosi się do tego drugiego obiektu, którego pierwszy jest częścią. Przykład: *kierownica* `IS_A_PART_OF`: *samochód*.

CONSISTS_OF

`CONSISTS_OF` (*holonimia*) – relacja syntagmatyczna, symetryczna (w pewnym stopniu) do relacji `IS_A_PART_OF`. Występuje ona pomiędzy słowem, które odnosi się do obiektu, który składa się z innego obiektu, a słowem odnoszącym do obiektu, z którego ten pierwszy się składa. Przykład: *stół* `CONSISTS_OF`: *blat*.

SOURCE

Relacja `SOURCE` opisuje[4] *konceptualne źródło* słowa. Ma ona zastosowanie głównie wobec słów, które odnoszą się do obiektów abstrakcyjnych i wskazuje obszar intelektualnej działalności człowieka, który jest ich źródłem. Przykład: *centaur* `SOURCE:` *mitologia*.

DESTINATION

`DESTINATION` to relacja, która opisuje[5] *przeznaczenie* obiektu, do którego odnosi się dane słowo. Ma ona zastosowanie jedynie wobec artefaktów (obiektów, które są wytworem człowieka). Przykład: *młotek* `DESTINATION`: *wbijanie*.

ROLE

`ROLE` to relacja, która opisuje *role*, w jakich może wystąpić obiekt, do którego odnosi się dane słowo. Role te należy rozumieć, jako specyficzne przypadki zastosowań danego przedmiotu. Przykład: *krzesło* `ROLE`: *tron* `RELATED_TO`: *król*.

[4] Mówimy, że relacja R opisuje własność W, nazywaną słowem S_2, słowa S_1, jeżeli relacja R występuje pomiędzy własnością W, a słowem S_1. Innymi słowy: własność W słowa S_1 nazywa się S_2.

[5] Mówimy, że relacja R opisuje własność W, nazywaną słowem S_2, obiektu O, do którego odnosi się słowo S_1, jeżeli relacja R występuje pomiędzy własnością W, a obiektem O. Innymi słowy: własność W przedmiotu O nazywa się S_2.

ACTION

ACTION to relacja, która opisuje *działania*, których podmiotem lub przedmiotem może być obiekt, do którego odnosi się dane słowo. W ramach tej relacji wyróżnione zostały trzy aspekty:

1. POSITIVE – relacja opisuje pozytywne działania, w których odniesienie danego słowa jest podmiotem. Przykład: *lek* ACTION POSITIVE: *leczyć*. (*z jakiego punktu widzenia pozytywne – przeznaczenia? Czy zatem dla pistoletu działaniem pozytywnym jest zabijanie?*)
2. NEGATIVE – relacja opisuje negatywne działania, w których odniesienie danego słowa jest podmiotem. Przykład: *lek* ACTION NEGATIVE: *szkodzić*.
3. PASSIVE – relacja opisuje działania, w których odniesienie danego słowa jest przedmiotem. Przykład: *lek* ACTION PASSIVE: *ulegać przeterminowaniu*.

STATE

STATE to relacja, która opisuje *stany* przedmiotu, do którego odnosi się dane słowo. W ramach tej relacji zostały wyróżnione dwa aspekty:

1. POSITIVE – relacja opisuje pozytywne stany obiektu, do którego odnosi się dane słowo. Przykład: *organizm* STATE: *żywy*.
2. NEGATIVE – relacja opisuje negatywne stany obiektu, do którego odnosi się dane słowo. Przykład: *organizm* STATE: *martwy*.

ACTOR

Relacja ACTOR opisuje *sprawcę* zdarzenia, do którego odnosi się dane słowo. Przykład: *sterowanie* ACTOR: *kapitan* RELATED_TO: *statek*.

OBJECT

Relacja OBJECT opisuje *przedmiot* podlegający procesowi lub zdarzeniu, do którego odnosi się dane słowo. Przykład: *wałkowanie* OBJECT: *ciasto*.

FROM

Relacja FROM opisuje *konceptualne źródło* zdarzenia, do którego odnosi się badane słowo. Konceptualnym źródłem zdarzenia jest obiekt, stan lub okoliczność, w którym zdarzenie ma swój początek. Konceptualne źródło zdarzenia niezależne jest od sprawcy zdarzenia. Rozpatrzmy dwa zdania: „Jaś daje książkę Małgosi” oraz „Małgosia kradnie książkę Jasiowi”. Chociaż podmiotem zdarzenia opisanego w pierwszym zdaniu jest Jaś, a opisanego w drugim – Małgosia, konceptualnym źródłem zdarzenia w obu przypadkach jest *Jaś*. Przykład: *darować* FROM: *darczyńca*.

TO

Relacja TO opisuje *konceptualny cel* zdarzenia, do którego odnosi się badane słowo. Konceptualnym celem zdarzenia jest obiekt, stan lub okoliczność, w którym zdarzenie się kończy . Jest ono drugim, po konceptualnym źródle, elementem konceptualnego kierunku zdarzenia. W przykładzie przedstawionym w opisie relacji FROM konceptualnym celem zdarzeń dawania i kradnięcia byłaby *Małgosia*. Przykład: *darować* TO: *obdarowany*.

INSTRUMENT

Relacja `INSTRUMENT` opisuje *narzędzie*, które zostało wykorzystane w realizacji zdarzenia, do którego odnosi się badane słowo. Przykład: *strzelanie* `INSTRUMENT`: *pistolet*.

TIME

Relacja `TIME` opisuje *czas*, w którym miało miejsce zdarzenie, do którego odnosi się badane słowo. Większość zdarzeń może odbywać się w dowolnym czasie, dlatego relacja ta ma głównie zastosowanie w przypadku zdarzeń jednostkowych, nazywanych najczęściej nazwami własnymi. Przykład: *Bitwa pod Grunwaldem* `TIME`: *rok 1410*.

PLACE

Relacja `PLACE` opisuje *miejsce*, w którym odbywa się zdarznie, do którego odnosi się badane słowo. Przykład: *wspinaczka* `PLACE`: *góry*.

MOOD

Relacja `MOOD` odnosi się do *sposobu*, w jaki może przebiegać (lub przebiegało) zdarzenie, do którego odnosi się badane słowo. Przykład: *bunt* `MOOD`: *gwałtowny*.

Rozdział 4

Ontologia Cyc

4.1. Ontologie

4.1.1. Ontologie w filozofii i informatyce

Wykorzystywane w informatyce pojęcie *ontologii* zostało zaczerpnięte z filozofii, gdzie można spotkać się z wieloma jego definicjami. Choć pojawiło się ono dopiero w XVII wieku za sprawą Rudolfa Gocleniusa (*Lexicon philosophicum, quo tantam clave philosophiae fores aperiuntur*, 1613), to w sposób ścisły łączy się z pojęciem metafizyki, którego historia sięga czasów Arystoteles, autora *tego co następują po fizyce*, czyli *Metafizyki*.

Obie dziedziny filozofii traktują o bycie, o tym co istnieje, bądź może istnieć, aczkolwiek przypisuje się im zwykle różny zakres przedmiotów badania. *Oksfordzki Słownik Filozoficzny* [2, s. 269] zamieszcza następującą definicję:

> **Ontologia** (ang. *ontology*). Termin wywodzący się z greckiego słowa oznaczającego byt, ale ukuty w XVII w. na oznaczenie gałęzi *metafizyki zajmującej się tym co istniej. [...]

Zatem traktuje on ontologię jako ten dział metafizyki, który zajmuje się tym co istniej, w domyśle faktycznie, a nie jedynie potencjalnie. Z kolei Roman Ingarden [16, s. 33–42] definiował ontologię, jako tę naukę, która bada wszystko to co może istnieć oraz sposoby istnienia, natomiast metafizykę, jako dział ontologii badający obiekty istniejące faktycznie.

To niemałe zamieszanie terminologiczne zostało, przynajmniej w pewnym stopniu, zredukowane na gruncie informatyki. Tutaj również możemy spotkać się z różnymi definicjami, aczkolwiek część z nich stara się opisać filozoficzne a nie informatyczne pojęcie ontologii. Np. Tom Gruber w [12] podaje krótką definicję:

> Ontologia jest specyfikacją konceptualizacji.[1]

[1] *An ontology is a specification of a conceptualization.* [Wszelkie cytaty angielskie będą podawane w moim przekładzie – A. P.]

W innym miejscu[2] napisane jest, że specyfikacja ta powinna być formalna, a konceptualizacja dzielona przez pewną wspólnotę użytkowników ontologii. Definicję tę przyjmiemy jako adekwatną dla zastosowań w informatyce i będziemy wykorzystywać w dalszej części tego dokumentu.

Z kolei Clay Shirky [35] podaje definicję zbliżoną do tej zaproponowanej przez Ingardena:

> Główny wątek ontologii, w sensie filozoficznym, to studia nad bytami i relacjami pomiędzy nimi. Pytanie, które zadaje ontologia to: jakie rodzaje rzeczy istnieją lub mogą istnieć w świecie oraz jakie rodzaje relacji mogą one posiadać wobec siebie nawzajem. Ontologia mniej koncentruje się na tym co jest [faktycznie], niż na tym co jest potencjalnie.[3]

Co prawda zauważa on, że funkcjonuje również definicja zaproponowana przez Grubera, lecz ostatecznie stwierdza:

> Wspólny wątek obu tych definicji to esencja, „bycie". Dla pewnej dziedziny, o jakich rodzajach rzeczy możemy powiedzieć, że istnieją w tej dziedzinie oraz o jakich relacjach możemy powiedzieć, że zachodzą pomiędzy nimi?[4]

Podkreśla on zatem, charakterystyczne dla filozofii zagadnienie *istnienia*, które w informatyce mogłoby wprowadzić jedynie zbędną niejasność.

Pierwsza definicja jest lepsza, gdyż unika tego pojęcia, mówiąc o formalnej specyfikacji pewnej konceptualizacji. Nie rozstrzyga się tutaj, czy jakieś obiekty istnieją czy nie – mówi się jedynie, że użytkownicy danej ontologii zgadzają się, że pewna konceptualizacja jest im wspólna. Jest to lepsze z metodologicznego punktu widzenia, gdyż przenosi akcent z teorii na zastosowania.

4.1.2. Ogólna charakterystyka ontologii

Definicja ontologii, z której będziemy korzystać jest następująca:

Definicja 4.1. ***Ontologia*** *jest formalną specyfikacją wspólnej konceptualizacji jakiejś dziedziny wiedzy.*

Każdy z elementów występujących w definicji (specyfikacja, konceptualizacja, dziedzina wiedzy) może zostać dookreślony na wiele różnych sposobów, dzięki czemu możemy uzyskać wiele rozmaitych systemów ontologicznych.

---

[2] Dostępne: http://wiki.ontoworld.org/index.php/Ontology

[3] *The main thread of ontology in the philosophical sense is the study of entities and their relations. The question ontology asks is: What kinds of things exist or can exist in the world, and what manner of relations can those things have to each other? Ontology is less concerned with what is than with what is possible.*

[4] *The common thread between the two definitions is essence, „Is-ness." In a particular domain, what kinds of things can we say exist in that domain, and how can we say those things relate to each other?* [Tłumaczenie to nie jest literalne, gdyż po pierwsze: pewne terminy, wykorzystane przez autora (jak esencja) mają swoją długą historię w filozofii i nie zamierzam proponować tutaj jakiś neologizmów, jak czyni to autor (Is-ness); po wtóre zaś: czasownik *relate* nie ma dobrego polskiego odpowiednika, co czyni przekład literalny stylistycznie nieakceptowalnym.]

Formalna specyfikacja

Formalna specyfikacja wyrażana jest najczęściej w jakimś ontologicznym języku formalnym. Istotne jest odróżnienie języka specyfikacji ontologii od samej ontologii, gdyż jeden język może być wykorzystywany w wielu ontologiach. Przykładami języków służących do definiowani ontologii są:

1. CycL [20]
2. DAML+OIL [15]
3. FLogic [18]
4. KIF [11]
5. LOOM [33]
6. OCML [27]
7. OIL [9]
8. Ontolingua [13]
9. OWL [25]
10. RDF(S) [24]
11. SHOE [14]
12. XOL [31]

Konceptualizacja

Konceptualizacja wybranej dziedziny wiedzy może być przeprowadzona na bardzo wiele sposobów. Tym niemniej w systemach ontologicznych[5] wykorzystuje się zwykle następujące pojęcia:

1. indywidua (instancje)
2. koncepty (pojęcia, kolekcje)
3. własności
4. relacje (predykaty)
5. funkcje
6. procesy
7. asercje

Indywidua reprezentują zwykle pojedyncze, indywidualne obiekty, występujące w opisywanej dziedzinie wiedzy. W języku naturalnym do obiektów tych odnosimy się wykorzystując nazwy własne, wyrażenia deiktyczne oraz zaimki (np. Margaret Thatcher, ten, on), zaś w językach sztucznych za pomocą stałych zinterpretowanych oraz zmiennych wolnych, którym przypisano określoną wartość.
Przykład: gdy dziedziną wiedzy jest geografia: *Polska*

Koncepty (pojęcia) reprezentują klasy indywiduów. W zależności od przyjętej teorii bazowej poszczególne klasy obiektów mogą być formułowane w sposób całkowicie arbitralny (jak to ma miejsce w teorii mnogości, w której pojęcie utożsamia się ze zbiorem) lub nie (jak to ma miejsce w języku naturalnym). W pierwszym wypadku pojęcie będzie tożsame ze swoją ekstensją (czyli zbiorem indywiduów, o których pojęcie to jest orzekane), w drugim przypadku – przynależność do ekstensji będzie implikowała pewne własności indywiduum. Wynika z tego również fakt, że w pierwszym wypadku ekstensja pojęcia jest niezmienna w czasie, natomiast w drugim – może się zmieniać. Pojęcia czasami za-

[5] Wyrażenie *systemy ontologiczne* używane jest zamiennie z terminem *ontologia* – są one tutaj traktowane jako synonimy.

stępowane są jednoargumentowymi predykatami.
Przykład (geografia): *państwo*.

Własności służą do opisu poszczególnych indywiduów lub pojęć. W pierwszym wypadku badana własność przynależy wybranemu indywiduum, w drugim zaś, przynależy ona wszystkim indywiduom, które należą do ekstensji danego pojęcia. Własności bywają zastępowane czasami jednoargumentowymi predykatami (`isRed(Brick)`) albo dwuargumentowymi predykatami, spośród których drugi argument reprezentuje własność (`hasProperty(Brick, RedColo`
Przykład (geografia): *wysokość n.p.m.*

Relacje (predykaty) służą do opisu związków jakie zachodzą pomiędzy poszczególnymi indywiduami a pojęciami (np. relacja *podpadania pod dane pojęcie*), indywiduami a indywiduami (np. relacja *kochania*: „Horacio kocha Magę"), etc. Własnością relacji jest ich arność, czyli liczba obiektów, pomiędzy którymi relacja ta zachodzi. Najczęściej relacje są dwu lub trój-argumentowe (binarne i ternarne), ale zdarzają się systemy, w których arność jest większa, bądź może przyjmować dowolne wartości. Zwykle określa się również typy argumentów, jakie mogą wystąpić na danej pozycji w danej relacji. Typ ten określany jest poprzez wskazanie pojęcia, pod które musi podpadać argument występujące na danej pozycji.
Przykład (geografia): *leżeć na południe od*.

Funkcje to specyficzny typ relacji, w których ostatni argument jest wyznaczany w sposób jednoznaczny przez poprzedzające go argumenty (tzn. początkowe argumenty nie mogą wchodzić w daną relację z więcej niż jednym argumentem, który występuje na ostatniej pozycji). Funkcje wykorzystywane są w systemach ontologicznych głównie ze względu na to, że pozwalają odnosić się do obiektów, które nie posiadają bezpośredniej reprezentacji w ontologii, bądź dlatego, że nie jest ona wymagana (a jej wprowadzenie spowodowałoby nadmierne przeciążenie ontologii), bądź też dlatego, że obiekt ten nie jest w pełni rozpoznany.
Przykład (geografia): *stolica kraju*.

Procesy to najsłabiej rozpracowany element systemów ontologicznych [28]. Ich złożoność i różnorodność powoduje, że są on szczególnie trudne do reprezentowania. Jeśli występują w jakiejś ontologii, to zwykle towarzyszą im pojęcia podmiotu i przedmiotu procesu, czasu i miejsca występowania, etc. Procesy występują w ontologiach, w których istotne jest reprezentowanie zmian, którym podlegają obiekty występujące w danej dziedzinie wiedzy.
Przykład: (geografia): *dryf [kontynentów]*.

Asercje służą do wyrażania zależności jakie występują pomiędzy wymienionymi wcześniej elementami. Najprostsze asercje stwierdzają, np. że jakaś relacja zachodzi pomiędzy dwoma konceptami (w Cyc asercja `(#$genls #$Cat #$Vertebrate)` stwierdza, że koty są kręgowcami). Asercje wyrażają jednak również bardziej skomplikowane zależności, wykorzystując aparat logiczny danej ontologii (w Cyc asercja

```
(#$domainAssumptions #$HumanPhysiologyMt
       (#$implies
           (#$isa ?ANIMAL #$Vertebrate) (#$isa ?ANIMAL #$Person)))
```

stwierdza, że w dziedzinie *ludzka fizjologia* panuje założenie, że wszelkie kręgowce, które w niej występują, to ludzie).

Pełny repertuar wyżej przedstawionych pojęć wykorzystywany jest tylko w najbardziej zaawansowanych systemach ontologicznych, głównie ze względu na to, że przetwarzanie tak skomplikowanej wiedzy nastręcza wiele trudności. Często korzysta się tylko z ich części, co może prowadzić do redukcji ekspresywności danej ontologii.

Dziedziny wiedzy

Systemy ontologiczne można podzielić na dwie grupy, ze względu na dziedziny wiedzy, które są w nich reprezentowane:
1. ontologie dziedzinowe
2. ontologie ogólne

W przypadku pierwszych opisywana dziedzina wiedzy jest na tyle wąska, że można określić ją mianem wiedzy eksperckiej. Jako przykład można podać ontologię Gene Ontology [36], która opisuje geny i ich produkty.

Natomiast dziedziną wiedzy ontologii drugiego typu jest zwykle szeroko rozumiana rzeczywistość ujmowana w sposób mniej lub bardziej abstrakcyjny. Potrzeba powstanie systemów drugiego typu bierze się m.in. stąd, że poszczególne ontologie dziedzinowe, aby mogły ze sobą współpracować, potrzebują wspólnej bazy, która obejmowałaby mniej i bardziej abstrakcyjne pojęcia, występujące we współpracujących ontologiach. Konieczność uzgadniania tych pojęć za każdym razem byłaby niezwykle czasochłonna, a nierzadko w ogóle niewykonalna. Z tego też względu powstają ontologie ogólne, które mogą być wykorzystywane właśnie jako rezerwuar najbardziej ogólnych pojęć.

Inną dziedziną zastosowania ontologii ogólnych jest np. przetwarzanie języka naturalnego, gdzie ogólna wiedza na temat świata jest niezbędna dla właściwego działania zaawansowanych algorytmów.

Przykładem ontologii ogólnej jest Cyc [19].

4.1.3. Zastosowania systemów ontologicznych

Systemy ontologiczne są coraz szerzej wykorzystywane w różnych dziedzinach informatyki: sztucznej inteligencji, wyszukiwaniu informacji, ekstrakcji i integracji wiedzy, przetwarzaniu języka naturlanego, systemach agentowych, modelowaniu, inżynierii oprogramowania, zarządzaniu konfiguracjami, bibliotekach elektronicznych, integracji baz danych [1, 6, 4, 28]. U podstaw ich sukcesu leży formalna reprezentacja wiedzy, dzięki której możliwe jest wyciąganie wniosków nie zawartych *explicite* w tych systemach, poprzez zastosowanie algorytmów automatycznego dowodzenia twierdzeń.

Ponieważ semantyka tych systemów określona jest w sposób ścisły, niezależne systemy, wykorzystujące tę samą ontologię, interpretują tak samo zawartą w nich wiedzę. Ma to szczególne znaczenie dla systemów wiedzy rozproszonej, gdzie współpraca pomiędzy różnymi systemami jest niemożliwa bez ustalenia jednoznacznej interpretacji zawartych w nich danych.

4.1.4. Znane systemy ontologiczne

W chwili obecnej na świecie powstaje wiele systemów ontologicznych. Powstają również specjalizowane wyszukiwarki[6] skoncentrowane na wyszukiwaniu informacji w wolnodostępnych ontologiach.

[6] http://swoogle.umbc.edu/

Do najszerzej wykorzystywanych ogólnych ontologii należy zaliczyć:
1. Cyc [19]
2. SUMO [28]

4.2. Historia i zastosowania Cyc

Głównym projektantem i pomysłodawcą ontologii Cyc jest Doug Lenat, do 1984 profesor Uniwersytetu Stanforda. Po opuszczeniu uniwersytetu zaczął on pracować w Microelectronics and Computer Technology Corporation (MMC) nad projektem Cyc. W 1994 opuścił MMC i założył firmę Cycorp, której głównym celem był rozwój ontologii Cyc.

Ontologia Cyc pomyślana była jako specyficzna baza wiedzy, zawierająca informacje stanowiące zrąb tego co zwykle nazywa się zdrowym rozsądkiem – czyli zbiorem przekonań dzielonych przez większość dorosłych, wykształconych przedstawicieli gatunku ludzkiego. Wraz z bazą wiedzy rozwijany był *silnik wnioskujący*, który na podstawie zgromadzonych faktów potrafi wysnuwać tezy, które nie są bezpośrednio w niej zawarte[7].

Główny cel jaki ma zostać osiągnięty dzięki budowie ontologii Cyc, to zapełnienie luki jaka występuje pomiędzy ludzką wiedzą a wiedzą jaką dysponuje komputer. Olbrzymie bazy wiedzy, które spotykane są w systemach komputerowych, niejednokrotnie przekraczają swoim zakresem możliwości poznawcze pojedynczego człowieka. Tym niemniej przed powstaniem Cyc, nie istniała baza, która zawierałaby wiedzę zdroworozsądkową. Dlatego też wykorzystanie rozległej wiedzy dziedzinowej często ograniczało się do prostego wyszukiwania informacji, co nie pozwalało na spożytkowanie całego potencjału zgromadzonej wiedzy.

Jeden z przykładów jaki pojawia się na stronie projektu, przedstawia sytuację, w której zgromadzona jest duża liczba fotografii wraz z krótkimi opisami. O ile banalne jest wyszukanie fotografii poprzez słowa kluczowe występujące w ich opisach, to znacznie trudniejsze jest wyszukanie ich poprzez wiedzę, która zawarta jest w tych opisach. Jeśli np. zdjęcie opisane jest następująco „Mężczyzna przykładający pistolet do głowy kobiety", to znalezienie go poprzez słowa kluczowe: „mężczyzna", „kobieta", „pistolet" jest oczywiście łatwe. Niemożliwe jest jednak odnalezienie tego zdjęcia poprzez słowo „przemoc", chociaż każdy normalnie myślący człowiek zakwalifikowałby je do tej kategorii, dysponując jedynie jego opisem. Dzieje się tak, gdyż ludzie wykorzystują wiedzę zdroworozsądkową, która mówi, że przyłożenie pistoletu, który jest narzędziem służącym do zabijania, do czyjejś głowy wiąże się zwykle z aktem przemocy[8].

Wraz z rozwojem technologii sieciowych oraz ontologii dziedzinowych pojawiła się również potrzeba powstania ontologii ogólnej, na podstawie której można by konstruować poszczególne ontologie dziedzinowe. W tym wypadku ontologia Cyc okazała się bardzo przydatnym narzędziem, gdyż poza wiedzą zdroworozsądkową zawiera najbardziej ogólne koncepty, niejednokrotnie wykorzystywane w poszczególnych ontologiach szczegółowych. Cyc posiada potencjalnie znacznie więcej zastosowań. Ich przedstawienie wykracza jednak poza ramy tej pracy. Najistotniejsza z naszego punktu widzenia jest możliwość wykorzystania jej w algorytmach przetwarzania języka naturalnego.

[7] Na podstawie: http://www.cyc.com/cyc/company/company
[8] Na podstawie: http://www.cyc.com/cyc/cycrandd/areasofrandd_dir/is

4.3. Dostępne wersje ontologii

Ontologia Cyc nie zdobyłaby z pewnością takiego rozgłosu, gdyby nie fakt, że firma Cycorp postanowiła udostępnić nieodpłatnie jedną z jej wersji, zwaną OpenCyc[9]. Na dzień dzisiejszy istnieją trzy główne wersje tej ontologii:

1. komercyjna
2. rozwojowa
3. otwarta

O wersji *komercyjnej* wiadomo najmniej, gdyż jest ona udostępniana jedynie klientom, którzy gotowi są, za bliżej nieokreśloną kwotę, dokonać jej zakupu. Wiadomo jednak, że stanowi ona podstawę dwóch pozostałych wersji, które rozprowadzane są bezpłatnie.

Wersja *rozwojowa*[10] nie jest dostępna powszechnie i aby ją uzyskać należy wystąpić do Cycorp z prośbą o jej udostępnienie. Jest ona przeznaczona głównie dla instytucji naukowych i centrów rozwojowych. Statystyki umieszczone na głównej stronie projektu mówią o ponad 300.000 konceptów, 26.000 relacji oraz 3.000.000 faktów i reguł opisujących wcześniej wymienione koncepty. Zaiste, jeśli przyjrzeć się wyłącznie liczbom, robią one olbrzymi wrażenie. Najnowsza jej wersja została udostępniona 29 czerwca br. i oznaczona jest numerem 1.0.

Wersja *otwarta* nie jest obłożona żadnymi restrykcjami w zakresie dostępności. Może być w sposób nieograniczony ściągana z internetu. Jej ostatnia wersja, również oznaczona numerem 1.0, która została udostępniona 14 lipca br., zawiera wszelkie koncepty, relacje oraz asercje dostępne w wersji rozwojowej. Nowością w stosunku do poprzednich wersji jest angielski leksykon, dostępny wcześniej wyłącznie w wersji rozwojowej. Wersja *otwarta* nie zawiera jednak m.in. podsystemu przetwarzania języka naturalnego, narzędzia eksportującego wiedzę i części dokumentacji systemu.

W niniejszej pracy wykorzystywana jest wersja *rozwojowa* 0.9, gdyż w trakcie realizacji tej pracy była ona jedyną wersją, która z jednej strony – zawierała angielski leksykon, niezbędny dla pełnej realizacji zamierzonego celu, z drugiej zaś – jako jedyna dała się uruchomić na komputerze autora niniejszej pracy.

4.4. Realizacja modelu teoretycznego

Ontologia Cyc, ze względu na swoją złożoność, wykorzystuje większość mechanizmów opisanych w punkcie 4.1.2. Występują w niej indywidua, kolekcje (czyli pojęcia), relacje oraz funkcje. Zdarzenia reprezentowane są jako pojęcia wykorzystujące specyficzny zestaw relacji. Jak było napisane wcześniej, jest to ontologia ogólna a językiem jej specyfikacji jest `CycL`.

Poniżej przedstawiamy szczegółową charakterystykę najważniejszych elementów wchodzących w skład Cyc. Chcemy jednak zauważyć, że ontologia ta jest na tyle złożona, że nawet częściowe opisanie wszystkich jej istotnych elementów mogłoby wystarczyć na kilka publikacji tego typu, dlatego też ograniczyliśmy się do opisu elementów, które, w uznaniu autora, stanowią zrąb tej ontologii. Niewątpliwie wybór ten jest po części arbitralny, został on jednak dokonany na podstawie wnikliwej analizy dokumentacji systemu oraz doświadczeń zgromadzonych w trakcie pracy z tą ontologią.

[9] http://www.opencyc.org
[10] http://research.cyc.com

Przykładowe hasła opracowane z wykorzystaniem przedstawionych środków opisu znajdują się w dodatku B.

4.4.1. Pojęcia podstawowe

Podstawowymi elementami Cyc są kolekcje **#$Thing**, **#$Collection** i **#$Individual** oraz relacje **#$genls** i **#$isa**.

Najprościej wyjaśnić ich znaczenie porównując je z podstawowymi pojęciami *teorii mnogości z ur-elementami* – odpowiednikiem relacji **#$genls** jest relacja zawierania zbioru A w zbiorze B: $A \subset B$, zaś **#$isa** – relacja przynależności elementu C do zbioru D: $C \in D$.

Kolekcja **#$Thing** jest korzeniem ontologii Cyc, co znaczy, że należą do niej wszystkie elementy występujące w Cyc (wszystkie obiekty występujące w uniwersum). Kolekcja **#$Collection** zawiera wszystkie obiekty, które same są kolekcjami, tzn. mogą zawierać (poprzez relację **#$isa**) inne kolekcje lub indywidua (odpowiada ona zbiorowi wszystkich zbiorów występujących w uniwersum), natomiast kolekcja **#$Individual** zawiera indywidua, czyli obiekty, które nie mogą zawierać innych obiektów (odpowiada ona zbiorowi wszystkich ur-elementów występujących w uniwersum).

Możemy zapisać to formalnie w następujący sposób:

1. $\forall x : x \in$ **#$Thing**
2. **#$Collection** $\subset$ **#$Thing**
3. **#$Individual** $\subset$ **#$Thing**
4. $\forall x : (x \in$ **#$Individual** $\rightarrow \forall y : (\neg y \in x))$
5. $\forall x : (x \in$ **#$Individual** $\leftrightarrow \neg x \in$ **#$Collection**$)$
6. $\forall x \forall y : (x \subset y \rightarrow (x \in$ **#$Collection** $\wedge\ y \in$ **#$Collection**$))$

Istotną obserwacją, którą możemy wysnuć z powyższych określeń jest to, że kolekcja **#$Thing** jest swoim własnym elementem, co czyni ją obiektem problematycznym z punktu widzenia matematycznej teorii mnogości.

4.4.2. Kolekcje i indywidua

Obiekty należące do kolekcji **#$Collection** odpowiadają klasie konceptów, opisanych w punkcie 4.1.2, natomiast te należące do kolekcji **#$Individual** – klasie indywiduów. Inne ważne kolekcje występujące w Cyc to:

1. **#$Intangible**
2. **#$TemporalThing**
3. **#$SpatialThing**
4. **#$PartiallyTangible**
5. **#$Event**
6. **#$ExistingStuffType**
7. **#$ExistingObjectType**

Do kolekcji **#$Intangible** należą obiekty, które nie są fizyczne, w tym sensie, że ich tworzywem nie jest materia fizyczna. Inna nazwa tej kolekcji mogłaby brzmieć „obiekty abstrakcyjne” albo „obiekty niematerialne”. Dowolna kolekcja elementów jest instancją tej kolekcji, niezależnie od tego, czy elementy tej kolekcji są obiektami fizycznymi czy nie. Jest tak dlatego, że kolekcje (pojęcia) są obiektami niematerialnymi.

Kolekcja **#$TemporalThing** jest specjalizacją kolekcji **#$Individual** a należą do niej obiekty, które posiadają jakąś lokalizację w czasie – mogą to być obiekty fizyczne, zdarzenia, interwały i punkty czasowe, etc. Do kolekcji tej nie należą żadne obiekty, które egzystują „poza czasem" – np. obiekty matematyczne.

Kolekcja **#$SpatialThing** również jest specjalizacją kolekcji **#$Individual** a należą do niej obiekty, które są przestrzennie rozciągłe lub posiadają lokalizację relatywną do innych obiektów należących do tej kolekcji w jakiejś przestrzeni (niekoniecznie fizycznej). Do kolekcji tej mogą zatem należeć zarówno obiekty fizyczne (młotki, góry, etc.) jak i obiekty matematyczne (koła, trójkąty, etc.). Kolekcja **#$SpatialThing-Localized**, będąca specjalizacją tej kolekcji, zawiera wyłącznie obiekty, które występują w przestrzeni fizycznej.

Kolekcja **#$PartiallyTangible** (specjalizacja **#$Individual**) zawiera obiekty, które zwykle określamy mianem obiektów fizycznych, tzn. posiadających swoje istnienie w czasie i przestrzeni fizycznej. Obiekty należące do tej kolekcji mogą być w części niefizyczne – np. książka składa się z obiektu fizycznego, który można wziąć do ręki oraz informacji zawartej na jej stronach, która nie jest obiektem fizycznym.

Kolekcja **#$Event** (specjalizacja **#$Individual**) zawiera dynamiczne sytuacje, w których następuje jakaś zmiana stanu świata. Kolekcja ta jest również specjalizacją **#$Intangible**, gdyż poszczególne zdarzenia nie są rozciągłe w przestrzeni. Jedynie obiekty, które biorą udział w zdarzeniach, są często obiektami fizycznymi.

Kolekcja **#$ExistingStuffType** to kolekcja kolekcji, których elementami są obiekty o charakterze substancji, tzn. takie obiekty, których fragment posiada te same (istotne) własności co cały obiekt – np. cały piasek i jego konkretna próbka to ta sama substancja.

Kolekcja **#$ExistingObjectType** to kolekcja kolekcji, których elementami są obiekty o charakterze jednostkowym, tzn. takie obiekty, których fragment nie posiada (istotnych) własności takich samych jak cały obiekt – np. kierownica, będąca elementem samochodu, sama nie służy do transportu ludzi.

Już pobieżne przyjrzenie się przedstawionym kolekcjom wykazuje, że ontologia Cyc jest bardzo skomplikowana. Do ważnych kolekcji należałoby również zaliczyć kolekcję kolekcji drugiego, trzeciego i czwartego rzędu, kolekcję typów kolekcji, etc. Nie sposób tu nawet wymienić wszystkich najbardziej ogólnych kolekcji (czy też konceptów), gdyż z jednej strony – jest ich bardzo dużo, z drugiej zaś – trudno wskazać granicę pomiędzy kolekcjami najbardziej ogólnymi a tymi mniej ogólnymi.

Przedstawione kolekcje charakteryzują się tym, że są bardzo ogólne oraz że większość kolekcji występujących w Cyc jest specjalizacjami tych kolekcji.

4.4.3. Relacje i funkcje

Przy pierwszym przeglądaniu Cyc wydawać by się mogło, że relacjami są wszystkie obiekty, które należą do kolekcji **#$Relation**. Tym niemniej nie jest to prawda, jeśli jako relacje uznamy środki służące do opisu *relacji* zachodzących pomiędzy konceptami i/lub indywiduami. Ponieważ środki takie mają pozwalać na formułowanie w języku wybranym jako formalizm danej ontologii asercji dotyczących tych zależności, muszą one być funkcjami prawdziwościowymi, a nie jedynie relacjami. Tzn. aby stwierdzić, że dana relacja np. *relacja bycia matką*, zachodzi pomiędzy indywiduami x i y, musimy w wybranym formalizmie dysponować predykatem np. **#$mother**, który pozwalałby sformułować asercję:

(**#$mother** x y). Predykat ten musi być funkcją prawdziwościową, która dla argumentów x i y zwraca wartość T (prawda).

Dlatego też w Cyc wyróżniono kolekcję **#$Predicate**, która jest specjalizacją kolekcji **#$TruthFunction**, która z kolei jest specjalizacją kolekcji **#$Relation**. Wynika to z faktu, że w ogólności – relacje nie muszą być funkcjami prawdziwościowymi, a w szczególności – nie wszystkie funkcje prawdziwościowe są predykatami (np. kwantyfikatory, choć są funkcjami prawdziwościowymi, nie są predykatami, dlatego nie służą do wyrażania relacji zachodzących pomiędzy konceptami).

Aby w łatwy sposób odróżnić predykaty od pozostałych środków opisu występujących w Cyc, są one pisane z małej litery, podczas gdy wszystkie pozostałe – z dużej. Ponadto relacje (nie predykaty!), które są funkcjami (ale nie prawdziwościowymi) posiadają postfiks *Fn*. Często predykaty posiadają stowarzyszone z nimi funkcje. Np. predykat **#$mother**, który jest funkcją prawdziwościową i służy do wyrażenia relacji *bycia matką* za pomocą asercji (**#$mother** x y) (x jest matką y), posiada stowarzyszoną funkcję **#$MotherFn**, która dla danego indywiduum zwraca jego matkę: (**#$MotherFn** y) (jeśli wcześniejsza relacja zachodzi oraz poszczególne symbole występujące w tych dwu formułach odnoszą się do tych samych indywiduów, to druga formuła zwraca (reprezentuje, odnosi się do) obiekt x).

Własności predykatów opisywane są również za pomocą predykatów. Do najważniejszych predykatów służących do ich opisu należą **#$arity**, **#$argIsa** oraz **#$argFormat**.

Predykat **#$arity** postaci (**#$arity** RELN N) służy do określania *arności* predykatu, tzn. liczby argumentów, które może on przyjąć, aby sformułować poprawną, w sensie gramatycznym, formułę wykorzystującą dany predykat. RELN to opisywany predykat, zaś N to liczba naturalna będąca jego arnością.

Predykat **#$argIsa** postaci (**#$argIsa** RELN N COL) służy do określania *typu* argumentu, który może pojawić się na pozycji N w formule zawierającej predykat RELN. COL to kolekcja do której musi należeć argument, aby formuła była poprawna.

Predykat **#$argFormat** postaci (**#$argFormat** RELN N FORMAT) służy do określania formatu wybranego argumentu, który może pojawić się na pozycji N w formule zawierającej predykat RELN. FORMAT to format argumentu, który w najprostszym przypadku może wymagać, aby na danej pozycji pojawił się pojedynczy argument. Tym niemniej czasami format może określać, że na danej pozycji mogą pojawić się dwa lub więcej argumentów – bierze się to stąd, że w Cyc występują predykaty o nieokreślonej arności.

Podobnie jak w przypadku kolekcji, tak i w przypadku predykatów przedstawienie pełnej informacji o ich reprezentacji byłoby zbyt obszerne, aby zmieścić je w ramach tej pracy. Spośród najczęściej wykorzystywanych relacji (poza wymienionymi wcześniej **#$isa** oraz **#$genls**) należy wspomnieć również **#$comment**, która wykorzystywana jest do umieszczania komentarzy na temat poszczególnych konceptów, relacji, itp.

4.4.4. Mikroteorie

Cyc jest niezwykle bogatą i złożoną ontologią. Zapanowanie, przez pojedynczą osobę, czy nawet zespół osób, nad tym gąszczem informacji jest zadaniem niewyobrażalnym. Wiadomo jednak, że ze względu na *regułę przepełnienia*, występowanie w niej sprzecznych informacji mogłoby uczynić ją całkowicie bezużyteczną. Z tego też względu wiedza zawarta w Cyc podzielona jest na fragmenty, którymi można łatwiej zarządzać – te fragmenty

to *mikroteorie*. Wiedza zawarta wewnątrz wybranej mikroteorii musi być niesprzeczna, ale nie wymaga się, aby ontologia Cyc była niesprzeczna globalnie.

Inferencje (rozumowania) zawsze prowadzone są wewnątrz pewnej wybranej mikroteorii bądź ich grupy, co zapewnia ich poprawność. Ponadto, mikroteorie tworzą hierarchię – poszczególne asercje należące do ogólnej mikroteorii, dziedziczone są przez jej specjalizacje. Daje to większą kontrolę nad wiedzą zgromadzoną w ontologii.

Ze względu na to, że mikroteorie są pełnoprawnymi obiektami ontologii Cyc, z ich wykorzystaniem można formułować asercje, które opisują ich założenia. Ta własność mikroteorii pozwala spojrzeć na nie jako na reprezentacje *kontekstów*, gdyż konteksty charakteryzują się tym, że narzucają zawsze pewne założenia, które nie muszą być przyjmowane globalnie.

Najbardziej ogólną mikroteorią, która występuje w Cyc jest **#$BaseKB**. Wszelkie asercje, które do niej należą, są zarazem dostępne dla każdej innej mikroteorii. Są to najbardziej ogólne prawdy, wykorzystywane we wszelkich inferencjach prowadzonych w ramach tej ontologii.

4.4.5. Leksykon

Niezwykle istotnym, z naszego punktu wiedzenia, elementem Cyc jest jej leksykon. Zawiera on mapowania występujące pomiędzy poszczególnymi konceptami ontologii Cyc a słowami języka angielskiego. Mapowanie to zrealizowane jest zgodnie z modelem znaczenia, opisanym w punkcie 2.2.

Pomimo tego, że nazwy poszczególnych konceptów w Cyc są słowami lub zlepkami słów języka angielskiego, to fakt ten nie jest w żaden sposób wykorzystywany w leksykonie. Poszczególne słowa języka wiązane są z odpowiednimi konceptami za pomocą predykatu **#$denotation**. Z drugiej strony, słowa te wiązane są z odpowiadającym im łańcuchem znaków za pomocą jednego z predykatów należących do kolekcji **#$SpeachPartPredicate**.

Poszczególne słowa reprezentowane są przez instancje kolekcji **#$LexicalWord**, które będziemy nazywać *konceptami słownymi*. Można je łatwo rozpoznać, gdyż zapisywane są w postaci **#$X-TheWord**, np. **#$Car-TheWord** reprezentuje angielskie słówko *car* (samochód). Słowo to należy również do kolekcji **#$EnglishWord** czyli do kolekcji wszystkich słów angielskich, która jest specjalizacją **#$LexicalWord**.

Poszczególne koncepty słowne mogą reprezentować jedną lub więcej części mowy. Realizuje się to za pomocą predykatu **#$posForms**. Aby zaś powiązać poszczególne koncepty słowne z ich pisownią (ponieważ nazwa konceptu również nie jest wystarczająca) wykorzystywane są predykaty należące do kolekcji **#$SpeachPartPredicate**, wśród których można wymienić:

1. **#$singular** – liczba pojedyncza mianownika
2. **#$plural** – liczba mnoga mianownika
3. **#$infinitive** – bezokolicznik czasownika
4. **#$regularDegree** – stopień równy przymiotnika

Wszystkie te predykaty przyjmują postać (SPP WORD STRING), gdzie SPP to dany predykat, WORD to słowny koncept (np. **#$Car-TheWord**) zaś STRING to łańcuch będący reprezentacją słowa wskazywaną przez dany predykat, np. (**#$singular #$Car-TheWord** "car"). Ponadto każda część mowy określa jakie predykaty mogą zostać wykorzystane do opisu

jej form (np. **#$CountNoun** (rzeczownik policzalny) akceptuje predykaty **#$singular** (mianownik liczby pojedynczej) oraz **#$plural** (mianownik liczby mnogiej)).

Predykat **#$denotation** wiąże wybrany koncept słowny z wybranym elementem Cyc (kolekcją, indywiduum, relacją, etc.). Przyjmuje on następującą postać (**#$denotation** WORD POS N DENOT), gdzie WORD to opisywany koncept słowny; POS to część mowy, jaką jest dane słowo w tej relacji; N to numer sensu w jakim jest używane dane słowo w tej relacji; a DENOT to denotowany element Cyc.

Przykład: „bank" jako rzeczownik posiada w języku angielskim dwa sensy – odpowiadają one polskiemu *bankowi* (organizacji) oraz *brzegowi* rzeki lub jeziora. W Cyc koncept słowny **#$Bank-TheWord** jako rzeczownik posiada dwie denotacje:

– (**#$denotation #$Bank-TheWord #$CountNoun** 0 **#$Bank-Topographical**)
– (**#$denotation #$Bank-TheWord #$CountNoun** 1 **#$BankOrganization**)

z których pierwsza odpowiada brzegowi, a druga organizacji.

Część III

Mapowanie

Rozdział 5

Mapowanie

5.1. Ogólny problem mapowania

5.1.1. Definicja

Po opisaniu Słownika Semantycznego Języka Polskiego oraz ontologii Cyc możemy szczegółowo przedstawić zagadnienia ich mapowania.

Z definicji 3.1 oraz 4.1 wynika, że obiekty te są strukturami abstrakcyjnymi służącymi do opisu ogólnie pojętej rzeczywistości i wykorzystują w tym celu pewne ściśle określone środki wyrazu (w przypadku słownika są to słowa i relacje semantyczne, w przypadku ontologii – koncepty, indywidua, relacje, etc.).

Łatwo można zauważyć pewne podobieństwo pomiędzy tymi strukturami – poszczególne elementy słownika posiadają swoje odpowiedniki w ontologii. Na tym właśnie polega *mapowanie* jednej struktury na drugą – na wskazaniu odpowiadających sobie elementów tych struktur i zachowaniu tej informacji w jednej ze struktur.

Definicja 5.1. ***Mapowanie*** *struktury abstrakcyjnej A na strukturę abstrakcyjną B (ozn.* $A \rightsquigarrow B$*) polega na rozpoznaniu elementów struktury B, które odpowiadają elementom struktury A i zachowaniu informacji o tym w jednej ze struktur.*

Problemem tej definicji jest jednak to, że nie wskazuje *podstawy*, która wykorzystywana jest do rozpoznania odpowiadających sobie elementów. Wskazanie owej podstawy jest czynnikiem koniecznym dla oceny poprawności proponowanego mapowania. Jak wskazaliśmy wcześniej – obie struktury, na swój sposób, odzwierciedlają rzeczywistość – dlatego też wyłącznie ona może być poszukiwaną podstawą.

Odwołując się do modelu znaczenia przedstawionego w punkcie 2.2, możemy sformułować warunki poprawności mapowania poszczególnych symboli[1] występujących w tych strukturach.

Definicja 5.2. ***Warunkiem koniecznym semantycznej poprawności mapowania*** *symbolu x występującego w strukturze abstrakcyjne A, na symbol y występujący w strukturze abstrakcyjne B, jest posiadanie przez symbol x tej samej denotacji (lub ekstensji) co symbol y.*

Innymi słowy – aby można było uznać, że symbole te odpowiadają sobie, obiekt (*resp.* zbiory obiektów), do którego odnosi się symbol występujący w pierwszej ze struktur, musi być identyczny z obiektem (*resp.* zbiorem obiektów), do którego odnosi się symbol występujący w drugiej ze struktur.

Należy zwrócić tutaj uwagę na dwa problemy. Pierwszy dotyczy ogólnej możliwości mapowani różnych struktur na siebie. Aby zachować ściśle warunek 5.2, konieczne jest, aby konceptualizacja wyrażona w obu strukturach była identyczna. Jest to w zasadzie bardzo mało prawdopodobne, w szczególności dla symboli posiadających wieloelementowe ekstensje. Dlatego też warunek ten może zostać osłabiony poprzez zastąpienie wymogu *identyczności* ekstensji, wymogiem *istotnego nachodzenia na siebie* ekstensji. Oczywiście nie da się tutaj ściśle sformułować na czym owa istotność miałaby polegać – stąd też biorą się podstawowe problemy w mapowaniu[2].

Drugi problem wynika z faktu, że znaczenie poszczególnych symboli nie zależy wyłącznie od ich ekstensji, ale konstytuowane jest przez relacje w jakie wchodzą z pozostałymi symbolami. Dlatego przedstawiony wcześniej warunek konieczny musi zostać uzupełniony warunkiem wystarczającym:

Definicja 5.3. ***Warunkiem wystarczającym semantycznej poprawności mapowania*** *symbolu x występującego w strukturze A na symbol y występujący w strukturze B, jest zachodzenie tych samych relacji pomiędzy symbolem x, a symbolami struktury A, które posiadają swoje odpowiedniki w strukturze B oraz pomiędzy symbolem y a symbolami występującymi w strukturze B, będącymi odpowiednikami symboli ze struktury A.*
Bardziej formalnie – jeśli warunek:
$\forall z_1, z_2 : \Big(\big(R(x, z_1) \wedge z_1 \rightsquigarrow z_2\big) \rightarrow R(y, z_2)\Big)$
gdzie:

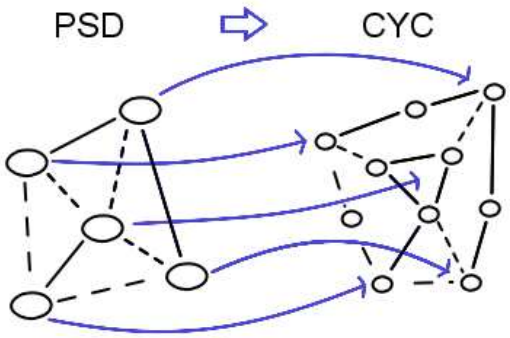
- $x \rightsquigarrow y$ *oznacza: symbol y odpowiada symbolowi x (symbol x jest zmapowany na symbol y).*
- x, z_1 *należą do struktury A*
- y, z_2 *należą do struktury B*

zachodzi dla każdej relacji R występującej zarówno w strukturze A jak i w strukturze B, to mapowanie $x \rightsquigarrow y$ jest poprawne.

Również z tym warunkiem wiążą się dalsze problemy – podstawowy dotyczy relacji R. Relacja ta musi być identyczna w obu strukturach, aby definicja ta mogła zostać zastosowana. Ponieważ mapowane struktury zwykle konstruowane są niezależnie, to nie można

[1] Symbole to takie elementy badanych struktur, które mogą być opisane za pomocą modelu znaczenia przedstawionego w punkcie 2.2.

[2] Istotność ta nie może być wyrażona np. w postaci liczbowej (90% pokrycie ekstensji), gdyż zwykle nie dysponuje się dostatecznie licznymi ekstensjami (o ile w ogóle są one dostępne).



Rysunek 5.1. Ogólny schemat mapowania Słownika Semantycznego na ontologię Cyc.

zagwarantować, że relacje takie będą faktycznie występować. Tym niemniej w większości systemów ontologicznych oraz słowników semantycznych może wskazać co najmniej kilka relacji, które mogą zostać utożsamione.

Warunek ten wskazuje również, że pierwszym, strategicznym krokiem, od którego należy rozpocząć mapowanie, jest rozpoznanie w mapowanych strukturach odpowiadających sobie relacji.

Inny problem dotyczy procesu mapowania – przedstawiony warunek można sprawdzić dopiero po zmapowaniu wszystkich symboli. Natomiast mapowanie (szczególnie ręczne oraz półautomatyczne) zwykle dokonywane jest iteracyjnie – tzn. w jednym kroku mapuje się tylko jeden symbol i dla niego sprawdza odpowiednie warunki, bazując na wcześniej zmapowanych symbolach. Może się jednak zdarzyć, że symbole, które zostaną zmapowane później, spowodują naruszenie tego warunku (tzn. proces mapowania nie jest monotoniczny ze względu na ten warunek). Problem ten podobny jest do problemu nauki języka obcego jedynie poprzez wskazywanie przedmiotów i formułowanie zdań z wykorzystanie już poznanych słów. Właściwa identyfikacja pierwotnego zestawu słów (symboli) konieczna jest aby proces ten zakończył się sukcesem.

Problem ten można rozwiązać poprzez ręczne zmapowanie najbardziej ogólnych symboli (tzn. symboli posiadających największe ekstensje) i bazowaniu na tej informacji w dalszym procesie mapowania. Najbardziej ogólne symbole wchodzą bowiem w relacje z największą liczbą innych symboli. Dodatkowo można zastosować strategię *top-down*, w której, wychodząc od najbardziej ogólnych symboli, mapuje się je zgodnie z rosnącym stopniem ich specyficzności.

Ogólny schemat mapowania Słownika Semantycznego na ontologię Cyc przedstawiony jest na rys. 5.1. Większe owale reprezentują słowa występujące w słowniku, mniejsze – kolekcje i indywidua ontologii Cyc; linie proste – różne rodzaje relacji; natomiast łuki zakończone strzałkami wskazują odpowiadające sobie elementy.

5.1.2. Różnice

Poniżej przedstawiamy, konieczne do uwzględnienia w procesie mapowania, różnice jakie mogą wystąpić pomiędzy ontologiami oraz pomiędzy ontologiami i słownikami semantycznymi.

Różnice występujące pomiędzy ontologiami

Zagadnienie mapowania formułowane jest głównie dla ontologii (np. [3, 6, 34]) – tzn. w definicji 5.1 określenie *abstrakcyjna struktura* odnosi się zwykle do ontologii. Dlatego też w literaturze przedmiotu można znaleźć najwięcej informacji na temat różnic występujących pomiędzy ontologiami.

W [5, 34] wymienia się następujące różnice:

1. Składnia języka reprezentacji wiedzy.
2. Ekspresywność języka reprezentacji wiedzy (np. obecność negacji, kwantyfikatorów, wartości domyślnych, operatorów modalnych, zbiorów, etc.).
3. Fundamentalne różnice reprezentacyjne (np. logika prawdziwościowa *vs.* probabilistyka bayesowska).
4. Konwencje modelowania (np. modelowanie bardziej specyficznego konceptu za pomocą nowego konceptu lub „atrybutu-typu").
5. Zasady reprezentacji „trudnych" pojęć (np. reprezentacja czasu za pomocą interwałów lub punktów czasowych).
6. Odmienne nazewnictwo dla elementów posiadających tę samą semantykę.
7. Odmienna semantyka dla elementów posiadających tę samą nazwę.
8. Reprezentacja własności za pomocą atrybutów *vs.* funkcji *vs.* predykatów.
9. Proste różnice strukturalne (np. jedna ontologia jest reorganizacją drugiej ontologii).
10. Złożone różnice strukturalne (np. predykaty akcji *vs.* zreifikowane zdarzenia).

Różnice występujące pomiędzy ontologiami a słownikami semantycznymi

Różnice, które występują pomiędzy poszczególnymi ontologiami, są jednak zwykle mniejsze niż pomiędzy ontologiami a słownikami semantycznymi. Do najistotniejszych różnic występujących pomiędzy tymi strukturami należą:

1. Przyjęty punkt widzenia.
2. Opisywana dziedzina wiedzy.
3. Środki opisu.
4. Środki identyfikacji.
5. Gęstość pojęciowa.

Punkt wiedzenia ontologii jest zazwyczaj naukowy, tzn. wiedza zawarta w ontologiach odzwierciedla, a przynajmniej powinna odzwierciedlać, mniej lub bardziej aktualny stan wiedzy naukowej na temat wybranej dziedziny wiedzy. Tymczasem w słownikach semantycznych dominuje językowy, czy też zdroworozsądkowy punkt widzenia. Aby uznać, że jakaś relacja zachodzi pomiędzy wybranymi słowami, czy znaczeniami, wystarczy świadectwo użytkowników języka lub świadectwo tekstu w danym języku. Częstokroć takie świadectwo byłoby nieakceptowalne z naukowego punktu wiedzenia.

Dziedzina wiedzy: tylko niektóre ontologie nadają się do mapowania na nie słowników semantycznych, mianowicie – ontologie ogólne. Ontologie dziedzinowe zawierają wiedzę specjalistyczną, natomiast słowniki semantyczne zawierają słownictwo wykorzystywane

w dyskursie niefachowym. Gdyby użyć ontologii dziedzinowej jako ontologii, na którą mapowany jest słownik semantyczny, to większość słów występujących w słowniku nie znalazłaby swoich odpowiedników.

Środki opisu to pojęcia jakie wykorzystywane są do opisu konceptualizacji w ontologii oraz relacje i ich elementy, które pojawiają się w słownikach semantycznych. Zidentyfikowanie odpowiadających sobie klas środków opisu nie jest jednak problematyczne, aczkolwiek należy zwrócić uwagę na pewne niebezpieczeństwa. Poszczególne odpowiadające sobie klasy przedstawione są w tabeli 5.1.

Ontologia	**Słownik semantyczny**
Koncepty	Rzeczowniki pospolite, przymiotniki
Indywidua	Nazwy własne
Relacje	Relacje semantyczne
Zdarzenia	Czasowniki, rzeczowniki pospolite

Tablica 5.1. Klasy środków opisu w ontologiach i słownikach semantycznych

Najbardziej problematyczne jest utożsamienie relacji występujących w ontologiach z relacjami semantycznymi występującymi w słownikach semantycznych. Co prawda niektóre relacje występujące w słownikach, jak np. *meronimia*, posiadają swoje odpowiedniki w ontologiach – gdyż relacja ta może występować np. pomiędzy indywiduami wybranej ontologii. Tym niemniej relacja *synonimii* zazwyczaj nie występuje w ontologiach, gdyż z założenia, w systemach tych identyfikacja poszczególnych klas obiektów powinna być jednoznaczna, dlatego też nie powinny występować np. dwa koncepty, które byłyby synonimiczne. Dlatego, z wyjątkiem sytuacji, w której chce się wskazać synonimiczne pojęcia występujące w innej ontologii, relacja ta nie powinna pojawiać się w ontologiach.

Środki identyfikacji to środki służące do identyfikacji poszczególnych elementów danej struktury abstrakcyjnej. Zarówno w ontologiach jak i w słownikach semantycznych wykorzystuje się w pewien sposób identyfikację słowną – tzn. nazwa danego elementu wskazuje do czego on się odnosi. Tym niemniej, z powodu zjawiska *homonimii* konieczne jest rozróżnianie poszczególnych konceptów, które odnoszą się do poszczególnych homonimów. W ontologiach często wykorzystuje się złożone nazwy, które wskazują z którym z homonimów mamy do czynienia (np. zamek-urządzenia, zamek-budynek, zamek-pistoletu, etc.) natomiast w słownikach semantycznych nie można skorzystać z tego rozwiązania. Poszczególne homonimy są wyróżnione, ale każdy z nich posiada tę samą etykietę[3]. Problem ten może być rozwiązany w przypadku, gdy ontologia posiada leksykon dla wybranego języka naturalnego.

Gęstość pojęciowa to liczba pojęć wykorzystywanych do opisu jakiegoś fragmentu rzeczywistości – może być ona jedynie porównywana, gdyż trudno wyodrębnić jakąś „jednostkę rzeczywistości". Powiemy, że dana struktura abstrakcyjna posiada większą gęstość pojęciową od innej, jeśli do opisu pewnego obszaru wiedzy wykorzystuje więcej pojęć, niż inna struktura. Zazwyczaj ontologie posiadają większą gęstość pojęciową niż słowniki seman-

[3] Czasami różnice pomiędzy homonimami przekładają się na pewne różnice gramatyczne, ale nie jest to zjawisko częste i nie mogłoby w pełni wystarczyć do identyfikacji poszczególnych homonimów.

tyczne. Dzieje się tak dlatego, że można do nich wprowadzać koncepty, które nie występują w języku naturalnym, a które mogą znacznie poprawić skuteczność prowadzonych w nich inferencji. Jako przykład może posłużyć koncept **#$PartiallyTangible** z ontologii CYC – trudno znaleźć w języku słowo, które odpowiadałoby rzeczy „częściowo dotykalnej". Tym niemniej pojęcie to jest przydatne ze względu na pewne generalizacje, których można dzięki niemu dokonać.

5.2. Mapowanie Słownika Semantycznego Języka Polskiego na ontologię Cyc

Po przedstawieniu ogólnego sposobu mapowania słowników semantycznych na ontologie, przystępujemy do szczegółowego przedstawienia mapowania Słownika Semantycznego Języka Polskiego na ontologię Cyc.

Najpierw szczegółowo omawiamy różnice występujące pomiędzy tymi strukturami, które muszą zostać uwzględnione w procesie mapowania, następnie zaś omawiamy środki wykorzystywane do mapowania oraz jego poszczególne sposoby.

5.2.1. Różnice

Słownik Semantyczny[4] i ontologia Cyc różnią się pod wieloma względami, które w istotny sposób wpływają na ich mapowanie. Niektóre różnice są na tyle duże, że uniemożliwiają dokonanie pełnego mapowania – zostały ona opisane w rozdziale 6. Teraz zaś skupimy się na tych różnicach wymienionych w punkcie 5.1.2, które udało się przezwyciężyć. Warto również zwrócić uwagę na fakt, że niektóre problemy wymienione w punkcie 5.1.2 w ogóle go nie dotyczą (np. problem reprezentacji wartości logicznych).

Język reprezentacji wiedzy

Język reprezentacji wiedzy Słownika Semantycznego jest znacznie uboższy niż ten wykorzystywany w Cyc. Poszczególne relacje semantyczne są co najwyżej trójargumentowe (relacje zawierające człon **#$RELATED_TO**). Co prawda w Słowniku przewidziane jest wykorzystanie *skryptów zdarzeń*, ale w obecnej chwili rozwiązanie to nie zostało jeszcze zastosowane, dlatego też nie będzie brane pod uwagę.

Poszczególne relacje dają się łatwo przetłumaczyć (w sensie syntaktycznym) na odpowiednie predykaty dwu i trójargumentowe.

Konwencje modelowania

Zarówno w przypadku Słownika Semantycznego jak i ontologii Cyc koncepty bardziej specyficzne modelowane są poprzez wprowadzenie nowego symbolu. W przypadku Słownika wynika to z faktu, że rejestruje on to co występuje w języku, w którym bardziej specyficzne koncepty często formułowane są w postaci wyrażeń złożonych. W ontologii Cyc koncepty bardziej specyficzne zawsze wprowadzone są jako niezależne specjalizacje konceptów bardziej ogólnych.

[4] Określenie to odnosi się do Słownika Semantycznego Języka Polskiego i będzie w ten sposób stosowane w dalszej części pracy, gdyż nie powinno rodzić niejednoznaczności

Odmienne nazewnictwo

Problem odmiennego nazewnictwa posiada dwa aspekty. Po pierwsze: poszczególne słowa opisywane w Słowniku Semantycznym należą do języka polskiego, podczas gdy Cyc posiada leksykon w języku angielskim. Jest to podastawowy problem, który stanowi sedno tej pracy i zostanie omówiony później (patrz punkt 5.2.3).

Po drugie zaś: odmienne słownictwo stosowane jest również do opisu poszczególnych relacji, co może prowadzić do pomyłek. W obu strukturach występuje relacja *isa*. Tym niemniej jej semantyka jest zupełnie inna: w Słowniku Semantycznym wiąże ona koncept ogólny z jego specjalizacjami, natomiast w ontologii Cyc wiąże instancję z kolekcją, do której ona należy (konkretny byt z pojęciem pod które on podpada). Dlatego też, na co zwracano uwagę już wcześniej, poprawne zmapowanie relacji jest kluczowe dla poprawnego mapowania i nie może zostać przeprowadzone w sposób automatyczny. Na szczęście poszczególne predykaty występujące w Cyc, poza słownym opisem, posiadają formalny zestaw reguł ich stosowania, co pozwala uniknąć zasadniczych błędów.

Przyjęty punkt widzenia

Zarówno w ontologii Cyc jak i w Słowniku Semantycznym deklarowany jest zdroworozsądkowy punkt widzenia. Tym niemniej postulat tej jest realizowany w pełni wyłącznie w Słowniku Semantycznym, gdyż poza zdroworozsądkowymi faktami, Cyc zawiera mnóstwo wiedzy specjalistycznej (np. z zakresu logiki, matematyki, filozofii, etc.), bez której niemożliwe byłoby prowadzenie ścisłych rozumowań.

Różnica ta nie stanowi jednak problemu ze względu na kierunek mapowania – zdroworozsądkowe fakty występujące w Słowniku Semantycznym mogą zostać przeniesione do Cyc w ramach pewnej mikroteorii, która nie będzie nakładała takich rygorów, jak np. mikroteoria *prawdy matematycznej*.

Środki identyfikacji

Poszczególne homonimy występujące w Słowniku Semantycznym mogą zostać odróżnione przez człowieka dzięki krótkiej informacji, która stowarzyszona jest z każdym hasłem słownikowym. Tym niemniej informacja ta byłaby niezwykle trudna do wykorzystania dla algorytmów automatycznego mapowania ontologii (konieczność parsowania zdań w języku polskim, etc.). Dlatego też poszczególne homominy mogą zostać odróżnione jedynie poprzez relacje w jakie wchodzą z innymi słowami. Jest to szczególnie ciekawy aspekt mapowania Słownika Semantycznego na ontologię Cyc, gdyż w pewnym stopniu przypomina problem wyboru właściwego znaczenia słowa w danym kontekście. Problem ten zostanie również omówiony bardziej szczegółowo w dalszej części tekstu (patrz p. 5.2.3).

Gęstość pojęciowa

Różnice w gęstości pojęciowej pomiędzy Słownikiem Semantycznym a ontologią Cyc są znaczne. W szczególności w obszarach pojęć ogólnych daje się zauważyć różnice pomiędzy wysoce skomplikowaną strukturą pojęciową Cyc, a relatywnie prostą strukturą słownika.

O ile różnice te nie stanowią problemu dla mapowania samego w sobie (mapowana jest struktura mniej skomplikowana na bardziej skomplikowaną, poza tym niektóre relacje są przechodnie), to niewątpliwie stanowi one wyzwanie dla konstruktorów algorytmów automatycznego mapowania, gdyż dwa słowa, które w Słowniku Semantycznym połączone są bezpośrednią relacją, mogą być w ontologii Cyc rozdzielone skomplikowanym

łańcuchem elementów pośredniczących. Rozpoznanie tych elementów i decyzja, że dane pojęcie bliższe jest semantycznie temu a nie innemu pojęciu, jest jednym z najtrudniejszych problemów, które należy rozwiązać, jeśli chce się osiągnąć pełną automatyzację mapowania.

5.2.2. Środki mapowania

Poniżej przedstawimy środki mapowania, czyli narzędzia i mechanizmy, które wykorzystane zostały w procesie mapowania. Z jednej strony dotyczą one sposobu przechowywania informacji o mapowaniu, z drugiej zaś sposobu identyfikacji poszczególnych, odpowiadających sobie elementów.

Przechowywanie informacji o mapowaniu

Mapowanie Słownika Semantycznego Języka Polskiego na ontologię Cyc, zgodnie z definicją 5.1 polega na znalezieniu w tej ontologii odpowiedników dla wszystkich elementów występujących w tym słowniku: relacji semantycznych, kategorii semantycznych oraz poszczególnych słów. Informacja o tym mapowaniu musi zostać umieszczona w jednej ze struktur. Jako miejsce przechowywania informacji o mapowaniu wybrana została ontologia Cyc, gdyż zawiera ona odpowiednią infrastrukturę pojęciową, stworzoną z myślą o mapowaniu na nią innych źródeł wiedzy.

Infrastruktura mapowania została opisana w [34]. Podstawowym jej elementem jest relacja **#$synonymousExternalConcept**, która przyjmuje trzy argumenty: TERM SOURCE i STRING. TERM to element występujący w Cyc, który jest odpowiednikiem mapowanego elementu. SOURCE to zewnętrzne źródło wiedzy, które mapowane jest na Cyc, natomiast STRING to identyfikator mapowanego elementu w tym źródle wiedzy. Asercja zawierająca tę relację służy do wyrażania faktu, że odpowiednie elementy w niej występujące, posiadają identyczną semantykę, tzn. są synonimiczne. Relacja **#$overlappingExternalCon**
przyjmuje te same argumenty co relacja **#$synonymousExternalConcept**. Elementy mapowane za pomocą tej relacji nie są jednak synonimiczne, ale ich znaczenia zachodzą na siebie w istotny sposób.

Aby mapowanie przebiegało zgodnie z zasadami przyjętymi w ontologii Cyc, konieczne było stworzenie odpowiedniej mikroteorii, w której można by przechowywać jego rezultaty. Mikroteoria ta nazywa się **#$PSDMappingMt**, została stworzona na wzór mikroteorii **#$WordNetMappingMt** i należy do kolekcji **#$MappingMicrotheory**. Ponadto konieczne było wprowadzenie do ontologii Cyc obiektu reprezentującego Słownik Semantyczny Języka Polskiego – występuje on jako **#$PolishSemanticDictionary**. Jako identyfikatory poszczególnych słów występujących w słowniku zostały użyte ich identyfikatory bazodanowe.

Relacje **#$synonymousExternalConcept** i **#$overlappingExternalConcept** nie wyczerpują jednak możliwości mapowania Słownika na ontologię Cyc. Aby uzyskać polski leksykon dla tej ontologii, konieczne jest wykorzystanie środków opisanych w punkcie 4.4.5: **#$denotation** oraz **#$LexicalWord**. Również w tym wypadku konieczne było wprowadzenie pewnych obiektów do ontologii Cyc:

1. **#$PolishLexicalMicrotheory** – kolekcja wszystkich mikroteorii leksykalnych, których językiem jest język polski.

2. **#$PolishLexiconMt** – mikroteoria, w której można definiować predykaty leksykalne specyficzne dla języka polskiego.
3. **#$GeneralPolishMt** – mikroteoria, w której występuje mapowanie pomiędzy poszczególnymi słowami języka polskiego a elementami występującymi w Cyc. (**#$PSDMapping-Mt** jest specjalizacją tej mikroteorii.)
4. **#$PolishWord** – kolekcja wszystkich słów języka polskiego. Specjalizacja kolekcji **#$LexicalWord**.

Słownik angielsko-polski

Identyfikacja w ontologii Cyc elementów odpowiadających poszczególnym elementom Słownika Semantycznego z racji różnicy języków wykorzystywanych w obu strukturach (angielskiego i polskiego), nie mogła by zostać dokonana bez wykorzystania słownika polsko-angielskiego. W przypadku mapowania ręcznego mógłby zostać wykorzystany dowolny, również papierowy, słownik polsko-angielski. Tym niemniej, chcąc umożliwić mapowanie półautomatyczne oraz w pełni automatyczne, konieczne było wykorzstanie słownika elektronicznego. Niestety, w obecnej chwili na polskim rynku nie udało się znaleźć słownika, który dysponowałby odpowiednimi mechanizmami pozwalającymi na bezpośrednie wykorzystanie go w zewnętrznej aplikacji. Wszelkie słowniki tego typu tworzone są wyłącznie z myślą o ludziach, a nie innych aplikacjach, które mogłyby z nich korzystać.

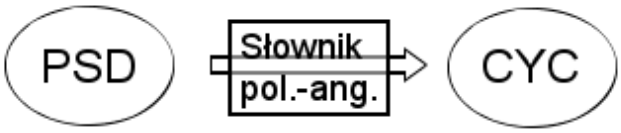
Dlatego też konieczne było wybranie jednego ze słowników elektronicznych tworzonych z myślą o ludziach i zaadaptowanie go na potrzeby naszego problemu. Szczegóły techniczne tego zagadnienia przestawione zostały w części implementacyjnej niniejszej pracy. Teraz zaś przedstawimy pokrótce jedynie najogólniejsze cechy *Wielkiego Multimedialnego Słownika polskiego-angielsko Oxford/PWN*, który został wybrany do realizacji tego zadania.

Z informacji dostępnej na stronie internetowej słownika [32] wynika, że zawiera on ponad 500 tys. jednostek leksykalnych polsko-angielskich oraz 120 tys. pozycji na liście wyszukiwania obejmujących słownictwo ogólne, slangowe i specjalistyczne. Przy każdym haśle dostępna jest informacja gramatyczna oraz większość haseł posiada kwalifikatory dziedziny, stylu oraz zakresu. Dzięki tym cechom może on zostać wykorzystany w procesie mapowania, gdyż toretycznie powinien on pokrywać dziedzinę Słownika Semantycznego (słownictwo ogólne).

Poszczególne hasła słownikowe zawierają zarówno bezpośrednie tłumaczenia danego słowa, jak i tłumaczenia zwrotów, w których może on występować. Z naszego punktu widzenia najbardziej istotne jest jednak wyodrębnienie poszczególnych homonimów danego słowa, gdyż tylko ono pozwala dokonać właściwego mapowania. Właśnie w tym obszarze pojawia się kilka problemów, które w mniejszym lub większym stopniu utrudniają wykorzystanie tego słownika.

Po pierwsze – poszczególne homonimy występują najczęściej jako odrębne pozycje w ramach jednego hasła, gdzie każdy posiada dodatkową informację pozwalającą na jego odróżnienie od pozostałych homonimów (wyrażone za pomocą relacji paradygmatycznych bądź syntagmatycznych – por. zamek(1) – do zamykania (rel. syntagmatyczna); zamek(2) – w broni palnej (rel. syntagmatyczna); zamek(3) – budowla (rel. paradygmatyczna)), jednakże czasami są one wyodrębnione na poziomie leksykalnym (tzn. występują jako odrębne hasła) i wtedy ta dodatkowa informacja nie zawsze się pojawia (por. zamęczyć(1) – doprowadzić do wyczerpania; zamęczyć(2) – ?).

Po drugie – wewnętrzna struktura haseł jest uzależniona od sposobu ich prezentacji,



Rysunek 5.2. Ogólny schemat mapowania z wykorzystaniem słownika polsko-angielskiego.

tzn. ich struktura logiczna nie została oddzielona od informacji o graficznym formatowaniu hasła. O ile nie stanowi to problemu dla ludzi, którzy oglądają hasło w postaci graficznej, to stanowi poważny problem przy konstrukcji algorytmów parsingu haseł słownikowych.

Po trzecie – występuje wiele różnych struktur opisu poszczególnych haseł, co sprawia, że trudno jest stworzyć ogólny mechanizm ich parsingu, a przez to trudno uzyskać gwarancję, że działa on poprawnie dla wszystkich haseł, a nie wyłącznie dla tego zbioru, który posłużył do jego konstrukcji (niemożliwe jest bowiem ręczne sprawdzenie, czy mechanizm parsingu działa poprawnie dla wszystkich haseł, ze względu na ich liczbę).

Po czwarte – informacja pozwalająca odróżnić poszczególne homonimy tworzona była wyłącznie z myślą o ludziach, dlatego też wydaje się, że nie zastosowano żadnych ogólnych zasad jej tworzenia: czasem jest to słowo o bardziej ogólnym znaczeniu (por. zamek – budowla), czasem jakaś dowolnie wybrana relacja syntagmatyczna (por. zamek – w broni palnej (rel. meronimii) lub paradygmatyczna (por. zamęczyć – doprowadzić do wyczerpania (rel. synonimii)). Dodatkowy problem stanowi fakt, że informacja ta czasem jest pojedynczym słowem, czasem zaś złożonym wyrażeniem.

Niektóre z przedstawionych problemów mogą zostać rozwiązane poprzez skonstruowanie wysoce skomplikowanych algorytmów parsingu, inne zaś dla całkowitego rozwiązania wymagałyby ingerencji w zawartość słownika.

Ogólny schemat mapowania Słownika Semantycznego na ontologię Cyc z wykorzystaniem słownika polsko-angielskiego przedstawiony jest na rysunku 5.2

5.2.3. Sposoby mapowania

Mapowanie Słownika Semantycznego na ontologię Cyc może odbywać się na trzy sposoby (por p. 1.1), z których każdy kolejny wiąże się z większym stopniem automatyzacji. W ramach opisywanego tutaj projektu udało się zrealizować mapowanie ręczne oraz półautomatyczne. Poniżej szczegółowo omawiamy poszczególne sposoby mapowania, wraz z uzasadnieniem ich stosowania.

Mapowanie ręczne

Ręczne mapowania Słownika Semantycznego na ontologię Cyc nie wymaga konstrukcji żadnych algorytmów automatycznej identyfikacji odpowiadających sobie elementów. Zajmujemy się nim jednak z dwóch powodów. Po pierwsze – interfejs użytkownika oraz mechanizmy przechowywania i pozyskiwania informacji wykorzystywane w mapowaniu ręcznym, z powodzeniem można wykorzystać w mapowaniu półautomatycznym, które wystarczy uzupełnić o odpowiedni algorytm selekcji kandydatów do mapowania. Po drugie – niektóre elementy, takie jak np. relacje i kategorie semantyczne występujące w Słowniku Semantycznym muszą zostać zmapowane ręcznie, gdyż z jednej strony nie można w zasadzie

stworzyć żadnego sensownego mechanizmu ich mapowania, z drugiej zaś poprawność ich mapowania jest kluczowa dla konstrukcji odpowiednich mechanizmów mapowania półautomatycznego i automatycznego.

Proces mapowania ręcznego jest następujący:

1. Ekspert wybiera do zmapowania jedno ze słów występujących w Słowniku Semantycznym.
2. Ekspert przeszukuje ontologię Cyc w celu znalezienia odpowiednika mapowanego słowa.
3. Po znalezieniu odpowiednika, ekspert żąda aby informacja o mapowaniu znalazła się w ontologii Cyc.

W wyniku żądania eksperta następujące akcje wykonywane są w ontologii Cyc:

1. Dodawane jest asercja wykorzystująca predykat **#$synonymousExternalConcept**, stwierdzająca, że w zewnętrznym źródle wiedzy (Słowniku Semantycznym) występuje koncept synonimiczny z konceptem wskazanym przez eksperta.
2. Sprawdzana jest informacja, czy występuje już koncept słowy **#$X-TheWord**, gdzie X, to zapisane z dużej litery polskie słowo, które jest mapowane[5]. Jeśli koncept ten nie istnieje, to jest tworzony.
3. Sprawdzana jest informacja czy koncept słowny, o którym mowa w poprzednim punkcie, posiada reprezentację dla typu gramatycznego zgodnego z mapowanym słowem (poprzez jeden z predykatów należących do kolekcji **#$SpeachPartPredicate**). Jeśli nie, to dodawana jest odpowiednia asercja.
4. Dodawana jest asercja wykorzystująca predykat **#$dentotation**, która wiąże koncept słowny, o którym mowa we wcześniejszych punktach, z konceptem wskazanym przez eksperta.

Efekty mapowania ręcznego wymienionych relacji i kategorii zostały przedstawione w punkcie 5.2.4.

Mapowanie półautomatyczne

Do realizacji mapowania półautomatycznego konieczne jest wykorzystanie słownika polsko-angielskiego oraz występujących w Cyc mechanizmów do identyfikacji konceptów odpowiadających poszczególnym słowom języka angielskiego.

Proces mapowanie półautomatycznego jest następujący:

1. Ekspert wybiera do zmapowania jedno ze słów występujących w Słowniku Semantycznym.
2. W słowniku polsko-angielskim poszukiwane są wszystkie jednowyrazowe tłumaczenia danego słowa na język angielski.
3. Dla każdego znalezionego słowa, będącego tłumaczeniem danego słowa języka polskiego na języka angielski, pobierane są z ontologii Cyc koncepty, które są związane z tym słowem poprzez predykat **#$dentotation**. Służy do tego komenda `denotation-mapper` języka `SubL`[6].
4. Zbiory konceptów uzyskanych dla poszczególnych angielskich słów sumuje się oraz sortuje w porządku alfabetycznym i przedstawia ekspertowi.

[5] Cyc korzysta z kodowania ISO-8859-1, dlatego konieczne jest zastąpienie w tym słowie polskich znaków diakrytycznych. Zagadnienie to omówione jest w części implementacyjnej

[6] Język `SubL` to wewnętrzny języka ontologii Cyc, który w przeciwieństwie do języka `CycL` służącego do manipulacji wiedzą, pozwala na bezpośrednią interakcję z silnikiem inferencyjnym.

5. Ekspert wybiera jeden lub więcej znalezionych konceptów jako odpowiedniki wybranego słowa polskiego i żąda aby informacja o mapowaniu znalazła się w ontologii Cyc (proces ten przebiega identycznie jak w przypadku mapowania ręcznego).

Przedstawiony proces może zostać usprawniony poprzez ograniczenie liczby kandydatów do mapowania, które przegląda ekspert. Ograniczenie to może zostać dokonane poprzez wprowadzenie pewnych reguł, wykluczających niepoprawne mapowanie. Reguły te mogą być sformułowane zarówno wobec wyboru słów języka angielskiego będących odpowiednikami słowa polskiego, jak i wobec konceptów występujących w ontologii Cyc, które są denotacjami odpowiednich słów angielskich. Podstawowa reguła, która może zostać zastosowana w obu wypadkach bazuje na typie gramatycznym mapowanego słowa[7]. Wykluczane są słowa i denotacje, których typ gramatyczny nie jest zgodny z typem gramatycznym mapowanego słowa.

Inne reguły mogłyby korzystać z pozostałych informacji zawartych w Słowniku Semantycznym (w szczególności z informacji o relacjach semantycznych, w jakie wchodzi dane słowo), tym niemniej będą one opisane w następnym punkcie, gdyż – z jednej strony – nie zostały one zaimplementowane, z drugiej zaś – trudno sformułować reguły, które działałyby w 100% przypadków, a tylko reguły pewne powinny być wykorzystywane w mapowaniu półautomatycznym.

Mapowanie półautomatyczne jest potrzebne ze względu na to, że stworzenie 100% poprawnych algorytmów mapowania automatycznego wydaje się bardzo odległe. Poza tym, dzięki interakcyjnej pracy, którą ono umożliwia, można wyłapać rozmaite błędy, które mogłyby zostać przeoczone, gdyby przystąpiło się natychmiast do konstrukcji algorytmów w pełni automatycznych. Pozwala ono bowiem obserwować reakcje systemu na różne dane i w iteracyjnym procesie doskonalić stosowane algorytmy.

Mapowanie automatyczne

Algorytm mapowania automatycznego jest naturalnym rozwinięciem algorytmów mapowania półautomatycznego, uzupełnionym o odpowiednie metody heurystyczne, który wykorzystując dostępną informację gramatyczną i semantyczną dąży do jednoznacznego wyboru poprawnego mapowania.

Problemy jakie pojawiają się w implementacji takiego algorytmu biorą się przede wszystkim z występowania homonimów zarówno w języku polskim jak i w języku angielskim. Kiedy w pierwszym etapie pobieramy ze słownika polsko-angielskiego tłumaczenia danego słowa, spotykamy się z pierwszym problemem – dane słowo posiada w języku polskim zwykle kilka homonimów, spośród których należy wybrać ten właściwy. Oczywiste jest w tym wypadku zastosowanie ograniczenia na typ gramatyczny do jakiego należy dane słowo, ale nie jest ono wystarczające (np. wszystkie homonimy słowa „zamek" są rzeczownikami, ale każdemu z nich odpowiada inne słowo w języku angielskim: zamek (budynek) – castle, zamek (w broni palnej, w drzwiach) – lock, zamek (błyskawiczny) – zipper).

Rozpoznanie homonimów odpowiadających sobie w Słowniku Semantycznym i słowniku polsko-agielskim jest konieczne dla właściwego przeprowadzenia procesu mapowania. W idealnej sytuacji w obu słownikach każde słowo posiada tyle samo rozpoznanych homonimów. Wtedy proces ten ulega uproszczeniu, gdyż zamiast niezależnego wyboru jed-

[7] Słownik Semantyczny oryginalnie nie posiada odpowiedniej informacji gramatycznej, ale może ona zostać pobrana z biblioteki form gramatycznych – `CLP`. Zagadnienie to opisane jest w części implementacyjnej.

nego z homonimów można dokonać dopasowania grupowego, tzn. dopasowywać wszystkie homonimy na raz, co powinno przynieść znacznie lepsze rezultaty. Z teoretycznego punktu widzenia sytuacja taka powinna zachodzić zawsze, gdyż jeśli oba słowniki przygotowane są rzetelnie, to muszą rozpoznawać te same homonimy. Sytuacja ta w praktyce wygląda jednak inaczej.

W procesie dopasowania wykorzystywane są z jednej strony – relacje semantyczne, za pomocą których opisane są hasła występujące w Słowniku Semantycznym, z drugiej – kwalifikatory dziedzinowe i inne środki wykorzystywane w słowniku polsko-angielskim do odróżnienia poszczególnych homonimów. Jak jednak wskazaliśmy w punkcie 5.2.2, informacja ta nie jest podawana w sposób systematyczny. Tym niemniej można np. stowarzyszyć poszczególne relacje semantyczne z różnymi wzorcami użycia słów i wykorzystać tę informację do rozpoznania tych relacji w słowniku polsko-angielskim. Na przykład: relacja meronimii może zostać stowarzyszona z wzorcem „w ...", tak jak w wyrażeniu: „zamek (w broni palnej)". Niestety rozwiązanie to musi uwzględnić dużą liczbę zróżnicowanych przypadków, co widać już na przytoczonym przykładzie – drugim członem relacji meronimii jest wyrażenie złożone („broni palnej"), ponadto nie jest ono użyte w mianowniku, etc. W Słowniku Semantycznym informacja ta przyjmuje co prawda postać: „zamek `IS_A_PART_OF` broń palna", więc może zostać łatwo dopasowana do informacji znajdującej się w słowniku polsko-angielskim, aczkolwiek widać jakiego rodzaju problemy mogą się tutaj pojawić.

Drugi problem w procesie mapowania automatycznego wynika z faktu, że nawet gdy w słowniku polsko-angielskim wybierzemy poprawnie odpowiedni homonim i gdy nawet okaże się, że posiada on w języku angielskim tylko jeden odpowiednik, to może się okazać, że ten angielski odpowiednik również jest słowem wieloznacznym. W tej sytuacji również można wykorzstać informację gramatyczną (predykat **`#$denotation`** wiąże dane słowo angielskie z danym konceptem Cyc zachowując typ gramatyczny tego słowa), aczkolwiek zazwyczaj nie jest ona wystarczająca. W tym wypadku proces dopasowania poszczególnych słow poprzez relacje semantyczne jest znacznie trudniejszy.

Pierwszy problem, który należy uwzględnić w realizacji procesu dopasowania słów języka polskiego do poszczególnych konceptów Cyc, dotyczy przynależności poszczególnych słów i konceptów do różnych języków naturalnych. Zatem jeśli mapujemy słowo „zamek (budynek)" na angielskie „castle" to musimy zwrócić uwagę na fakt, że jest on rodzajem „building" (**`#$genls`** **`#$Castle`** **`#$Building`**). Zatem aby wykorzystać tę informację musimy mieć wcześniej zmapowane polskie słowo „budynek" na koncept **`#$Building`**.

Problem ten można częściowo rozwiązać stosując, wcześniej wspomnianą, strategię „top-down", w której mapowanie rozpoczyna się od najbardziej ogólnych pojęć i postępuje w dół heterarchii. Pojęcia najbardziej ogólne mapowane są ręcznie, aby zapewnić ich poprawność. Oznacza to z jednej strony zmapowanie kategorii semantycznych Słownika na odpowiednie kolekcje Cyc, z drugiej zaś – znalezienia dla najbardziej ogólnych kolekcji występujących w Cyc polskich odpowiedników. Tym niemniej, jak zostało to zauważone w punkcie 3.4.1, przynależność do danej kategorii semantycznej nie może być w sposób pełny utożsamiona z relacją generalizacji/specjalizacji, co stanowi dodatkowy problem. Dlatego też mapowanie poszczególnych kategorii semantycznych na kolekcje Cyc ma nieco inny charakter – poszczególne kategorie nie są utożsamiane z kolekcjami Cyc. Semantyka tego mapowania jest następująca: słowa należące do kategorii zmapowanej na daną kolekcję (lub zbiór kolekcji), wchodzą z nią prawdopodobnie w relację **`#$genls`** (*resp.* wchodzą w tę relację z jedną kolekcją ze

zbioru), ale informacja ta nie jest wprowadzana bezpośrednio do ontologii Cyc, lecz może być wykorzystana jedynie w procesie mapowania.

Inny problem jaki wiąże się ze strategią „top-down" dotyczy faktu, że rozwiązuje ona problem braku mapowania pomiędzy słowami języka polskiego a konceptami Cyc, tylko w odniesieniu do relacji **#$genls**. Informacja niesiona przez pozostałe relacje może być również wykorzystana, ale nie ma gwarancji, że odpowiednie słowa już zostały zmapowane. Można w tym wypadku dokonać jedynie ograniczonego mapowania dla tych słów (o ile nie występuje problemu niejednoznaczności), które różni się tym od mapowania pełnego, że nie wykorzystuje informacji semantycznej, a jedynie informację słownikową. W przeciwnym bowiem razie stanęlibyśmy bezpośrednio przed problemem błędnego koła – aby zmapować słowo A, konieczne byłoby zmapowanie słowa B, do którego zmapowania potrzebne byłoby zmapowanie słowa A, itd.

Drugi problem występujący w procesie dopasowania polskich słów do konceptów Cyc, wiąże się z faktem, że gęstość pojęciowa Słownika Semantycznego oraz ontologii Cyc różni się znacznie. O ile w słowniku wybrana para słow może być połączona bezpośrednią relacją, o tyle w Cyc pomiędzy odpowiednikami tych słów może występować wiele pojęć pośredniczących. W tym wypadku konieczne staje się często przeglądanie jakiejś części heterarchii pojęć, co jest procesem dosyć problematycznym, szczególnie ze względu na trudność w ustaleniu kryteriów tego przeglądania (np. jak głęboko i jak dalego należy przeglądać heterarchię, która może mieć setki poziomów i setki tysięcy węzłów). W rozwiązaniu tego problemu konieczne jest wykorzystanie silnika inferencyjnego ontologii, który pozwala obliczyć np. domknięcia relacji, itp. Właśnie w tym wypadku owocny mógłby okazać się algorytm CtxMatch przedstawiony w [3].

Trzeci problem wiąże się z wyborem relacji, które powinny być uwzględnione w procesie dopasowywania – czy skoncentrować się wyłącznie na relacjach generalizacji/specjalizacji i meronimii, czy też uwzględnić inne relacje. Z jednej strony wykorzystując pozostałe relacje, można uzyskać większą precyzję dopasowania (jeśli np. dwa homonimy posiadają jakiegoś nieodległego przodka w heterarchii pojęciowej, to trudno będzie je rozróżnić wykorzystując jedynie relacje generalizacji/specjalizacji), z drugiej – należy liczyć się ze znaczną komplikacją procesu dopasowania oraz należy uwzględnić fakt, że mniej popularne relacje wykorzystywane w Słowniku Semantycznym (np. *konceptualne źródło zdarzenia* opisany w punkcie 3.4.2) mogą nie posiadać swoich ścisłych odpowiedników w ontologii Cyc. Trudno wskazać poprawne rozwiązanie tego problemu, nie dysponując odpowiednimi danymi empirycznymi, dlatego też powstrzymujemy się od tego.

Wymienione tutaj problemy nie są wszystkimi, z którymi można spotkać się w procesie automatycznego mapowania. Zważywszy jednak na fakt, że każdy z nich z osobna stanowi nie lada wyzwanie dla konstruktorów algorytmów, kończymy tę litanię przystępując do przedstawienia tej części mapowania, która z konieczności została dokonana ręcznie.

5.2.4. Elementy zmapowane ręcznie

Wielokrotnie zwracaliśmy uwagę, że niektóre elementy Słownika Semantycznego mogą zostać zmapowane na ontologię Cyc jedynie ręcznie. Dotyczy to przede wszystkim relacji oraz kategorii semantycznych.

Relacje

Poniżej przedstawiamy mapowanie poszczególnych relacji semantycznych występujących w Słowniku Semantycznym na odpowiadające im predykaty Cyc.

1. SYNONYMY: **#$synonymousExternalConcept**[8]
2. SIMILAR_TO: **#$overlappingExternalConcept**[9]
3. IS_A_KIND_OF: **#$genls**
4. IS_A: **#$genls**[10]
5. IS_A_PART_OF: **#$partTypes**
6. CONSISTS_OF: **#$partTypes**[11]
7. SOURCE:
8. DESTINATION: **#$typeIntendedBehaviorCapable**
9. ROLE: **#$someTypePlaysRoleInSituationType**
10. ACTION: **#$typeBehaviorCapable-PerformedBy**
11. STATE:
12. ACTOR: **#$agentTypePerformsActsOfType**
13. OBJECT: **#$objectActedOn-TypeType**
14. FROM: **#$from-Generic**
15. TO: **#$to-Generic**
16. INSTRUMENT: **#$eventTypeUsesDeviceType**
17. TIME: predykaty należące do kolekcji **#$ComplexTemporalPredicate**
18. PLACE: **#$eventOccursAtLocationType**
19. MOOD:

Dla niektórych relacji nie udało się znaleźć predykatów, które by im odpowiadały. Problem ten zostanie szerzej omówiony w rozdziale 6.

Kategorie semantyczne

Poniżej przedstawiamy mapowanie poszczególnych kategorii semantycznych występujących w Słowniku Semantycznym na odpowiadające im kolekcje Cyc. Semantyka tego mapowania, jak wcześniej wspominaliśmy, nie utożsamia kategorii semantycznych z kolekcjami Cyc, lecz wskazuje, że słowa należące do odpowiedniej kategorii są najprawdopodobniej specjalizacjami jednej z kolekcji Cyc stowarzyszonych z daną kategorią. Dlatego też dopuszczalne jest mapowanie jednej kategorii na więcej niż jedną kolekcję Cyc oraz mapowanie na kolekcje, które są uogólnieniami kolekcji, na które zmapowane zostały inne kategorie semantyczne.

1. HUMAN: **#$Person**, **#$HomoSapiens**
2. PLANT: **#$Plant**, **#$Fungus**

[8] Relacja synonimii nie występuje wewnątrz ontologii Cyc, gdyż z założenia każdy koncept posiada jedno-jednoznaczną identyfikację.

[9] Również relacja podobieństwa znaczeń nie występuje wewnątrz ontologii Cyc, a może zachodzić jedynie pomiędzy konceptami Cyc a konceptami występującymi w innym źródle wiedzy.

[10] Należy zwrócić uwagę, że choć relacje IS_A_KIND_OF oraz IS_A mapowane są na relację **#$genls**, to semantyka tego mapowania jest odmienna: „X IS_A: Y” znaczy (**#$genls** Y X), natomiast: „X IS_A_KIND_OF: Y” znaczy (**#$genls** X Y).

[11] Również w tym wypadku, należy zwrócić uwagę na fakt, że mapowanie to ma w obu przypadkach inną semantykę. „X IS_A_PART_OF Y” znaczy (**#$partTypes** Y X), natomiast „X CONSISTS_OF Y” (**#$partTypes** X Y).

3. ANIMAL: **#$NonPersonAnimal**, **#$NonHumanAnimal**
4. LOCATION: **#$EnduringThing-Localized**
5. PHYS_OBJ: **#$Partially-Tangible**
6. SET: **#$SetOrCollection**
7. STRUCTURE: **#$AbstractStructure**, **#$System**
8. NUMBER: **#$ScalarInterval**
9. TIME: **#$TimeInterval**, **#$Time-Quantity**
10. ABSTRACT_OBJ: **#$Intangible**
11. EVANT: **#$Situation**
12. NAME: **#$ProperNameString**
13. SELF: **#$Thing**

5.2.5. Przykład

Przykład pełnego mapowania słów występujących w Słowniku Semantycznym na odpowiednie kolekcje ontologii Cyc, przedstawiony jest w dodatku B.

Rozdział 6

Problemy mapowania

6.1. Wstęp

Pomiędzy Słownikiem Semantycznym o ontologią Cyc występuje szereg różnic, o czym pisaliśmy w rozdziale 5. Większość z nich została w pewnym stopniu przezwyciężona, aczkolwiek niektóre z nich rodzą problemy, których nie da się pokonać bez zmiany pewnych założeń przyjętych w konstrukcji którejś z mapowanych struktur. Problemy te uznaliśmy za nierozwiązywalne i przedstawione są one w niniejszym rozdziale.

6.2. Relacje

6.2.1. STATE

W poprzednim rozdziale wskazaliśmy, że niektóre relacje występujące w Słowniku Semantycznym nie posiadają swoich odpowiedników. Może się to wydawać bardzo dziwne, jeśli weźmie się pod uwagę fakt, że liczba wszystkich relacji i ich kombinacji (uwzględniających aspekty) sięga w Słowniku Semantycznym 32, podczas gdy w ontologii Cyc występuje ich ponad tysiąc. Problem tkwi właśnie w olbrzymiej liczbie relacji występujących w Cyc. Zamiast jednej relacji STATE, w Cyc spotykamy różnego rodzaju predykaty i kolekcje, które mogłyby służyć np. do wyrażenia zależności zachodzącej pomiędzy człowiekiem a jego stanem, *człowiek* STATE POSITIVE: *wesoły*.

W Cyc słowo angielskie „happy" będące odpowiednikiem słowa „wesoły" wiąże się z kolekcją **#$Happy** obejmującą wszystkich agentów, którzy są weseli. Kolekcja **#$AgentTypeByEmotio** jest kolekcją wszystkich kolekcji typu **#$Happy**, tzn. odnoszących się do grup agentów w określonym stanie emocjonalnym.

Kiedy jednak dalej będziemy eksplorować ontologię Cyc, to szybko okaże się, że to dopiero wierzchołek góry lodowej – łatwo bowiem możemy natknąć się na kolekcję **#$FeelingAtribut** które jest kolekcją wszystkich atrybutów emocjonalnych, jakie mogą zostać przypisane

agentom, a wśród nich możemy znaleźć **#$Happiness**. Przypisanie to dokonywane jest za pomocą predykatu **#$feelsEmotion**.

Zatem, wydawać by się mogło, że należy utożsamić relacje STATE z predykatem **#$feelsEmotion**. Rozwiązanie takie jest oczywiście błędne z co najmniej dwóch powodów: predykat **#$feelsEmotion** jest zbyt wąskim predykatem aby pokryć semantykę relacji STATE. Telewizor, który jest zepsuty, nie może być opisany za jego pomocą. Drugi powód dotyczy aspektów relacji STATE – wyróżniamy trzy jej aspekty, natomiast predykat **#$feelsEmotion** nie wprowadza takiego podziału.

6.2.2. ACTION

Inny przykład ilustrujący problemy związane z relacjami jest następujący: powszechnie wiadomo, że psy szczekają. W Słowniku Semantycznym informacja ta przyjmuje postać: „pies ACTION: szczekać". Pojawia się ona również w ontologii Cyc, tym niemniej w znacznie bardziej skomplikowanej postaci: (**#$animalTypeMakesSoundType #$Dog #$BarkingSound**). Jeśli myślimy, że predykat **#$animalTypeMakesSoundType** generalizuje się do jakiegoś odpowiednika relacji ACTION, to jesteśmy w błędzie – jego generalizacja opisana jest za pomocą skomplikowanej funkcji logicznej, której nie będziemy tutaj nawet przytaczać.

Pomimo tego, że relacja ACTION została utożsamiona z predykatem **#$typeBehavior-Capable-PerformedBy**, to utożsamienie to nie obejmuje predykatu **#$animalType-MakesSoundType**!

6.2.3. RELATED_TO

Inny problem związany z relacjami wiąże się z wykorzystaniem w Słowniku Semantycznym elementu opcjonalnego RELATED_TO – wyraża on fakt, że wybrana relacja zachodzi wyłącznie w określonym kontekście pozajęzykowym. Rozwiązanie to nie może być jednak w żaden sposób przeniesione do Cyc – co prawda można w tej ontologii reprezentować różne konteksty, za pomocą mikroteorii, ale konieczność tworzenia mikroteorii dla bardzo dużej klasy słów mijała by się zupełnie z celem, gdyż wykorzystanie zgromadzonej w nich informacji stało by się niezmiernie problematyczne. Dlatego też element opcjonalny RELATED_TO nie został w żaden sposób zmapowany.

6.3. Heterarchia konceptów

Liczba kategorii semantycznych w Słowniku Semantycznym jest niewielka. Natomiast liczba konceptów, które w ontologii Cyc w ogóle nie posiadają odpowiedników w postaci słów jest olbrzymia. Aby uzmysłowić sobie istotę problemu, warto prześledzić relacje jakie zachodzą pomiędzy konceptami **#$BarkingSound** oraz **#$Sound**. Na pierwszy rzut oka wydawać by się mogło, że powinny być one połączone bezpośrednio predykatem **#$genls**. Taka sytuacja miałaby z pewnością miejsce w Słowniku Semantycznym. Tym niemniej w ontologii Cyc sytuacja wygląda inaczej:
#$BarkingSound $\subset$ **#$AnimalSound** $\subset$ **#$AudibleSound** $\subset$ **#$Sound** $\subset \ldots$
Zatem pomiędzy tymi konceptami występują aż dwa koncepty pośredniczące, które nie mają swoich odpowiedników w języku.

Przykład ten ukazuje jak różna jest gęstość pojęciowa Słownik Semantycznego i on-

tologii Cyc. Pomimo tego, że wcześniej uznaliśmy, że nie stanowi to poważnego problemu, to w zasadzie należy uznać, że sytuacja taka ma miejsce tylko w niektórych przypadkach. Nieraz bowiem może zdarzyć się tak, że nie wiadomo będzie, do którego z elementów w tym łańcuchu należy przypisać wybrane słowo języka polskiego. Może „dźwięk” nie powinien być zmapowany na **#$Sound**, lecz na **#$AudibleSound**. Algorytm mapowania półautomatycznego sugeruje oba rozwiązania.

6.4. Podsumowanie

W rozdziale tym przedstawiliśmy tylko niektóre poważne problemy na jakie można się natknąć mapując Słownik Semantyczny na ontologię Cyc. Niestety, nie dają się one łatwo rozwiązać, rodząc pytanie, czy któryś z mapowanych systemów nie posiada błędnej konstrukcji – czy ubogie środki Słownika Semantycznego wystarczą do opisu rzeczywistości albo czy ontologia Cyc daje się w ogóle użytkować, ze względu na olbrzymi stopień swojej komplikacji. Odpowiedzi na te pytania mogą dostarczyć jedynie aplikacje, które korzystają z jednej z tych technologii. Być może słabości obu systemów mogą być przezwyciężone poprzez ich współpracę.

Część IV

Komputerowy model danych

Rozdział 7

Komputerowy model danych

7.1. Wstęp

Jak wiemy z poprzednich rozdziałów, pomiędzy Słownikiem Semantycznym a ontologią Cyc występuje wiele różnic semantycznych. Środki opisu rzeczywistości, opisywany język naturalny, przyjęty punkt widzenia to tylko niektóre elementy, które muszą zostać pogodzone. Jednakże problemy poziomu semantycznego to nie wszystkie problemy, z którymi musi uporać się twórca systemu pozwalającego na mapowanie tych struktur. Druga klasa problemów, z którymi trzeba się zmierzyć, pojawia się na poziomie technicznym i wynika z zastosowania różnych narzędzi programistycznych w konstrukcji systemów przechowujących i przetwarzających wymienione struktury.

W niniejszym rozdziale przedstawimy techniczną stronę tych systemów, z szczególnym uwzględnieniem dostępnych wywołań API oraz wykorzystywanych przez nie struktur danych. Przyjrzymy się również dodatkowym narzędziom, takim jak biblioteka `CLP` oraz elektroniczny słownik polsko-angielski, które niezbędne są do realizacji głównego celu tej pracy.

System pozwalający na mapowanie Słownika Semantycznego na ontologię Cyc, zwany *platformą mapowania* oraz wykorzystywane przez niego narzędzia i technologie zostaną przedstawione w dalszej części pracy. W tym miejscu chcemy jedynie zaznaczyć, że do jego realizacji wykorzystany został język Java [7], dlatego też przy opisie technicznej strony mapowanych struktur, uwzględniamy jedynie te mechanizmy, które są dostępne właśnie dla niego.

7.2. Słownik Semantyczny Języka Polskiego

Słownik Semantyczny Języka Polskiego jest podstawowym systemem współpracującym z platformą mapowania. W swej oryginalnej postaci funkcjonuje on jako baza danych zawierająca informacje o poszczególnych słowach, kategoriach i relacjach semantycznych oraz

jako zestaw skryptów języka `php`, służących do przeglądania i edytowania tej bazy. W niniejszej pracy przedstawiamy jedynie strukturę bazy danych Słownika Semantycznego, gdyż z jednej strony – towarzyszące jej skrypty nie zostały wykorzystane w platformie mapowania, a z drugiej – struktura bazy uległa pewnym zmianom, które motywowane były możliwością jej łatwiejszej współpracy z tą platformą.

7.2.1. Struktura bazy danych

W opisie struktury bazy danych, wykorzystywanej przez Słownik Semantyczny, przyjęto następujące konwencje typograficzne: klucze podstawowe wyróżnione zostały `podkreśleniem` natomiast obce – *`kursywą`*. `Pola` z innych `tabel`, do których odnoszą się klucze obce, reprezentowane są w następujący sposób: → `Tabela.pole`.

Concept

Tabela `Concept` przechowuje słowa i kategorie semantyczne:

1. `id : INTEGER` – bazodanowy identyfikator słowa.
2. `user_id : INTEGER` – identyfikator użytkownika, który opracowywał słowo (pole to jest wykorzystywane jedynie w celech informacyjnych).
3. `name : VARCHAR` – reprezentacja słowa w postaci łańcucha znaków.
4. `comment : VARCHAR` – komentarz dodany do słowa.
5. `script : VARCHAR` – skrypt związany ze słowami należącymi do kategorii `EVENT`. W obecnej chwili pole to jest niewykorzystywane.
6. `lexeme_id : INTEGER` – identyfikator leksemu z biblioteki `CLP` odpowiadającego danemu słowu.

Relation

Tabela `Relation` przechowuje relacje semantyczne:

1. `id : INTEGER` – bazodanowy identyfikator relacji semantycznej.
2. *`name_id`* `: INTEGER` – identyfikator nazwy relacji → `Relation_name.id`.
3. *`aspect_id`* `: INTEGER` – identyfikator aspektu relacji → `Relation_aspect.id`.
4. *`related_id`* `: INTEGER` – identyfikator dodatkowego atrybutu relacji → `Relation_related.id`.

Relation_aspect

Tabela `Relation_aspect` przechowuje aspekty relacji semantycznych:

1. `id : INTEGER` – bazodanowy identyfikator aspektu.
2. `name : VARCHAR` – nazwa aspektu.

Relation_name

Tabela `Relation_name` przechowuje nazwy poszczególnych relacji semantycznych:

1. `id : INTEGER` – bazodanowy identyfikator nazwy.
2. `name : VARCHAR` – nazwa relacji.

Relation_related

Tabela `Relation_related` przechowuje dodatkowe atrybuty relacji semantycznych:

1. `id : INTEGER` – bazodanowy identyfikator atrybutu relacji.
2. `name : VARCHAR` – nazwa atrybutu relacji.

Tuple

Tabela `Tuple` przechowuje krotki relacji semantycznych:

1. `id : INTEGER` – bazodanowy identyfikator krotki.
2. *`relation_id`* `: INTEGER` – identyfikator relacji, do której należy dana krotka → `Relation.id`.
3. `sub_group : INTEGER` – identyfikator grupy krotek, do której należy dana krotka (niewykorzystywany).
4. *`concept1_id`* `: INTEGER` – identyfikator konceptu, który jest pierwszym elementem krotki → `Concept.id`.
5. *`concept2_id`* `: INTEGER` – identyfikator konceptu, który jest drugi elementem krotki → `Concept.id`.
6. *`concept3_id`* `: INTEGER` – identyfikator konceptu, który jest trzecim elementem krotki → `Concept.id`. Jeżeli krotka należy do relacji dwuargumentowej, pole to ma wartość `null`.

V_relation

Tabela `V_relation` jest zmaterializowanym widokiem powstałym przez wywołanie następującego zapytania `SQL`:

```
SELECT id,
relation_name.name AS name,
relation_aspect.name AS aspect,
relation_related.name AS related
FROM relation LEFT OUTER JOIN relation_aspect
ON relation.aspect_id = relation_aspect.id
LEFT OUTER JOIN relation_name
ON relation.name_id = relation_name.id
LEFT OUTER JOIN relation_related
ON relation.related_id = relation_related.id
```

Została ona wprowadzona ze względu na ułatwienia w mapowaniu obiektowo-relacyjnym oraz zwiększenie efektywności działania bazy danych.

7.3. Ontologia CYC

Ontologia Cyc rozpowszechniana jest jako baza wiedzy wyposażona w rozbudowany silnik inferencyjny oraz systemy pomocnicze objemujące m.in. interpreter poleceń języków `SubL` i `CycL` oraz serwer pozwalający przeglądać zawartość ontologii za pomocą przeglądarki internetowej i manipulować zgromadzonymi w niej danymi za pomocą specjalizowanego protokołu internetowego `CFASL`.

Z technicznego punktu widzenia ontologia ta jest systemem bardzo skomplikowanym. Świadczy o tym chociażby rozmiar kodu źródłowego silnika inferencyjnego liczony w kilku milionach wierszy. Możliwość jej współpracy z programami zewnętrznymi zapewniona jest przez interfejs programistyczny zbudowany na bazie protokołu `CFASL`. Jest on dostępny dla języka Java jako biblioteka `OpenCyc.jar`.

7.3.1. OpenCYC API

Interfejs programistyczny oferowany przez bibliotekę `OpenCyc` zawiera dwa podstawowe pakiety: `org.opencyc.api` oraz `org.opencyc.cycobject`. Każdy z tych pakietów zawiera szereg klas, które pozwalają na współpracę ontologii z zewnętrznymi programami, aczkolwiek aby współpraca ta była efektywna, wystarczy skorzystać z niewielkiego ich podzbioru.

Najważniejszym elementem interfejsu jest klasa `org.opencyc.api.Access`, która odpowiedzialna jest za komunikację z serwerem Cyc. Do konstruktora tej klasy można przekazać szereg parametrów obejmujących adres hosta i port na którym działa serwer ontologii, sposób komunikacji, etc.

Po utworzeniu obiektu klasy `Access`, cała ontologia dostępna jest do naszej dyspozycji, gdyż obiekt ten pozwala wywoływać bezpośrednio dowolne poprawne zapytanie języka `CycL`. Tym niemniej, ten sposób komunikacji z ontologią jest nieco niewygodny, gdyż każde wywołanie języka `CycL` musi zostać sformułowane w postaci odpowiedniego łańcucha znaków lub obiektu `org.opencyc.cycobject.CycList`. Dlatego też klasa `Access` zawiera szereg metod, które często wykorzystuje się współpracując z serwerem Cyc. Poniżej przedstawiamy jedynie te wywołania, które były przydatne w realizacji platformy mapowania. Przytaczanie pełnej listy wywołań mija się z celem, m.in. dlatego, że jest ona bardzo długa.

Najczęściej stosowane metody klasy `Access`:

1. `CycConstant find(String searchString)` – zwraca stałą (koncept, predykat, mikroteorię, etc.), której identyfikator jest identyczny z wartością parametru `searchString` lub warotść `null`, jeśli nie ma takiej stałej.
2. `Object conversObject(Object command)` – zwraca obiekt, który jest rezultatem wywołania komendy `command` języka `CycL`. Komenda ta może być przekazana w postaci obiektu klasy `String` lub `CycList`.
3. `boolean isCollection(CycObject cycObject)` – sprawdza czy dany obiekt jest kolekcją.
4. `boolean isIndividual(CycObject cycObject)` – sprawdza czy dany obiekt jest indywiduum.
5. `boolean isPredicate(CycObject cycObject)` – sprawdza czy dany obiekt jest predykatem.
6. `void assertSynonymousExternalConcept(CycFort cycTerm, CycFort informationSource, String externalConcept, CycObject mt)` – dodaje do mikroteorii `mt` asercję stwierdzającą, że występujący w Cyc koncept `cycTerm` jest synonimiczny z konceptem `externalConcept` występującym w źródle wiedzy `informationSource`.
7. `void assertGaf(CycList gaf, CycObject mt)` – dodaje do mikroteorii `mt` asercję `gaf`.
8. `void unassertGaf(CycList gaf, CycObject mt)` – usuwa asercję `gaf` z mikroteorii `mt`.
9. `CycList queryVariable(CycVariable queryVariable, CycList query, CycObject mt, HashMap queryProperties)` – zwraca wszystkie obiekty, które w formule `query` mogą zastąpić zmienną `queryVariable`, tak aby utworzona w

ten sposób asercja, była prawdziwa w mikroteorii `mt`. Zmienna `queryProperties` zawiera zmienne wpływające na sposób działania silnika wnioskującego.

10. `CycList queryVariables(CycList queryVariables, CycList query, CycObject mt, HashMap queryProperties)` – działa podobnie do wywołania opisanego w poprzednim punkcie, z tą różnicą, że w formule może występować więcej niż jedna zmienna.
11. `boolean isQueryTrue(CycList query, CycObject mt, HashMap queryProperties)` – sprawdza, czy formuła `query` jest prawdziwa w mikroteorii `mt`. Zmienna `queryProperties` – j.w.
12. `CycFort createIndividual(String individualName, String comment, CycFort commentMt, CycFort isa)` – tworzy nowe indywidum o nazwie `individualName`, opatrzone w mikroteorii `commentMt` komentarzem `comment` oraz należące do kolekcji `isa`.
13. `void kill(CycConstant cycConstant)` – usuwa stałą `cycConstant` z ontologii Cyc.
14. `assertIsa(String cycFortName, String collectionName, String mtName)` – dodaje do kolekcji `mtName` asercję stwierdzającą, że obiekt `cycFortName` należy do kolekcji `collectionName`.
15. `boolean hasSomePredicateUsingTerm(CycConstant predicate, CycFort term, Integer argumentPosition, CycObject mt)` – sprawdza, czy w mikroteorii `mt` występuje asercja wykorzystująca predykat `predicate`, w której obiekt `term` występowałby na pozycji `argumentPosition`.

7.3.2. Struktury danych

W wywołaniach przedstawionych w poprzednim punkcie pojawia się kilka klas z pakietu `org.opencyc.cycobject`, wykorzystywanych do wymiany danych z serwerem Cyc. Ich charakterystyka przedstawiona jest poniżej:

1. `CycObject` – interfejs, który implementują wszystkie obiekty wymieniane z serwerem Cyc, które nie są typami podstawowymi języka Java (takimi jak `String`, `int`, czy `boolean`). Zawiera on wywołania pozwalające na konwersję obiektu do łańcucha znaków i dokumentu XML; pobranie stałych, które wykorzystywane są przez obiekt, itp.
2. `CycFort` – klasa abstrakcyjna służąca do reprezentacji obiektów, które posiadają w Cyc unikalny identyfikator liczbowy. Zawiera wywołania pozwalające manipulować tym identyfikatorem oraz porównywać dany obiekt z innymi obiektami występującymi w Cyc.
3. `CycConstant` – klasa dziedzicząca z klasy `CycFort`, służąca do reprezentacji obiektów, które posiadają w Cyc unikalną nazwę, takich jak kolekcje, indywidua, mikroteorie, predykaty, itp. Zawiera m.in. wywołanie pozwalające pobrać nazwę danego obiektu.
4. `CycVariable` – klasa służąca do reprezentowania zmiennych w zapytaniach języka `CycL`.
5. `CycList` – klasa ta reprezentuje listę obiektów wymienianych z serwerem Cyc. Może ona zawierać obiekty klasy `CycObject` jak i obiekty niektórych klas podstawowych języka Java. Klasa ta jest odpowiednikiem formuł języka `CycL`, dlatego stosuje się ją m.in. do formułowania zapytań w tym języku oraz reprezentowania ich wyników. Za-

wiera ona wywołania pozwalające tworzyć formuły tego języka poprzez dodawanie i usuwanie odpowiednich obiektów, np. stałych `CycConstant` czy zmiennych `CycVariable`.

Kiedy przyglądamy się strukturom występującym w pakiecie `org.opencyc.cycobject` uderza nas fakt, że nie znajdujemy tam bezpośrednich reprezentantów elementów z poziomu semantycznego, takich jak indywidua, kolekcje, mikroteorie, czy predykaty. Wszystkie one reprezentowane są za pomocą klasy `CycConstant`, a ich charakter rozpoznawany jest za pomocą odpowiednich wywołań API.

7.4. Biblioteka CLP

Jednym z podstawowych problemów pojawiających się w komputerowym przetwarzaniu języków fuzyjnych jest konieczność rozpoznawania różnych form danego leksemu. Bez obecności specjalizowanej biblioteki, która zawiera informacje o formach fleksyjnych poszczególnych słów, przetwarzanie to jest właściwie niemożliwe. Konieczność wykorzystania biblioteki tego typu w opisywanym tutaj projekcie wynika z faktu, że poszczególne hasła występujące w Słowniku Semantycznym oraz w słowniku angielsko-polskim zawierają słowa w formach obocznych.

CLP [10] jest biblioteką napisaną w języku C++ pozwalającą uporać się z tym problemem. Poszczególne leksemy posiadają unikatowe identyfikatory liczbowe, a biblioteka zawiera wywołanie, które na podstawie formy danego wyrazu określa, do których leksemów może on należeć. Wywołanie to oraz pozostałe wywołania dostępne w bibliotece opisane są poniżej:

1. `void clp_init(int tryb)` – inicjalizacja biblioteki.
2. `void clp_info(struct info_res * out)` – zwraca w strukturze `info_res` podstawowe informacje dotyczące biblioteki.
3. `void clp_rec(const char* inp, int* out, int* num)` – zwraca w tablicy `out` identyfikatory leksemów, których jedna z form identyczna jest z argumentem `inp`. Liczba rozpoznanych leksemów zwracana jest poprzez parametr `num`.
4. `void clp_bform(int id, unsigned char* out)` – zwraca w parametrze `out` formę podstawową leksemu o identyfikatorze `id`.
5. `void clp_forms(int id, unsigned char* out)` – zwraca w parametrze `out` wszystkie formy leksemu o identyfikatorze `id`. Poszczególne formy oddzielone są dwukropkiem.
6. `int clp_type(int id)` – zwraca typ leksemu o identyfikatorze `id`: 1 – jednoczłonowy leksem pospolity; 2 – dwuczłonowy czasownik zwrotny.
7. `int clp_class(int it)` – zwraca klasę gramatyczną (fleksyjną) leksemu o identyfikatorze `id` (patrz [10, s. 2]).
8. `void clp_label(int id, char* out)` – zwraca etykietę fleksyjną wyrazu (patrz [21]).

7.5. Słownik polsko-angielski Oxford/PWN

Wielki Multimedialny Słownik angielsko-polski/polsko-angielski Oxford/PWN[1] jest programem przeznaczonym do uruchomienia w systemie ©Windows. Ponadto nie był on twor-

[1] Dla wygody dalej zwany „Słownikiem angielsko-polskim” lub „Słownikiem polsko-angielskim” w zależności od tego, do której jego części będziemy się odwoływać.

zony z myślą o współpracy z innymi programami komputerowymi, dlatego nie udostępnia żadnego API.

Skorzystanie z tego słownika jako podstawowego narzędzia umożliwiającego dokonanie automatycznego lub półautomatycznego mapowania rodzi wiele problemów natury technicznej. Pierwszy z nich dotyczy rozpoznania *fizycznej* struktury słownika, przez co rozumiemy tutaj rozpoznanie, które spośród plików wykorzystywanych przez słownik zawierają dane słownikowe oraz rozpoznanie sposobu przechowywania haseł w tych plikach. Drugi problem, który pojawia się, kiedy chcemy skorzystać z tego słownika to konieczność rozpoznania *logicznej* struktury haseł, przez co rozumiemy rozpoznanie organizacji informacji wewnątrz poszczególnych haseł.

O ile przy rozpoznaniu struktury fizycznej mogliśmy posłużyć się opensourcowym projektem `Kydpdict`[2], który pozwala uruchomić słownik na systemie operacyjnym `Linux`, o tyle struktura logiczna haseł wymagała samodzielnego, dogłębnego zbadania.

7.5.1. Struktura fizyczna

Podstawowe dane słownika polsko-angielskiego zgromadzone są w pliku `polang.win`, który domyślnie po instalacji w systemie ©Windows znajduje się w katalogu `c:\Program Files\PWN\WSPWNOUP2004\Data`. Plik ten składa się z trzech części: nagłówek zawiera dane na temat fizycznego rozmieszczenia dwóch pozostałych części oraz liczby dostępnych haseł; indeks zawiera offsety poszczególnych haseł; trzecia część zawiera hasła słownikowe.

Hasła słownikowe składają się z polskiego słowa (etykiety), po którym następuje pozostała część hasła. Jest ona zwykle skompresowana za pomocą algorytmu `zip`.

7.5.2. Struktura logiczna

W idealnej sytuacji struktura logiczna haseł słownika byłaby ściśle oddzielona od ich reprezentacji graficznej i zapisana byłaby w jakimś powszechnie uznanym standardzie, np. `XML`. Niestety rzeczywistość jak zwykle znacząco odbiega od ideału. Nie dość, że hasła słownika zapisane są w języku jedynie przypominającym `XML`, nie dość, że ich struktura logiczna miesza się na wielu poziomach z ich reprezentacją graficzną, to jeszcze poszczególne hasła wykorzystują niekompatybilne schematy definicyjne.

W dodatku B umieszczonych jest kilka haseł zaczerpniętych ze słownika, które ilustrują omawiane tutaj problemy. Na ich podstawie trudno zbudować jakiś konsystentny obraz struktury logicznej tych haseł, aczkolwiek spróbujemy tego dokonać.

Elementy występujące we wszystkich hasłach

Pierwszy element, który występuje we wszystkich hasłach słownikowych to etykieta danego hasła, czyli słowo w języku polskim, po którym następuje skrót kategorii gramatycznej, do której ono należy.

Drugi element, który wspólny jest wszystkim hasłom, to wykorzystanie tagów `<PL>` oraz `<GB>` do odróżnienia słów i wyrażeń należących odpowiednio: do języka polskiego oraz angielskiego.

[2] http://members.elysium.pl/ytm/html/kydpdict.html

Homonimy polskie

Element najbardziej interesujący z naszego punktu widzenia, czyli sposób wyróżniania poszczególnych homonimów polskich jest chyba najsłabiej dopracowany.

Przeanalizujmy najpierw hasło *zamek*: poszczególne homonimy są tutaj wyróżnione poprzez wypunktowanie z wykorzystaniem numeracji arabskiej. Tłumaczenie każdego homonimu poprzedzone jest dodatkową informacją w języku polskim, pozwalającą na odróżnienie go od pozostałych homonimów.

W tym wypadku widać wyraźnie, że struktura logiczna pomieszana jest ze sposobem prezentacji, gdyż ta dodatkowa informacja wyróżniona jest za pomocą tagu `<SMALL>` oraz nawiasów okrągłych „`( )`”. Ponadto, o czym pisaliśmy szerzej w punkcie 5.2.2, informacja ta nie jest podawana według jakiegoś ogólnego schematu semantycznego, co nie stanowi problemu dla ludzi, ale stanowi poważną barierę dla jej wykorzystania w algorytmach mapowania. Co więcej, chociaż informacja ta jest zwykle podawana po numerze porządkowym, to czasem zdarza się (3. homonim hasła *zamek*), że jest poprzedzona dodatkową informacją gramatyczną.

W przypadku tego hasła dziwi nas jeszcze jeden fakt – wyrażenia złożone którym odpowiada jeden wyraz angielski, również nie są traktowane jednakowo. *Zamek skałkowy* umieszczony jest w głównej części hasła w grupie homonimu opatrzonego etykietą *w broni palnej*. Natomiast wyrażenie *zamek błyskawiczny*, które mogłoby znaleźć się w grupie pierwszego homonimu, opatrzonego etykietą *do zamykania*, znajduje się w sekcji wyrażeń złożonych, które wyróżnione są w postaci źródłowej hasła za pomocą słowa kluczowego `&square`.

Nie jest to jednak koniec problemów – kiedy przyjrzymy się hasłu *amerykanizować*, zauważymy, że w tym wypadku dwa różne leksemy *amerykanizować* oraz *amerykanizować się* zostały opatrzone jedną etykietą, co mogłoby sugerować, że są homonimami. Z kolei przy haśle *zamęczyć* oba homonimy posiadają odrębne etykiety: *zamęczyć*[1] oraz *zamęczyć*[2]. Dodatkowo hasło, które zawiera pierwszy homonim zawiera również odrębny leksem *zamęczyć się*. Żeby tego było mało, to tylko pierwszy homonim posiada dodatkową informację pozwalającą zidentyfikować jego znacznie (*doprowadzać do wyczerpania*).

Wszystkie te problemy bledną jednak po zanalizowaniu hasła *amortyzować*. Jest to idealny przykład bałaganu jaki panuje w strukturze haseł słownikowych – pomimo, że jest to jedno hasło, wykorzystano w nim aż trzy różne sposoby opisu poszczególnych homonimów.

Wszystkie homonimy wewnątrz tego hasła opatrzone są kwalifikatorami dziedzinowymi (*Ekon.*, *Techn.*). Kwalifikatory te opatrzone są tagami `<SMALL>`, tak jak to miało miejsce w przypadku hasła *zamek*, ale zrezygnowano z nawiasów. Nie stanowi to jednak wielkiego problemu, gdyż liczba kwalifikatorów dziedzinowych jest niewielka. To co jednak uderza nas w tym haśle to dodatkowa informacja pozwalająca identyfikować poszczególne homonimy. Po pierwsze – zamiast zastosować nawiasy okrągłe oraz tag `<SMALL>`, tak jak to miało miejsce w przypadku hasła *zamek*, zastosowano nawiasy kwadratowe „`[]`” oraz tag `<I>`. Po drugie – informacja ta umieszczana jest to przed angielskim odpowiednikiem słowa (*maszyna, urządzenie*), to znowu po nim (*koszty, wstrząs, uderzenie, upadek*). Najdziwniejsze jest jednak to, że w jednym podpunkcie arabskim (2.), wyróżniono dwa homonimy, przez co nie wiadomo właściwie czy kwalifikator *Tech.* stosuje się do nich obu, czy tylko do pierwszego.

Prawdopodobnie, gdyby przytoczono jeszcze inne hasła, to można by znaleźć jeszcze więcej niekonsekwencji w wyróżnianiu poszczególnych homonimów. Poprzestaniemy jed-

nak na tych wymienionych tutaj, gdyż i tak stanowią one doskonałą ilustrację problemów z jakimi musi zmierzyć się konstruktor algorytmu parsowania haseł słownikowych.

Synonimy angielskie

Na szczęście sytuacja z synonimami angielskimi wygląda znacznie lepiej niż z homonimami polskimi. Synonimy angielskie odpowiadające jednemu słowu lub homonimowi polskiemu oddzielone są od siebie przecinkiem „`,`”. Tym niemniej kiedy przyglądamy się postaci źródłowej tłumaczenia wyrażenia złożonego *zamek błyskawiczny*, które posiada w brytyjskiej odmianie języka angielskiego inny odpowiednik niż w amerykańskiej (odpowiednio: *zip*, *zipper*), to zauważamy że pomiędzy nimi występuje tajemniczy tag `<TEXTSECTION>`, którego przeznaczenia autorowi niniejszej pracy nie udało się rozpoznać do dnia dzisiejszego.

Przykłady użycia

W większości haseł słownikowych pojawiają się przykłady użycia poszczególnych słów. Występują one po wyrazie lub wyrazach będących angielskimi odpowiednikami danego słowa/homonimu. Poszczególne użycia oddzielone są średnikami „`;`” i każde z nich składa się z wyrażenia polskiego, opatrzonego tagiem `<B>` oraz jego angielskiego odpowiednika. Czasami w tym samym miejscu pojawiają się kolokacje (co można zaobserwować na przykładzie hasła *zamek*), co jest o tyle dziwne, że zazwyczaj wyróżnione są one za pomocą specjalnych oznaczeń, o których mowa w następnym punkcie.

Kolokacje i powiedzenia

Kolokacje i powiedzenia występują zawsze na końcu hasła. Pierwsze wyróżnione są za pomocą słowa kluczowego `&square`. Po nim następuje lista kolokacji oddzielonych średnikami.Powiedzenia, w których występuje dane słowo, występują po kolokacjach i wyróżnione są słowem kluczowym `&squareb`. Również one odseparowane są od siebie średnikami.

Grupy gramatyczne

Ostatnim ważnym elementem struktury logicznej haseł są sekcje wyróżnione cyframi rzymskimi. Na wstępie warto zwrócić uwagę, że również w tym wypadku mamy do czynienia ze zmieszaniem struktury logicznej z informacją o reprezentacji graficznej hasła, gdyż do oddzielenia poszczególnych sekcji wykorzystuje się tag `<IMG>`, którego atrybut `SRC` wskazuje plik graficzny zawierający daną cyfrę rzymską. Problem ten można oczywiście łatwo pokonać utożsamiając numerację sekcji z odpowiednimi plikami (1 : `rzym1.jpg`, 2 : `rzym2.jpg`, etc.), aczkolwiek nigdy nie wiadomo, czy w kolejnej wersji słownika odpowiednie pliki graficzne nie zostaną zmienione...

Znacznie poważniejszy problem wiąże się z zawartością owych sekcji. Z pobieżnych obserwacji kilku haseł wynika, że w sekcjach tych znajdują się słowa, które choć posiadają tę samą etykietę, należą do różnych kategorii gramatycznych (por. *afro* – przymiotnik oraz rzeczownik). Po numerze każdej sekcji następuje etykieta kategorii gramatycznej. Sytuacja ta komplikuje się jednak, kiedy uwzględnimy hasło *admirał* – w tym wypadku słowa w obu sekcjach należą do tej samej kategorii gramatycznej (rzeczownik), przy czym pierwsze jest obiektem ożywionym, a drugie nieożywionym. Istotne jest jednak to, że zachowano w tym wypadku układ logiczny haseł – tutaj odpowiednia etykieta również pojawia się po numerze sekcji.

Kiedy jednak spojrzymy na hasło *amerykanizować*, to dostrzeżemy, że ta reguła nie jest

stosowana zawsze. W wypadku kiedy występują dwa czasowniki – jeden zwrotny a drugi nie, to czasownik niezwrotny i jego wariacje występują w początkowych sekcjach, natomiast czasownik zwrotny występuje jako ostatni. Niestety zamiast umieścić odpowiednią etykietę gramatyczną, w tym wypadku po numerze sekcji występuje cały zwrot: *amerykanizować się*.

Co więcej na przykładzie tego hasło można również dostrzec dziwny sposób wykorzystania odsyłaczy – hasła *amerykanizować (się)* oraz *zamerykanizować (się)* odsyłają do siebie nawzajem. Jeśli chciałoby wykorzystać ten mechanizm w programie komputerowym, to szybko doszłoby do zapętlenia.

Część V

Platforma mapowania

Rozdział 8

Architektura systemu

Platforma mapowania ontologii charakteryzuje się strukturą modułową. Jej architektura przedstawiona jest na rysunku 8.1.

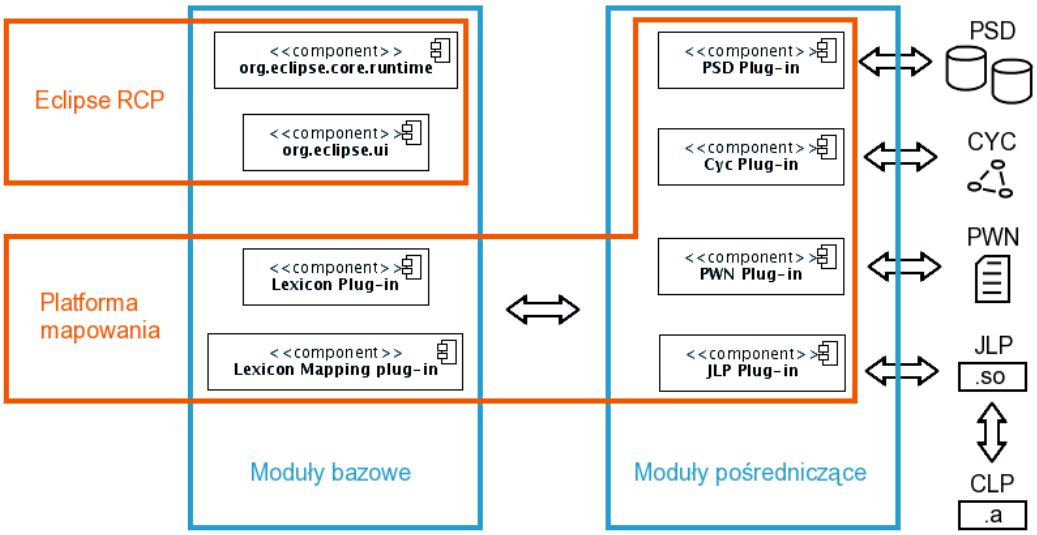
Elementy, które tworzą platformę oraz jej środowisko działania, dzielą się na: *bazowe*, *pośredniczące* oraz *zewnętrzne*.

Elementy *bazowe* tworzą jądro platformy i zawierają podstawowe struktury danych i algorytmy oraz elementy graficznego interfejsu użytkownika. Wśród tych elementów można wyróżnić platformę `Eclipse RCP` dostarczającą elementy GUI oraz infrastrukturę pozwalającą budować platformę mapowania w sposób modułowy. Dalej zaś możemy wyróżnić moduł `Lexicon Plug-in`, w którym zdefiniowane są podstawowe struktury danych wykorzystywane w pozostałych częściach systemu oraz moduł `Lexicon Mapping Plug-in` zawierający algorytmy mapowania oraz elementy GUI wykorzystywane przy mapowaniu. Platforma `Eclipse RCP` opisana jest w rozdziale 9, moduł `Lexicon Plug-in` – w rozdziale 10 natomiast moduł `Lexicon Mapping Plug-in` – w rozdziale 13.

Elementy *pośredniczące* odpowiadają za konwersję i wymianę danych pomiędzy platformą a współdziałającymi z nią systemami zewnętrznymi. Moduł `PSD Plug-in` odpowiada za współpracę ze Słownikiem Semantycznym Języka Polskiego, `Cyc Plug-in` – z ontologią Cyc, `PWN Plug-in` – ze Słownikiem polsko-angielskim Oxford/PWN, zaś `JLP Plug-in` – z biblioteką `CLP`. Moduły te opisane są w rozdziale 11.

Elementy *zewnętrzne* tworzą środowisko działania platformy. Charakteryzują się tym, że powstały i egzystują w sposób całkowicie[1] niezależny od niej. Są one źródłem oraz ujściem danych, które przetwarzane są w ramach platformy.

[1] Z wyjątkiem Słownika Semantycznego, w którym struktura bazy danych uległa pewnym zmianom, aby ułatwić jego współpracę z Platformą mapowania.



Rysunek 8.1. Architektura *Platformy mapowania ontologii*.

Rozdział 9

Wykorzystywane technologie

9.1. Wstęp

Z analizy przedstawionej w rozdziale 7 wynika, że współpracujące systemy są wysoce heterogeniczne. Każdy z nich wykorzystuje odrębny model danych i udostępnia zupełnie inne API. Aby móc zrealizować platformę mapowania, konieczne jest wybranie narzędzi, które pozwolą na dopasowanie tych systemów do siebie. Ponadto, aby umożliwić mapowanie ręczne oraz półautomatyczne, konieczne jest wyposażenie platformy mapowania w graficzny interfejs użytkownika.

Te wymagania wpłynęły w znaczący sposób na kryteria wyboru narzędzi wykorzystywanych do realizacji platformy mapowania. Najważniejsze z nich wymienione są poniżej:

1. Możliwość łatwej integracji heterogenicznych systemów.
2. Możliwość wypracowania wspólnych struktur danych dla różnych modeli danych.
3. Dostępność na różne platformy systemowe.
4. Dojrzałość wykorzystywanych rozwiązań.
5. Możliwość tworzenia graficznego interfejsu użytkownika.

9.2. Java i narzędzia z nią związane

9.2.1. Język Java

Język Java [7] spełnia dość dobrze wymienione wcześniej kryteria, gdyż posiada wiele cech pożądanych przy realizacji platformy mapowania.

Pierwsza cecha, która została uwzględniona, to jego przenośność – jest on dostępny na niemal wszystkie platformy sprzętowe i systemowe. Ponadto nie wymaga on rekompilacji kodu źródłowego po przeniesieniu na inną platformę sprzętową lub programową, jak to ma miejsce np. w przypadku języka C++.

Jest to typizowany język obiektowy wysokiego poziomu. Dzięki temu uzyskujemy wszelkie pozytywne cechy języków obiektowych – łatwą reużytkowalność kodu, niezawodność, wysoki stopień abstrakcji, itp. Cechy ta sprawiają, że dobrze nadaje się on do tworzenia aplikacji wyposażonych w graficzny interfejs użytkownika, który jest integralnym elementem platformy mapowania.

Bardzo ważną cechą, niejednokrotnie decydującą o wyborze tego czy innego języka, jest dostępność odpowiednich narzędzi, szeroko wykorzystywanych we współczesnych technikach tworzenia aplikacji. Dotyczy to specjalizowanych narzędzi informujących o przebiegu działania programu (loggerów, zwanych również dziennikami); pozwalających na automatyzację kompilacji, archiwizowania i testowania poszczególnych elementów systemu, etc.

W realizacji platformy mapowania wykorzystane zostały trzy narzędzia tego typu – `log4j`[1] do logowania, `jUnit`[2] do łatwego tworzenia testów oraz `Ant`[3] do automatyzacji procesu kompilacji, budowania i archiwizacji systemu. Poza tym wykorzystano również przenośny system zarządzania bazami danych napisany w języku Java – `HSQLDB` oraz narzędzie mapowania obiektowo-relacyjnego dla Javy – `Hibernate`. Narzędzia te zostały omówione w następnych punktach.

9.2.2. HSQLDB

`HSQLDB`[4] jest systemem zarządzania relacyjnymi bazami danych napisanym w całości w języku Java. System ten jest wykorzystywany do zarządzania bazą danych wykorzystywaną przez Słownik Semantyczny Języka Polskiego. O wyborze tego systemu zadecydowała przede wszystkim możliwość bezproblemowego przenoszenia go na różne platformy sprzętowe i systemowe oraz możliwość wykorzystania prostego sposobu przechowywania danych opartego o pliki. Dodatkowym atutem, który przemawia na jego korzyść, jest możliwość uruchomienia go z poziomu programowego.

9.2.3. Hibernate i XDoclet

`Hibernate`[5] jest systemem mapowania obiektowo-relacyjnego przeznaczonym dla języka Java. Dzięki jego wykorzystaniu współpraca relacyjnych baz danych z językami obiektowymi staje się znacznie łatwiejsza – znika potrzeba pisania żmudnych instrukcji SQL, które muszą być nieustannie przepisywane aby podążać za zmianami w modelu obiektowym. Zamiast tego dostęp do poszczególnych struktur danych odbywa się bezpośrednio z poziomu obiektowego języka programowania, zachowując wszystkie jego pozytywne cechy i udogodnienia: polimorfizm, dziedziczenie, kompozycję, struktury asocjacyjne oraz kolekcje.

System `Hibernate` został wykorzystany ze względu na jego dojrzałość oraz możliwość współpracy z najrozmaitszymi systemami zarządzania bazami danych. Dzięki temu, zamiana jednego systemu na inny jest znacznie łatwiejsza, niż w przypadku, w którym język

[1] http://logging.apache.org/log4j/docs/
[2] http://www.junit.org/index.htm
[3] http://ant.apache.org/
[4] http://www.hsqldb.org/
[5] http://www.hibernate.org/

SQL wykorzystuje się bezpośrednio – ingerencja przeprowadzana jest w jednym pliku konfiguracji, bez konieczności modyfikowania kodu źródłowego programu.

W połączeniu z `Hibernate` wykorzystana została technologia `XDoclet`[6], która pozwala na definiowanie mapowania obiektowo-relacyjnego w tych samych plikach, w których znajduje się kod źródłowy programu. Dzięki temu wszelkie informacje związane z wybraną strukturą danych wymagającą przechowywania w bazie danych zorganizowane są w jednym pliku, umożliwiając znacznie lepszą kontrolę nad całością, niż w przypadku, w którym mapowanie obiektowo-relacyjne zdefiniowane jest w osobnym pliku.

9.3. Eclipse RCP

9.3.1. Wstęp

System `Eclipse`[7] to zestaw otwartych narzędzi służących do tworzenia oprogramowania. Są one napisane w języku Java i początkowo pozwalały na tworzenie oprogramowania tylko w tym języku. W obecnej chwili, dzięki zastosowaniu mechanizmu wtyczek, który omówiony jest w następnym punkcie, powstaje wiele specjalizowanych narzędzi dla różnych języków programowania, które zintegrowane są z tym systemem.

Najciekawszym, z naszego punktu widzenia, elementem systemu jest `Eclipse Rich Client Platform`[8] – zestaw bibliotek, które można bezpośrednio wykorzystać do stworzenia zupełnie samodzielnej aplikacji, wyposażonej w bogaty interfejs użytkownika. Posiada on kilka istotnych cech, które przyczyniają się do jego rosnącej popularności oraz zdecydowały o wykorzystaniu go do realizacji opisywanego tutaj projektu platformy mapowania ontologii.

Platforma `Eclipse RCP` oparta jest o zestaw bibliotek graficznych `SWT`[9]. Biblioteki te są znakomitą alternatywą wobec dołączonych do języka Java bibliotek `AWT` i `Swing`. Przede wszystkim, korzystają one z kontrolek graficznych dostarczanych przez system operacyjny na którym są uruchomione. Dzięki temu programy napisane z ich wykorzystaniem niczym nie różnią się w wyglądzie np. od programów napisanych w języku C++ wykorzystujących natywne rozwiązania danej platformy systemowej. Ponadto, programy te działają znacznie szybciej i zajmują mniej pamięci operacyjnej niż ich odpowiedniki wykorzystujące biblioteki `AWT` i `Swing`, gdyż odpowiednie wywołania oraz procedury związane z obsługą interfejsu graficznego realizowane są bezpośrednio przez system operacyjny a nie maszynę wirtualną Javy.

`Eclipse RCP` dostarcza bogatą infrastrukturę, która konieczna jest do realizacji zaawansowanych programów wykorzystujących graficzny interfejs użytkownika. Ponadto platforma ta wykorzystuje mechanizm *wtyczek*, który pozwala w dogodny sposób podzielić system na mniejsze elementy, które mogą być tworzone, rozwijane i wykorzystywane niezależnie. Cecha ta jest szczególnie przydatna w realizacji naszego projektu, gdyż integruje on wiele elementów, które powstają całkowicie niezależnie od siebie. Infrastruktura i mechanizmy te omówione są szerzej w następnych punktach.

---

[6] http://xdoclet.sourceforge.net/xdoclet/index.html

[7] http://www.eclipse.org/

[8] http://wiki.eclipse.org/index.php/Rich_Client_Platform

[9] http://www.eclipse.org/swt/

9.3.2. Wtyczki

Świat programów komputerowych charakteryzuje się tym, że żaden program nie powstaje w całkowitej próżni – aby szybko stworzyć system posiadający wymaganą funkcjonalność konieczne jest wykorzystanie wielu narzędzi i bibliotek programistycznych, które realizują często wykorzystywane funkcjonalności, takie jak parsowanie dokumentów XML, zabezpieczanie przed nieupoważnionym dostępem, komunikację przez Internet, itp. Wiele z nich, zyskując popularność, staje się nieformalnymi standardami.

Z drugiej jednak strony, ze względu na, charakterystyczną dla rynku oprogramowania, ciągłą dążność do ulepszenia produktów i zwiększenia ich funkcjonalności, biblioteki powszechnie wykorzystywane do realizacji typowych zadań, zmieniają się nieustannie. Co więcej – tworzą one skomplikowaną sieć zależności, gdyż niejednokrotnie system X korzysta z biblioteki Y w wersji A, natomiast system Z korzysta z tej samej biblioteki w wersji B, co nieuchronnie prowadzi do konfliktów pomiędzy różnymi fragmentami oprogramowania oraz spędza sen z powiek integratorów systemów informatycznych.

Aby, przynajmniej częściowo, rozwiązać ten problem wprowadzona została koncepcja komponentów – niezależnie powstających fragmentów oprogramowania, które poprzez zgodność z ściśle określonymi interfejsami pozwalają rozluźnić związki pomiędzy poszczególnymi elementami integrowanego systemu oraz wymieniać je niezależnie, jeśli zajdzie taka potrzeba.

W systemie `Eclipse` niezależnie powstające komponenty zwane są wtyczkami. Poprzez zastosowanie mechanizmu separacji poszczególnych wtyczek, możliwe jest tworzenie poszczególnych elementów systemu przez niezależne grupy wytwórców oraz dostosowywanie tej platformy do własnych potrzeb.

Mechanizm wtyczek wykorzystywany w `Eclipse` oparty jest o technologię wypracowaną przez *OSGi Alliance*[10]. Jej podstawowym elementem są paczki[11] – komponenty o ściśle określonych własnościach. W przeciwieństwie do Javy, która w ramach komponentu nie pozwala np. na wyspecyfikowanie prywatnych klas – zasady dostępu i zależności łączące poszczególne paczki mogą być określone z dużą precyzją. Ponadto infrastruktura ta pozwala na dynamiczne instalowanie, ładowanie, zwalnianie i usuwanie poszczególnych paczek, bez konieczności restartu maszyny wirtualnej, na której one działają. W przeciwieństwie do *web-servisów* paczki nie wymagają żadnej warstwy pośredniczącej, co sprawia, że ich wydajność jest znacznie większa. Dodatkowo – z systemem paczek zintegrowane jest system bezpieczeństwa, który przenika wszystkie jego elementy oraz zestaw serwisów, służących do konfigurowania i zarządzania paczkami.

Definiowanie zależności, ładowanie klas oraz cykl życia wtyczki określone są poprzez mechanizmy OSGi zdefiniowane dla paczek[12]. Platforma `Eclipse RCP` wzbogaca te mechanizmy poprzez *rozszerzenia*[13] oraz *punkty rozszerzeń*[14]. Wtyczka może pozwalać na rozszerzanie swojej funkcjonalności poprzez definiowanie punktów rozszerzeń oraz rozszerzać funkcjonalność innych wtyczek poprzez wykorzystywanie punktów rozszerzeń, które są w nich zdefiniowane.

[10] http://www.osgi.org/osgi_technology/index.asp
[11] ang. *bundle*
[12] Paczki i wtyczki są w platformie `Eclipse RCP` tymi samymi jednostkami.
[13] ang. *extension*
[14] ang. *extension point*

Możliwość definiowania własnych punktów rozszerzeń jest równie cenna co duża liczba już dostępnych punktów rozszerzeń. Dzięki nim można bowiem obsługiwać rozszerzenia, które tworzone są niezależnie lub też dopiero zostaną stworzone. Z cechy tej korzystamy w realizacji naszej platformy – definiowany jest jeden punkt rozszerzeń, który wykorzystywany jest do wyszukiwania i pobierania elementów przeznaczonych do zmapowania. Zagadnienie to jest omówione szczegółowo w punkcie 10.3.1.

Dwie podstawowe wtyczki, na których budowane są programy wykorzystujące platformę `Eclipse RCP`„ to: `org.eclipse.core.runtime`, zawierający podstawowe klasy zarządzające wykonaniem aplikacji oraz `org.eclipse.ui`, która definiuje podstawowe elementy interfejsu graficznego. Mogą być one wykorzystane poprzez zdefiniowanie rozszerzeń obejmujących m.in.: widoki, perspektywy i akcje, które omówione są w następnym punkcie.

9.3.3. Elementy graficznego interfejsu użytkownika

Większość bibliotek graficznych wykorzystywanych do tworzenia aplikacji z GUI zawiera zwykle zestaw typowych kontrolek, w skład którego wchodzą przyciski, etykiety, pola wprowadzania danych, etc. To co wyróżnia platformę `Eclipse RCP` na tle pozostałych systemów tego typu, to możliwość wykorzystania zintegrowanych komponentów, które mogą być definiowane w sposób deklaratywny (tzn. nie w kodzie źródłowym lecz w pliku definicji wtyczki `plugin.xml`). Pozwala to na odseparowanie wizualnych aspektów aplikacji od wewnętrznych mechanizmów jej działania oraz przyspiesza tworzenie aplikacji wykorzystujących GUI.

Do najczęściej wykorzystywanych elementów, które są niezwykle przydatne w tworzeniu takich aplikacji, należy zaliczyć: perspektywy, widoki oraz akcje.

Perspektywy

Perspektywa[15] to w terminologii `Eclipse RCP` pewna konfiguracja okien (widoków) występujących wewnątrz głównego okna programu. Pozwala ona określić wzajemne położonie oraz proporcje poszczególnych okien. Poszczególne okna występujące w ramach jednej perspektywy mogą być otwierane, zamykane, przestawiane i grupowane. Ponadto, w ramach perspektywy można określić miejsce otwarcia okien, które nie są otwierane wraz z inicjalizacją perspektywy.

W programie można zdefiniować wiele rozmaitych perspektyw, związanych z różnymi aspektami działania systemu. Np. w naszej platformie mapowania ontologii zdefiniowane są dwie perspektywy – jedna służy do wyszukiwania słów w Słowniku Semantycznym, druga natomiast służy do porównywania definicji słów występujących w Słowniku Semantycznym i konceptów występujących w ontologii Cyc oraz mapowania jednych na drugie.

Ciekawą własnością perspektyw jest to, że ich stany (tzn. pozycja i rozmiary poszczególnych okien) są zapamiętywane pomiędzy poszczególnymi uruchomieniami systemu, dzięki czemu użytkownik w łatwy sposób może dostosować je do własnych, specyficznych wymagań.

Perspektywy definiowane są jako rozszerzenia `org.eclipse.ui.perspectives`.

[15] ang. *perspective*

Widoki

Widoki[16] to w terminologii `Eclipse RCP` okienka występujące wewnątrz głównego okna programu. Ich początkowa konfiguracja jest definiowana w ramach domyślnej perspektywy. Poszczególne okienka, w zależności od własności, jakie zostały im przypisane w ramach danej perspektywy, mogą być zamykane i otwierane, maksymalizowane i minimalizowane, zawierać stowarzyszone z nimi akcje, etc.

W ramach danego okienka umieszczane są kontrolki, służące do sterowania aplikacją. Tym niemniej najczęściej stosuje się widoki do przedstawiania danych w postaci tabel lub drzew, a poszczególne akcje, które mogą być wywoływane na tych danych, umieszcza się w menu głównym, menu podręcznym lub menu danego widoku. W realizacji naszej platformy widok tabelaryczny wykorzystano np. do prezentacji wyników wyszukiwania danego słowa w Słowniku Semantycznym, a widok drzewiasty – do prezentacji relacji semantycznych opisujących dane słowo. Natomiast okienko wyszukiwania słów korzysta bezpośrednio z kontrolek `SWT`.

Widoki definiowane są jako rozszerzenia `org.eclipse.ui.views`.

Akcje

Akcje[17] to, w terminologii `Eclipse RCP`, zdarzenia wywoływane przez użytkownika w ramach jego interakcji z systemem – takie jak np. wybranie pozycji w menu lub wciśnięcie guzika. Istotną cechą akcji jest to, że kod odpowiedzialny za realizację odpowiedniego algorytmu związanego z daną akcją może być w łatwy sposób odseparowany od pozostałych jej aspektów. Przyjęto tutaj zasadę, że jedna akcja może posiadać wiele odnośników, występujących w różnych fragmentach GUI (np. może występować w menu głównym oraz jako ikona na pasku akcji). Akcje są zatem warstwą pośredniczącą pomiędzy poszczególnymi elementami GUI a odpowiednim kodem. Ponadto posiadają one wiele interesujących własności (np. możliwość związania typowych skrótów klawiszowych z odrębnymi akcjami, w zależności od perspektywy, która akurat jest wyświetlana), których nie będziemy tutaj jednak szczegółowo omawiać.

W naszej platformie akcje wykorzystywane są np. do realizacji mapowania pomiędzy słowami występującymi w Słowniku Semantycznym oraz konceptami ontologii Cyc.

Z akcjami związane są następujące rozszerzenia `org.eclipse.ui.viewActions`, `org.eclipse.ui.editorActions` oraz `org.eclipse.ui.actionsSets`.

[16] ang. *view*
[17] ang. *action*

Rozdział 10

Wspólne API

10.1. Wstęp

Aby mapowanie z wykorzystaniem platformy mapowania było skuteczne i łatwe, konieczne jest aby dane prezentowane jej użytkownikom były konsystentne. Algorytmy mapowania automatycznego również mogą ulec uproszczeniu, jeśli model danych, na których operują, będzie spójny. Takie same wymagania dotyczą modelu dostępu do danych.

Niestety, jak wynika z rozdziału 7, zarówno główne jak i pomocnicze systemy, współpracujące z platformą mapowania, posiadają zupełnie różne modele danych oraz różne modele dostępu do danych, począwszy od rozbudowanego, wielowątkowego serwera ontologii Cyc, a skończywszy na pojedynczym pliku słownika polsko-angielskiego.

Dlatego też, aby uprościć programowanie z wykorzystaniem tych systemów, konieczne było opracowanie API obejmującego wspólny model danych oraz wspólny model dostępu do danych. Bez tych modeli współpraca z poszczególnymi systemami byłaby niewątpliwie uciążliwa. Dzięki wspólnym modelom zyskujemy również, jeśli zajdzie taka potrzeba, możliwość wymiany jednego z elementów tej złożonej struktury, bez konieczności reimplementacji pozostałych elementów.

10.2. Wspólny model danych

Podstawowe typy danych z którymi mamy do czynienia w Słowniku Semantycznym oraz ontologii Cyc to: słowa/koncepty, relacje/predykaty oraz asercje. Z każdym z nich stowarzyszony jest jeden typ danych we wspólnym modelu danych. Pewne własności ontologii Cyc (np. występowanie liczb i łańcuchów w obiektach klasy `CycList`) jak i systemów pomocniczych, spowodowały, że model danych obejmuje jeszcze dwa typy danych, przydatne do reprezentacji bardziej nietypowych danych.

Wszystkie opisywane tutaj typy danych znajdują się w pakiecie `lexicon.interfaces`.

10.2.1. IPrintableElement

Typ `IPrintableElement` nie posiada żadnych wywołań, przez co może wydawać się zbędny. Jest to interfejs markujący, który pozwala rozpoznać, że mamy do czynienia z daną zaczerpniętą z jednego ze źródła danych. Dana taka po wywołaniu metody `toString()` obiektu klasy `Object` powinna zwrócić łańcuch znaków zawierający jej poprawną reprezentację. Przykładem elementu, który nie implementuje żadnego bardziej specjalizowanego interfejsu jest łańcuch znaków występujący w asercjach ontologii Cyc.

10.2.2. IEntity

Typ `IEntity` dziedziczy z typu `IPrintableElement`. Zawiera on trzy metody:
1. `String getName()` – zwraca pełną nazwę (etykietę) obiektu.
2. `String getDescription()` – zwraca opis obiektu.
3. `String getDistinctiveLabel()` – zwraca etykietę obiektu, która pozwala odróżnić go od innych obiektów posiadających tę samą nazwę (w szczególności przydatna jest do wyodrębniania homonimów).

Dzięki wprowadzeniu tego typu danych możliwe jest odróżnianie w wynikach wyszukiwania elementów, które choć posiadają tę samą nazwę, różnią się semantyką. Typ ten przydany jest również do reprezentowania obiektów, które choć nie są konceptem lub relacją w pełnym tego słowa znaczeniu, posiadają jakąś informację definicyjną.

10.2.3. IConcept

Typ `IConcept` dziedziczy z typu `IEntity`. Typ ten służy do reprezentowania słów/konceptów. Zawiera on 6 metod:
1. `IRelation[] getRelations()` – zwraca wszystkie relacje, o których wiadomo, że zachodzą pomiędzy tym konceptem, a pozostałymi konceptami.
2. `IRelation[] getRelations(int index)` – zwraca wszystkie relacje, o których wiadomo, że zachodzą pomiędzy tym konceptem, a pozostałymi konceptami, oraz, że koncept ten występuje w tych relacjach na pozycji `index`.
3. `ITuple[] getTuples()` – zwraca wszystkie krotki (asercje), w których występuje ten koncept.
4. `ITuple[] getTuples(IRelation relation)` – zwraca wszystkie krotki należące do relacji `relation`, w których występuje ten koncept.
5. `ITuple[] getTuples(int index)` – zwraca wszystkie krotki, w których koncept ten występuje na pozycji `index`.
6. `ITuple[] getTuples(IRelation relation, int index)` – zwraca wszystkie krotki należące do relacji `relation`, w których koncept ten występuje na pozycji `index`.

Dzięki wprowadzeniu wywołań z argumentem `index` możliwe, jest grupowanie poszczególnych krotek, w których występuje dany koncept, w zależności od pozycji, na której on występuje.

10.2.4. IRelation

Typ `IRelation` dziedziczy z typu `IEntity` i służy do reprezentowania relacji/predykatów. Zawiera on 2 metody:

1. `int getArity()` – zwraca arność relacji.
2. `ITuple[] getTuples()` – zwraca wszystkie krotki należące do danej relacji.

10.2.5. ITuple

Type `ITuple` dziedziczy z typu `IEntity` i służy do reprezentowania asercji. Asercje są szczególnym typem obiektów, które wyrażają informację o powiązaniach pomiędzy poszczególnymi elementami. Są one odpowiednikiem zdań twierdzących języka naturalnego lub rekordów (krotek) określonej tabeli w bazie danych. Innymi słowy są one instancjami jakiejś relacji, która zachodzi pomiędzy konceptami. W dalszej części tego dokumentu określeń „asercja", „krotka" oraz „instancja relacji" będziemy używali zamiennie.

Typ ten posiada 3 metody:
1. `IRelation getRelation()` – zwraca relację, która jest właścicielem danej krotki.
2. `IPrintableElement getElement(int index)` – zwraca element występujący na pozycji `index` w danej krotce.
3. `IPrintableElement[] getElements()` – zwraca wszystkie elementy występujące w danej krotce, w kolejności ich występowanie.

10.3. Wspólny model dostępu do danych

Komunikacja pomiędzy platformą mapowania a poszczególnymi systemami współpracującymi, odbywa się poprzez moduły pośredniczące opisane w rozdziale 11, z wykorzystaniem mechanizmu wtyczek opisanego w punkcie 9.3.2. Dlatego też konieczne było zdefiniowanie punktu rozszerzeń, w którym poszczególne, współpracujące z platformą moduły pośredniczące mogłyby rejestrować swoją obecność. Punkt rozszerzeń oraz interfejs, poprzez który następuje komunikacja, zostały opisane poniżej.

10.3.1. Punkt rozszerzeń

Platforma mapowania definiuje jeden punkt rozszerzeń: `lexicon.knowledgeSource`. Każda wtyczka chcąca z niego skorzystać musi określić trzy parametry:
1. `id` – identyfikator źródła wiedzy
2. `name` – nazwę źródła wiedzy
3. `class` – klasę implementującą interfejs `IKnowledgeSource`, poprzez który następuje komunikacja z platformą mapowania.

Dzięki wprowadzeniu tego punktu rozszerzeń możliwa jest komunikacja z elementami, które nie są znane w trakcie tworzenia platformy mapowania. Dzięki temu, integrowania niezależnych źródeł wiedzy z platformą staje się znacznie prostsze.

10.3.2. IKnowledgeSource

Interfejs `lexicon.interfaces.IKnowledgeSource` służy do komunikacji pomiędzy platformą mapowania a współpracującymi z nią systemami. Definiuje on cztery metody:
1. `IConcept[] findConcept(String conceptName, boolean caseSensitive, boolean exactPhrase)` – zwraca wszystkie koncepty, których nazwa zgodna jest z argumentem `conceptName`. Parametr `caseSensitive` określa, czy w dopasowaniu ma być uwzględniona wielkość liter, natomiast parametr `exactPhrase` określa, czy słowo kluczowe ma dokładnie odpowiadać nazwie, czy też może być jej fragmentem.

2. `IRelation[] findRelation(String relationName, boolean caseSensitive, boolean exactPhrase)` – zwraca wszystkie relacje, których nazwa zgodna jest z argumentem `relationName`. Pozostałe argumenty: j.w.
3. `IEntity[] find(String searchString, boolean caseSensitive, boolean exactPhrase)` – zwraca wszystkie koncepty i relacje, których nazwa zgodna jest z argumentem `searchString`. Pozostałe argumenty: j.w.
4. `String getName()` – zwraca nazwę źródła wiedzy.

10.4. Lexicon Plug-in

Platforma mapowania budowana jest zgodnie z modelem wtyczkowym opisanym w punkcie 9.3.2. Przedstawione w poprzednich punktach interfejsy oraz punkt rozszerzeń zgromadzone są we wtyczce `Lexicon Plug-in`, która stanowi bazę platformy mapowania.

Zdefiniowanie wspólnych modeli danych i dostępu do danych pozwoliło na zbudowanie wokół tej wtyczki prototypowego systemu umożliwiającego przeglądanie zawartości poszczególnych źródeł wiedzy. Korzystał on z wywołania `IKnowledgeSource.search` do wyszukiwania wprowadzanych terminów, `IEntity.getDistinctiveLabel` do wyświetlania listy znalezionych wyników, `IEntity.getName` do wyświetlania nazwy wybranego elementu, `IEntity.getDescription` do wyświetlania opisu wybranego elementu oraz `IConcept.getRelations` i `IConcept.getTuples` do wyświetlania relacji i krotek związanych z wybranym elementem, o ile był on konceptem.

Ten prototypowy system nie został udostępniony w ostatecznej wersji projektu, ale wiele rozwiązań wypracowanych w trakcie jego konstrukcji (np. poszczególne widoki – wyszukiwania, wyświetlania listy wyników, wyświetlania poszczególnych relacji, etc.) znalazło swoje bezpośrednie zastosowanie w ostatecznej wersji platformy, co przyczyniło się do sprawnego utworzenia pozostałych jej elementów.

Rozdział 11

Moduły pośredniczące

11.1. Wstęp

Aby zrobić użytek z mechanizmu wtyczek platformy `Eclipse RCP` opisanego w punkcie 9.3.2 oraz z wspólnego API przedstawionego w rozdziale 10 konieczne było stworzenie modułów pośredniczących w wymianie danych pomiędzy platformą mapowania a poszczególnymi systemami zewnętrznymi.

Rozmiary tych modułów różnią się znacznie. W przypadku ontologii Cyc są one niewielkie, bo i sam system ułatwia współpracę z zewnętrznymi programami. Z kolei w przypadku słownika polsko-angielskiego moduł ten przybrał znaczne rozmiary, m.in. ze względu na konieczność opracowania specjalizowanych parserów haseł słownikowych.

11.2. PSD Plug-in

Słownik Semantyczny współpracuje z platformą mapowania jako wtyczka `Polish Semantic Dictionary Plug-in`[1] i wykorzystuje jedynie dane zgromadzone w oryginalnej bazie danych. Ponadto, struktura tej bazy uległa pewnym zmianom w stosunku do oryginału, ze względu na konieczność dostosowania jej do ogólnie przyjętego modelu danych (patrz p. 10.2).

11.2.1. PSDConcept

Klasa `psd.datastructure.PSDConcept` reprezentuje słowa oraz kategorie semantyczne występujące w Słowniku Semantycznym. Na poziomie semantycznym elementy te są wyraźnie przeciwstawione, dlatego ich wspólna reprezentacja na poziomie implementacji może wydawać się błędna. Podstawową przesłanką dla tego kroku była możliwość znacznego uproszczenia implementacji, bez utraty jakiejkolwiek informacji z poziomu

[1] W skrócie `PSD Plug-in`

semantycznego. Poszczególne kategorie semantyczne mogą być bardzo łatwo odróżnione od słów, gdyż zawsze pisane są dużymi literami. Gdyby to jednak nie wystarczyło (np. gdy w słowniku będą reprezentowane skróty nazw własnych), to informacja o ich odrębności wynika z faktu, że ich identyfikatory bazodanowe są kolejnymi liczbami od 1 do 12. Jeśli ta informacja również nie pozwalałaby jednoznacznie ich zidentyfikować, to można wprowadzić dodatkową flagę, która informowałaby o ich szczególnym charakterze.

Ujednolicenie reprezentacji słów i kategorii semantycznych ma też jeden bardzo pozytywny aspekt, który istotny jest z teoretycznego punktu widzenie – poszczególne kategorie również wymagają opisu z wykorzystaniem relacji semantycznych, co będzie znacznie łatwiejsze do zaimplementowania, gdy na poziomie implementacyjnym nie będą się różnić od słów.

Klasa `psd.datastructure.PSDConcept` realizuje interfejs `lexicon.interfaces.IConcept`. Została ona rozszerzona o pewne wywołania charakterystyczne dla Słownika Semantycznego:

1. `PSDConcept getCategory()` – zwraca obiekt, który reprezentuje kategorię semantyczną, do której należy dane słowo.
2. `String getComment()` – zwraca komentarz dołączony do słowa (komentarze pozwalają szybko odróżnić homonimy, bez konieczności przeglądania wszystkich relacji, w jakie wchodzi dane słowo z innymi słowami).
3. `Word getLexeme()` – zwraca leksem z biblioteki `JLP`, który odpowiada danemu słowu. Ze względu na niejednoznaczność leksemów – użytkownik systemu musi w niektórych przypadkach przy pierwszym pobraniu wskazać właściwy leksem spośród proponowanych przez system.
4. `int getUserId()` – zwraca identyfikator użytkownika, który opracowywał dane słowo.
5. `String getWord()` – zwraca reprezentację słowa w postaci łańcucha znaków.

11.2.2. PSDRelation

Klasa `psd.datastructure.PSDRelation` reprezentuje relacje semantyczne oraz relację przynależności słowa do kategorii semantycznej. Również te elementy są wyraźnie przeciwstawione na poziomie semantycznym – powody ich utożsamienia na poziomie implementacyjnym są identyczne jak w pierwszym wypadku. Również tutaj nie dochodzi do utraty informacji, ponieważ informacja o odrębności relacji przynależności do kategorii może być w bardzo łatwy sposób reprezentowana w kodzie źródłowym i nie musi być przechowywana w bazie danych.

Klasa `PSDRelation` realizuje interfejs `lexicon.interfaces.IRelation` i również ona posiada kilka wywołań, których nie znajdujemy w tym interfejsie:

1. `String getAspect()` – zwraca aspekt relacji. W przypadku gdy dana relacja nie posiada aspektów wartością zwracaną jest `null`.
2. `String getFullName()` – zwraca pełną nazwę relacji, obejmującą również jej aspekt oraz arność[2].
3. `String getRelated()` – zwraca etykietę `RELATED_TO` o ile dana relacja jest trójargumentowa. W przeciwnym razie zwracana jest wartość `null`.

[2] Ponieważ w Słowniku Semantycznym występują relacje dwu i trójargumentowe, to trójargumentowe relacje odróżniane są poprzez etykietę `RELATED_TO`, której semantyka opisana jest w punkcie 3.4.2

11.2.3. PSDTuple

Klasa `psd.datastructure.PSDTuple` reprezentuje krotki relacji semantycznych, czyli obiekty, które zawierają informację, że dana relacja semantyczna zachodzi pomiędzy jakimiś słowami.

Realizuje ona interfejs `lexicon.interfaces.ITuple`. Ze względu na mapowanie obiektowo-relacyjne oraz występowanie relacji dwu i trójargumentowych, posiada ona dwie klasy pochodne `psd.datastructure.PSDPair` oraz `psd.datastructure.PSDTriple`, które reprezentują krotki relacji odpowiednio: dwu i trójargumentowych.

Klasa `PSDTuple` definiuje dodatkowe metody w stosunku do implementowanego interfejsu:

1. `PSDConcept getFirstArgument()` – zwraca pierwszy argument krotki.
2. `PSDConcept getSecondArgument()` – zwraca drugi argument krotki.
3. `String toDetailedString()` – zwraca szczegółową reprezentację tekstową krotki.

Klasa `PSDTriple` definiuje jeszcze jedną metodę: `PSDConcept getThirdArgument()`, która zwraca trzeci argument krotki.

11.2.4. Mapowanie obiektowo-relacyjne

Już na pierwszy rzut oka widać które klasy z pakietu `psd.datastructure` odpowiadają którym tabelom bazy danych słownika:

1. `PSDConcept` $\leftrightarrow$ `Concept`
2. `PSDRelation` $\leftrightarrow$ `V_relation`
3. `PSDTuple` $\leftrightarrow$ `Tuple`

Tym niemniej jest kilka kwestii, które wymagają dalszego omówienia. Po pierwsze: jak wspomnieliśmy wcześniej, z `PSDTuple` dziedziczą dwie klasy `PSDPair` oraz `PSDTriple`. Pierwsza z nich odpowiada wierszom tabeli `Tuple`, dla których spełniony jest warunek `concept3_id IS NULL`, natomiast druga, tym dla których spełniony jest warunek przeciwny – `concept3_id IS NOT NULL`.

Druga kwestia dotyczy powiązań pomiędzy poszczególnymi klasami. Aby móc zastosować `Hibernate`, konieczne jest ich wskazanie[3]:

1. `PSDConcept.getLexemeId()` $\rightarrow$ `Concept.lexeme_id`
2. `PSDConcept.getComment()` $\rightarrow$ `Concept.comment`
3. `PSDConcept.getWord()` $\rightarrow$ `Concept.name`
4. `PSDConcept.getUserId()` $\rightarrow$ `Concept.user_id`
5. `PSDRelation.getName()` $\rightarrow$ `V_relation.name`
6. `PSDRelation.getAspect()` $\rightarrow$ `V_relation.aspect`
7. `PSDRelation.getRelated()` $\rightarrow$ `V_relation.related`
8. `PSDTuple.getRelation()` $\rightarrow$ `Tuple.relation`
9. `PSDTuple.getFirstArgument()` $\rightarrow$ `Tuple.concept1_id`
10. `PSDTuple.getSecondArgument()` $\rightarrow$ `Tuple.concept2_id`
11. `PSDTriple.getThirdArgument()` $\rightarrow$ `Tuple.concept3_id`

Dzięki wskazaniu tych zależności nie ma konieczności pisania żmudnych zapytań `SQL`, gdyż są one generowane „w locie” przez `Hibernate`. Dodatkowo, powiązanie metody

[3] Właściwie: wskazanie zależności pomiędzy metodami dostępowymi poszczególnych klas a odpowiadającymi im polami poszczególnych tabel.

dostępowej z identyfikatorem obiektu odpowiedniej klasy sprawia, że po wywołaniu tej metody od razu otrzymujemy cały obiekt, a nie tylko jego identyfikator.

11.2.5. PSDKnowledgeSource

Dostęp do danych realizowany jest za pośrednictwem klasy `psd.db.PSDKnowledgeSource`. Klasa ta realizuje interfejs `lexicon.interfaces.IKnowledgeSource` i posiada dwa dodatkowe wywołania:

1. `PSDConcept findById(int conceptId)` – zwraca słowo którego identyfikator bazodanowy ma wartość tożsamą z wartością argumentu `conceptId`.
2. `PSDRelation[] findRelations()` – zwraca wszystkie relacje występujące w Słowniku Semantycznym.

Dzięki zastosowaniu `Hibernate`, wyszukiwanie w bazie danych poszczególnych słów, relacji czy krotek może odbywać się na wysokim poziomie abstrakcji. Zamiast operować zapytaniami `SQL` można wyszukiwać obiekty poszczególnych klas wykorzystując klasę `org.hibernate.Criteria`, która pozwala formułować różne kryteria wyszukiwania z poziomu języka obiektowego. Można również skorzystać z udostępnianego przez `Hibernate` języka `HQL`, który umożliwia nawigowanie po tabelach bazy danych w sposób identyczny jak po obiektach Javy. W ten sposób zaimplementowane są właśnie wszystkie wywołania związane z wyszukiwaniem i pobieraniem elementów występujących w Słowniku Semantycznym.

11.2.6. Rozszerzenie psd.knowledgeSource

Aby wtyczka `PSD Plug-in` mogła zostać rozpoznana przez platformę mapowania, musi wykorzystywać punkt rozszerzeń `lexicon.knowledgeSource`. Poniżej przedstawiamy parametry tego rozszerzenia określone w pliku `plugin.xml`:

1. `id` : `psd.knowledgeSource`
2. `name` : `Polish Semantic Dictionary`
3. `class` : `psd.db.PSDKnowledgeSource`

11.3. Cyc Plug-in

Współpraca ontologii Cyc z platformą mapowania zapewniona jest przez wtyczkę `Cyc Plug-in`. Wtyczka ta definiuje w pakiecie `cyc` odpowiednie klasy implementujące interfejsy wspólnego modelu danych oraz klasę pośredniczącą w wymianie danych z serwerem Cyc. Wszystkie te klasy stanowią opakowania na odpowiednie klasy definiowane w `OpenCyc API`.

11.3.1. CycPrintableElement

Klasa `CycPrintableElement`, implementująca interfejs `IPrintableElement` jest opakowaniem na klasę `CycObject`. Obiekty klasy `CycPrintableElement` delegują wywołania metody `toString()` do obiektu klasy `CycObject`, którego referencję przechowują.

11.3.2. CycEntity

Klasa `CycEntity` dziedziczy z klasy `CycPrintableElement` i implementuje interfejs `IEntity`. Jest ona opakowaniem na klasę `CycConstant`, dzięki czemu wywołania związane z identyfikacją przechowywanego obiektu (nazwa, komentarz, etc.) realizowane są poprzez odpowiednie wywołania `OpenCyc API` (np. `getDescription()` $\rightarrow$ `Access.getComme`

11.3.3. CycConcept

Klasa `CycConcept`, dziedzicząca z klasy `CycEntity` i implementująca interfejs `IConcept`, reprezentuje indywidua i kolekcje występujące w Cyc.

Najważniejsze wywołania związane z zależnościami jakie występują pomiędzy danym konceptem a instancjami relacji, w których on występuje, realizowane są w następujący sposób. W momencie, gdy wywoływana jest metoda bazująca na tych zależnościach z serwera ontologii, poprzez wywołanie funkcji `gather-index-in-any-mt` języka `SubL`, pobierane są wszystkie krotki, w których występuje dany koncept. Następnie są one sortowane w zależności od tego do jakiej relacji należą i na której pozycji występuje w nich badany koncept, a potem umieszczane w tablicach haszujących. Tak posortowane krotki pozwalają na realizację wszystkich wywołań z interfejsu `IConcept`.

Klasa `CycConcept` definiuje jeszcze jedno dodatkowe wywołanie: `CycFort get-CycConstant()`, które zwraca referencję do przechowywanego obiektu klasy `CycConstant`.

11.3.4. CycRelation

Klasa `CycRelation`, dziedzicząca z klasy `CycEntity` i implementująca interfejs `IRelation`, reprezentuje predykaty występujące w ontologii Cyc.

Implementuje ona metodę `int getArity()` poprzez wywołanie metody `getArity()` obiektu klasy `Access`. Metoda `getTuples()` nie została zaimplementowana.

11.3.5. CycTuple

Klasa `CycTuple` jest opakowaniem na klasę `CycList`, reprezentującą asercję języka `CycL`. Metoda `getRelation()` interfejsu `ITuple` zwraca pierwszy element listy, czyli predykat, którego instancją jest dana krotka, natomiast matoda `getElements()` zwraca pozostałe elementy listy, czyli poszczególne pola (argumenty) danej krotki.

Metoda `getElements()` rozpoznaje, czy obiekty występujące w obiekcie klasy `CycList` są stałymi języka `CycL`. Wtedy opakowuje je w klasę `CycConcept`. W przeciwnym razie opakowuje je w klasę `CycPrintableElement`.

11.3.6. CycKnowledgeSource

Klasa `CycKnowledgeSource`, implementująca interfejs `IKnowledgeSource`, jest opakowaniem na klasę `Access`. Poprzez utworzenie obiektu tej klasy, inicjalizuje ona połącznie z serwerem Cyc.

Poszczególne wywołania interfejsu `IKnowledgeSource` delegowane są do metod zdefiniowanych w klasie `Access` – większość z nich może być realizowana przez specyficzne metody w niej występujące. Jedynie wyszukiwanie oparte o nieścisłe dopasowanie słowa-klucza, angażuje wywołanie języka `SubL` : `constant-apropos`, które jest realizowane przez metodę `converseObject`.

Klasa `CycKnowledgeSource` zawiera również metodę `getAccess`, która zwraca obiekt klasy `Access` wykorzystywany do realizacji połączenia z ontologią Cyc.

11.3.7. Rozszerzenie cyc.knowledgeSource

Parametry rozszerzenia `lexicon.knowledgeSource` definiowane przez wtyczkę `Cyc Plug-in` są następujące:
1. `id` : `cyc.knowledgeSource`
2. `name` : `Research CYC`
3. `class` : `cyc.CycKnowledgeSource`

11.4. JLP Plug-in

11.4.1. Biblioteka JLP

Biblioteka `CLP` nie daje się bezpośrednio wykorzystać w platformie mapowania ontologii, za względu na niezgodność języków programowania. Dlatego konieczne stało się stworzenie tej biblioteki w wersji dla języka Java (zwanej `JLP`), wykorzystującej mechanizm wywołań natywnych (Java Native Interface) [7, s. 771]. W gruncie rzeczy rozwiązanie takie, dzięki wprowadzeniu abstrakcyjnych struktur danych, ukrywających przed programistą niskopoziomowe wywołania, uprościło korzystanie z biblioteki.

Implementacja biblioteki `JLP` wykorzystuje dwie podstawowe struktury danych:
1. `winnie.jlp.structures.GrammarCategory`
2. `winnie.jlp.structures.Word`

Struktura `GrammarCategory` jest typu `enum` i zawiera stałe reprezentujące w bibliotece CLP poszczególne kategorie gramatyczne [10, s. 2].

Struktura `Word` reprezentuje pojedynczy leksem należący do języka polskiego. Najważniejsze metody tej struktury opisane są poniżej:
1. `String getBaseForm()` – zwraca formę podstawową danego leksemu (np. dla rzeczownika pospolitego będzie to forma mianownika liczby pojedynczej). Jest realizowana przez wywołanie `clp_bform`.
2. `GrammarCategory getGrammarCategory()` – zwraca kategorię gramatyczną leksemu. Jest realizowana przez wywołanie `clp_class`.
3. `String[] getForms()` – zwraca wszystkie formy należące do danego leksemu.
4. `boolean isBaseForm(String form)` – sprawdza czy argument „form” jest formą podstawową danego leksemu.
5. `String getGrammarLabel()` – zwraca klasę fleksyjną danego leksemu. Klasa fleksyjna określa sposób odmiany form danego leksemu [21]. Jest realizowana przez wywołanie `clp_label`.

Biblioteka JLP reprezentowana jest przez klasę `winnie.jlp.JLPLibrary`, która posiada jedną interesującą nas metodę: `Word[] findWord(String anyForm)`. Metoda ta, dla zadanego argumentu, zwraca wszystkie leksemy, których jedna z form pokrywa się z tym argumentem. Jest ona realizowana poprzez wywołanie `clp_rec`.

Biblioteka ta jest rozprowadzana w postaci pliku `jlp-0.1.jar`. Do swojego uruchomienia wymaga zainstalowanej biblioteki `CLP` oraz biblioteki systemowej `libjlp.so`.

11.4.2. JlpEntity i JlpKnowledgeSource

Aby umożliwić gładką współpracę biblioteki JLP z platformą mapowania ontologii, stworzona została wtyczka, która pośredniczy w wymianie danych pomiędzy nimi.

Wtyczka `JLP Plug-in` definiuje dwie klasy `jlp.JlpKnowledgeSource` oraz `jlp.datastructure.JlpEntity` implementujące interfejsy odpowiednio `lexicon.interfaces.IKnowledgeSource` oraz `lexicon.interfaces.IEntity`.

Pierwsza z klas pozwala na pobieranie leksemów z biblioteki `JLP`, druga natomiast jest opakowaniem na klasę `winnie.jlp.structures.Word`. Metoda `getName()` klasy `JlpEntity` zwraca podstawową formę leksemu, `getDescritpion()` zwraca wszystkie formy leksemu, natomiast `getDistinctiveLabel()` zwraca typ gramatyczny leksemu poprzedzony jego formą podstawową. W klasie `JlpKnowledgeSource` zaimplementowana została jedynie metoda `find(String searchString, boolean caseSensitive, boolean exactMatch)`, która dla danego argumentu, w zależności od wartości argumentu `exactMatch`, zwraca wszystkie leksemy, których forma podstawowa lub jedna z form tożsama jest z argumentem `searchString`.

11.4.3. Rozszerzenie jlp.knowledgeSource

Parametry rozszerzenia `lexicon.knowledgeSource` definiowanego przez wtyczkę `JLP Plug-in` są następujące:

1. `id` : `jlp.knowledgeSource`
2. `name` : `JLP`
3. `class` : `jlp.JlpKnowledgeSource`

11.5. PWN Plug-in

Współpraca pomiędzy platformą mapowania a słownikiem polsko-angielskim Oxford/PWN realizowana jest za pośrednictwem wtyczki `PWN Plug-in`. Pomimo faktu, że w algorytmach mapowania półautomatycznego implementowanych przez platformę, wykorzystuje się tylko ułamek wiedzy zgromadzonej w tym słowniku, mianowicie jednowyrazowe angielskie odpowiedniki słów polskich i stowarzyszoną z nimi informację gramatyczną, to moduł ten przyjmuje duże rozmiary. Wynika to z konieczności przetworzenia informacji, która posiada skomplikowaną strukturę fizyczną oraz logiczną. Ponadto w trakcie realizacji tej wtyczki brano pod uwagę możliwość późniejszego wykorzystania pozostałej informacji, która znajduje się w słowniku, dlatego ona również jest przetwarzana do postaci lepiej nadającej się do wykorzystania w językach programowania wysokiego poziomu.

11.5.1. Wczytywanie słownika

Pierwszym krokiem umożliwiającym współpracę słownika polsko-angielskiego z platformą mapowania było napisanie odpowiednich procedur wczytujących i przetwarzających jego strukturę fizyczną.

Klasa `pwn.PwnDictionray` odpowiedzialna jest za wczytanie pliku `polang.win` oraz wyekstrahowanie z niego informacji o położeniu indeksu oraz poszczególnych haseł. Zadanie to realizowane jest przez metodę `readIndices`, po której wywoływana jest metoda `readEntries`. Ta druga metoda, wykorzystując informację pozyskaną przez pier-

Symbol	**Znak tekstu**
`CLOSE_PAR`	] oraz)
`CLOSING`	>
`COMMA`	,
`END`	koniec tekstu
`OPENING_C`	</
`OPENING`	<
`OPEN_PAR`	[oraz (
`SEMICOLON`	;
`TEXT`	pozostałe znaki
`WHITESPACE`	białe spacje

Tablica 11.1. Symbole skanera.

wszą metodę, tworzy klasę implementującą interfejs `pwn.IDictionaryIndex`[4], która wykorzystywana jest do wyszukiwania poszczególnych haseł. Konieczność wprowadzenia dodatkowego indeksu, poza tym udostępnianym przez słownik, wynika z faktu, że oryginalny indeks zawierał jedynie, pozbawione etykiet, offsety poszczególnych haseł.

`IDictionaryIndex` zawiera dwie metody: `addWord` oraz `findWord`. Pierwsza wykorzystywana jest w trakcie inicjalizacji słownika do wprowadzenia do indeksu informacji o poszczególnych hasłach, druga natomiast – do wyszukiwania poszczególnych słów w indeksie. W trakcie dodawania poszczególnych haseł, tworzone są obiekty klasy `pwn.datastructure.D` (implementującej interfejs `IEntity`), które zawierają etykietę hasła oraz informacje o jego fizycznym położeniu w pliku. Pozostała część hasła wczytywana jest dopiero w momencie gdy nastąpi pierwsze odwołanie do danego hasła, dzięki czemu proces inicjalizacji słownika przebiega dosyć szybko.

Klasa `DictionaryEntry` posiada dwie podstawowe metody: `getWord`, która zwraca etykietę hasła oraz `getDefinition`, która zwraca pozostałe informacje zawarte w haśle słownikowym. Informacje te udostępniane są w postaci obiektu klasy `String` i stanowią treść hasła zakodowaną w strukturze logicznej.

11.5.2. Parsowanie haseł

Aby wydobyć treść poszczególnych haseł zakodowaną w ich strukturze logicznej, konieczne jest parsowanie łańcucha zwracanego przez metodę `DictionaryEntry.getDefinition()`. Parsowanie to odbywa się na trzech poziomach: tekstowym, symbolicznym oraz logicznym.

Skaner – poziom tekstowy

Klasa `pwn.translator.Scanner` realizuje parsowanie na poziomie tekstowym, tzn. przekształca poszczególne znaki lub ciągi znaków tekstu w symbole. Symbole te, zdefiniowane w wyliczeniu `ScannerSymbol`, przedstawione są w tabeli 11.1.

[4] Teoretycznie możliwe jest wykorzystanie różnych implementacji indeksu słownika – drzewiastej i tablicowej. W praktyce poprawnie działa jedynie implementacja tablicowa, zaimplementowana w klasie `pwn.datastructure.ArrayIndex`.

Kilka białych spacji występujących obok siebie zamienianych jest na jedne symbol `WHITESPACE`, podobnie jak znaki należące do kategorii `TEXT`. Bieżący symbol pobierany jest za pomocą metody `getSymbol()`, natomiast zawartość węzła tekstowego – za pomocą metody `getContent()`.

Zastanawiające może się wydawać nieuwzględnienie cudzysłowów w tej kategoryzacji. Okazuje się jednak, że cudzysłowy występują jedynie w tagach[5] i ich wyróżnienie nie było konieczne dla właściwej realizacji parsowania.

Parser – poziom symboliczny

Klasa `pwn.translator.Parser` przekształca dane z poziomu symbolicznego do poziomu logicznego. Znaczy to, że przekształca poszczególne tagi, czyli sekwencje: `< TEXT ... >` w pojedyncze symbole. Przekształcanie całych tagów w symbole motywowane jest faktem, że z jednej strony – nie posiadają one, poza nazwą taga, informacji istotnej z logicznego punktu widzenia; z drugiej zaś – przekształcenie takie pozwala na usunięcie informacji dotyczącej graficznej reprezentacji haseł, przez co interpretacja haseł na poziomie logicznym jest ujednolicona: występujące na tym poziomie węzły tekstowe zawierają wyłącznie treść haseł (np. wartość kwalifikatora dziedzinowego), natomiast ich charakter określany jest przez towarzyszące im węzły logiczne (np. parę symboli `QUAL_O, QUAL_C`).

Parser przekształca poszczególne symbole skanera w symbole klasy `ParserSymbol`. Są one opisane w tabeli 11.2. Podobnie jak skaner, posiada on dwie metody `getSymbol()` oraz `getContent()`, których semantyka jest taka sama jak w przypadku skanera.

Translator – poziom logiczny

Klasa `pwn.translator.Translator` dokonuje właściwej interpretacji danych słownikowych na poziomie logicznym, tzn. przekształca poszczególne hasła słownikowe w odpowiednie struktury danych, które zawierają informację o powiązaniach pomiędzy słowami polskimi i angielskimi.

`Translator` jest również parserem, a do realizacji swojego zadania wykorzystuje tablicę parsingu, której fragment przedstawiony jest w tabeli 11.3[6]. Poszczególne kolumny tabeli odpowiadają poszczególnym symbolom zwracanym przez klasę `Parser`, natomiast wiersze odpowiadają stanom translatora. Każdy wpis w tablicy składa się z dwóch bajtów: starszy baj koduje akcję, która powinna zostać wykonana przez translator (np. zinterpretowanie bieżącego węzła tekstowego jako kwalifikatora dziedzinowego), a młodszy – stan, do którego powinien przejść translator, po wykonaniu akcji. Liczba 0 oznacza akcję pustą.

Translator początkowo znajduje się w stanie 0. W trakcie wykonywania parsingu pobiera on kolejne symbole z parsera, zagląda do tablicy pod adres określany przez bieżący symbol oraz swój aktualny stan i wykonuje zakodowaną tam akcję.

11.5.3. Struktury danych

Wtyczka `PWN Plug-in` definiuje dodatkowe struktury danych w stosunku do tych zaproponowanych w rozdziale 7. Struktury te lepiej nadają się do przechowywania danych

[5] Czyli sekwencjach znaków rozpoczynających się od symbolu `<` a kończących symbolem `>`.

[6] Pełna tablica parsingu znajduje się w kodzie źródłowym wtyczki `PWN Plug-in`, dlatego jej przytaczanie nie ma sensu.

Symbol	**Tag lub symbol skanera**
END	END
KEY_O	*pozostałe tagi otwierające*
KEY_C	*pozostałe tagi zamykające*
LANG_A_O	<PL>
LANG_A_C	</PL>
LANG_B_O	<GB>
LANG_B_C	</GB>
PARENTHESIS_C	CLOSE_PAR
PARENTHESIS_O	OPEN_PAR
PAR_O	<P>
PAR_C	</P>
POS_O	<I>
POS_C	</I>
QUAL_O	<SMALL>
QUAL_C	</SMALL>
SUB_GRP	SEMICOLON
TEXT	TEXT
TRANS_SP	COMMA
WHITE_SPC	WHITESPACE

Tablica 11.2. Symbole parsera.

słownika bilingwalnego, aczkolwiek w przyszłości może zostaną dostosowane do wspólnego modelu.

Wszystkie omawiane struktury znajdują się w pakiecie `pwn.datastructure`.

Klasa `SemanticEntry` reprezentuje grupę polskich homonimów, które zakwalifikowane są do tej samej kategorii gramatycznej. Ponadto, ponieważ w niektórych hasłach słownikowych grupa taka posiada dodatkowe kwalifikatory, to klasa ta pozwala przechowywać referencję do klasy `Qualification`. Obiekty tej klasy mogą również zawierać listę fraz translacyjnych reprezentowanych przez klasę `TranslationPhrase`.

Klasa `SemanticSubentry` reprezentuje jeden polski homonim należący do danej grupy homonimów. Obiekty tej klasy mogą przechowywać referencję do kwalifikatorów oraz zawierać listę obiektów klasy `Translation`, odpowiadających odrębnym synonimom angielskim.

Klasa `Translation` reprezentuje pojedyncze tłumaczenie angielskie danego słowa polskiego. Zawiera ona listę obiektów klasy `Word`, które reprezentują poszczególne słowa tego tłumaczenia – ich liczba może być większa o 1, gdyż niektóre słowa polskie nie posiadają jednowyrazowych odpowiedników angielskich.

Klasa `Word` reprezentuje pojedyncze słowo angielskie lub polskie. Poza znakową reprezentacją słowa, posiada on również stowarzyszoną z nim informację gramatyczną.

Klasa `TranslationPhrase` składa się z dwóch list, spośród których pierwsza zawiera słowa polskie, a druga – słowa angielskie. Reprezentuje ona *frazy translacyjne*, tzn.

	Symbole			
Stany	POS_O	POS_C	TEXT	...
0	0x0000	0x0000	0x0000	...
1	0x0002	0x0000	0x0000	...
2	0x0000	0x0000	0x080A	...
3	0x0000	0x0000	0x0000	...
4	0x0000	0x0000	0x090C	...
⋮	⋮	⋮	⋮	⋱

Tablica 11.3. Fragment tablicy parsingu klasy `Translator`.

występujące w słowniku wyrażenia, które posiadają dane znaczenie wyłącznie jako całość (np. kolokacje).

Klasa `Qualification` reprezentuje kwalifikatory dziedziny, zakresu oraz dodatkowe informacje stowarzyszone z poszczególnymi hasłami, pozwalające odróżnić poszczególne homonimy.

Klasa `PartOfSpeach` reprezentuje kategorie gramatyczne, które wykorzystywane są do opisu poszczególnych haseł słownikowych.

11.5.4. Dostęp do danych

Dostęp do danych realizowany jest przez klasę `pwn.PolEngDictionary`, która dziedziczy z klasy `pwn.PwnDictionary`. Implementuje ona wywołanie `search()` interfejsu `IKnowledgeSource`, które zwraca obiekt klasy `DictionaryEntry`. Jak wiemy z opisu tej klasy, zawiera ona dane słownikowe w postaci niesparsowanej, dlatego klasa `PolEngDictionary` definiuje dodatkowe wywołanie `String[] getSimpleTranslations(String searchString)`, które dla parametru `searchString`, będącego polskim słowem, zwraca wszystkie jednowyrazowe wyrażenia, będące jego angielskimi odpowiednikami.

Wywołanie to realizowane jest w sposób następujący – w indeksie lokalizowane są wszystkie hasła pasujące do parametru `searchString`. Następnie zawartość każdego znalezionego hasła podlega prasowaniu, w wyniku czego powstaje lista obiektów klasy `SemanticEntry`. Te obiekty przeglądane są w poszukiwaniu jednowyrazowych tłumaczeń, które umieszczane są w tablicy zwracanej jako wynik wywołania `getSimpleTranslations`.

11.5.5. Rozszerzenie pwn.dictionaryPlEn

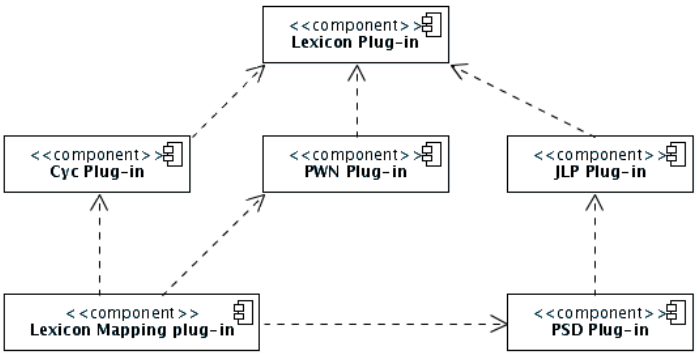
Wtyczka `PWN Plug-in` definiuje dwa rozszerzenia `lexicon.knowledgeSource`. Pierwsze z nich reprezentuje słownik polsko-angielski a drugie słownik angielsko-polski.

Słownik polsko-angielski:

1. `id` : `pwn.dictionaryPlEn`
2. `name` : `pol - eng`
3. `class` : `pwn.PolEngDictionary`

Słownik angielsko-polski:

1. `id` : `pwn.dictionaryEnPl`
2. `name` : `eng - pol`



Rysunek 11.1. Zależności występujące pomiędzy modułami platformy mapowania.

3. `class:pwn.EngPolDictionary`
   W platformie mapowania wykorzystywane jest tylko pierwsze rozszerzenie.

11.6. Zależności pomiędzy modułami

Opisane w tym rozdziale moduły[7] tworzą sieć zależności przedstawioną na rysunku 11.1.

[7] Wtyczka `Lexicon Plug-in` została opisana w rozdziale 7 natomiast `Lexicon Mapping Plug-in` w rozdziale 13.

Rozdział 12

Interfejs użytkownika

12.1. Wstęp

Ostatnim, najważniejszym modułem tworzącym platformę mapowania, jest wtyczka `Lexicon Mapping Plug-in`. Z jednej strony, zawiera ona elementy graficznego interfejsu użytkownika, które pozwalają na przeglądanie mapowanych struktur oraz ich mapowanie, z drugiej – mechanizmy i algorytmy, realizujące żądania użytkownika.

W bieżącym rozdziale opiszemy elementy GUI, które definiowane są przez wtyczkę `Lexicon Mapping Plug-in`, w następnym rozdziale zaś opisane są elementy realizujące mapowanie poprzez interakcję z modułami wymienionymi w rozdziale 11.

12.2. Perspektywy

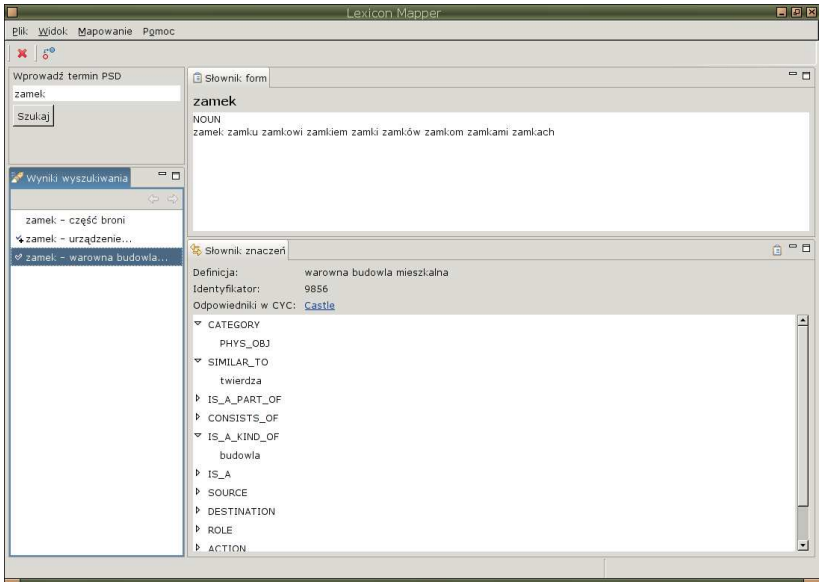
Wtyczka `Lexicon Mapping Plug-in` definiuje dwie perspektywy związane z dwoma podstawowymi funkcjonalnościami platformy mapowania: przeglądaniem Słownika Semantycznego oraz mapowaniem go na ontologię Cyc.

Do zdefiniowania perspektyw wykorzystuje się punkt rozszerzeń `org.eclipse.ui.perspectives` zdefiniowany we wtyczce `org.eclipse.ui` (patrz p. 9.3.2).

12.2.1. Przeglądanie Słownika Semantycznego

Perspektywa `lexicon.mapping.perspective.PsdPerspective` służy do przeglądania Słownika Semantycznego. Jest ona przedstawiona na rysunku 12.1 – pozwala ona wyszukiwać w nim słowa języka polskiego, przeglądać ich formy gramatyczne (poprzez wykorzystanie wtyczki `JLP Plug-in`) oraz przeglądać ich definicje wyrażone w postaci relacji semantycznych.

Poszczególne widoki i akcje związane z tą perspektywą opisane są dalej.



Rysunek 12.1. Perspektywa wyszukiwania.

12.2.2. Mapowanie

Przedstawiona na rysunku 12.2 perspektywa `lexicon.mapping.perspective.MappingPerspective`, służy do mapowania słów występujących w Słowniku Semantycznym na odpowiadające im koncepty Cyc. Dzięki swojemu podłużnemu układowi, w którego górnej części znajduje się opis wybranego słowa polskiego, a w dolnej – wybranego konceptu Cyc, pozwala porównywać te elementy przed dokonaniem mapowania.

Koncepty Cyc wyświetlone w dolnej części tej perspektywy to zaproponowane przez platformę potencjalne odpowiedniki danego słowa polskiego. Jeśli jednak użytkownik uzna, że żaden z nich nie odpowiada temu słowu, może on wprowadzić własne propozycje w okienku przeszukiwania ontologii Cyc i dokonać mapowania na koncepty, które nie zostały zaproponowane przez platformę.

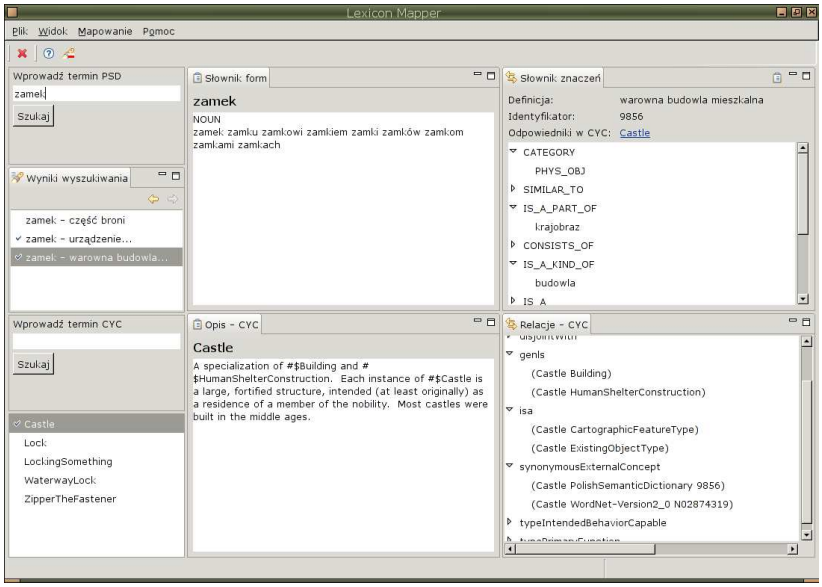
Poszczególne widoki i akcje związane z tą perspektywą opisane są dalej.

12.3. Widoki

Wtyczka `Lexicon Mapping Plug-in` definiuje szereg widoków, które umożliwiają podstawową interakcję użytkownika z platformą mapowania.

Wszystkie widoki definiowane są w ramcha punktu rozszerzeń `org.eclipse.ui.views`, przez co muszą one dziedziczyć z klasy `org.eclipse.ui.part.ViewPart`. Ponadto, ponieważ niektóre własności i funkcjonalności widoków powtarzają się (np. możliwość dynamicznej zmiany zawartości widoku), to zostały one wyabstrahowane w postaci interfejsów oraz klas ogólnego przeznaczenia, które można znaleźć w pakiecie `lexicon.views`.

Interfejs `IConfigurableView` został zdefiniowany dla widoków, które pozwalają dynamicznie zmieniać swoją zawartość (metoda `void setInput(Object intput)`) oraz pozwalają na odczytanie stanu z poprzedniego uruchomienia aplikacji (metoda `IMemento getConfiguration()`).



Rysunek 12.2. Perspektywa mapowania.

Klasa abstrakcyjna `AbstractConfigurableView` implementuje funkcjonalność tego interfejsu związaną z zachowywaniem stanu widoku pomiędzy uruchomieniami platformy. Dodatkowo, wymusza ona na swoich klasach pochodnych implementację metod:

1. `void hookContextMenu()`
2. `void hookDoubleClickAction()`
3. `void makeActions()`
4. `void contributeToActionBars()`

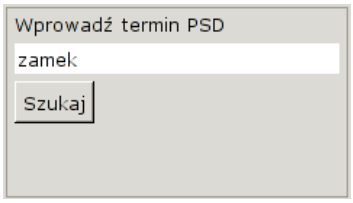
Zdefiniowanie tych metod pozwala na konsystentą obsługę podstawowych funkcjonalności związanych z widokami – obsługę podwójnego kliknięcia, menu kontekstowego oraz prywatnych akcji danego widoku.

Klasa abstrakcyjna `AbstractTableView` dziedziczy z klasy `AbstractConfigurableView` i służy do konstruowania widoków prezentujących dane w postaci listowej lub tabelarycznej. Definiuje ona akcje związane z tworzeniem widoku, reakcją na zmianę zawartości widoku, zamknięciem widoku, etc.

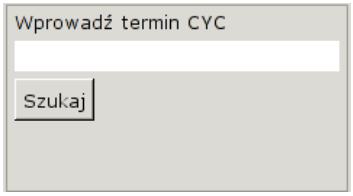
Klasa abstrakcyjna `AbstractTreeView` również dziedziczy z klasy `AbstractConfigurableView` i służy do konstrukcji widoków prezentujących dane w postaci drzewiastej. Definiuje ona akcje podobnie jak poprzednia klasa, z uwzględnieniem różnic w wykorzystywanej kontrolce graficznej.

12.3.1. Wyszukiwanie

Podstawowe funkcjonalności związane z wyszukiwaniem słów i konceptów w omawianych źródłach wiedzy realizowane są przez klasę `lexicon.views.SearchView`. Klasa ta dziedziczy z klasy `AbstractConfigurableView` i tworzy poszczególne elementy widoku wyszukiwania – etykiety, pole wprowadzania poszukiwanego słow/konceptu, przycisk wyzwalający akcję szukania, etc. Wszystkie te elementy tworzone są w metodach, która mogą być przedefiniowane przez klasy pochodne, co pozwala w łatwy sposób dos-



Rysunek 12.3. Widok wyszukiwania słów w Słowniku Semantycznym.



Rysunek 12.4. Widok wyszukiwania konceptów w ontologii Cyc.

tosować okienko wyszukiwania do indywidualnych potrzeb danego źródła informacji (np. zamiana etykiety). Klasa ta obsługuje również, opisane w punkcie 12.5, mechanizmy informujące użytkownika o postępie w wykonaniu bieżącej akcji (w tym wypadku – akcji wyszukiwania).

Wyszukiwanie w Słowniku Semantycznym realizowane jest przez widok `lexicon.mapping.views.psdSearchView`, przedstawiony na rysunku 12.3. Realizująca ten widok klasa `PsdSearchView` dziedziczy z klasy `SearchView` i w stosunku do oryginału modyfikuje parametry wyszukiwania (źródło wiedzy, sposób dopasowania), etykietę opisującą przeznaczenie tego widoku oraz sposób wyszukiwania danych[1].

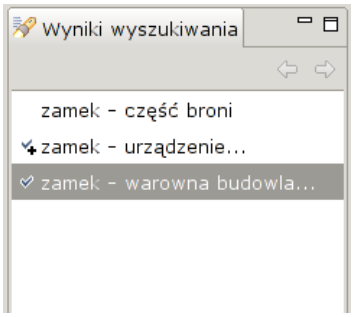
Wyszukiwanie w ontologii Cyc realizowane jest przez widok `lexicon.mapping.views.cycSearchView`, przedstawiony na rysunku 12.4. Realizująca ten widok klasa `CycSearchView` również dziedziczy z klasy `SearchView` i w stosunku do oryginału modyfikuje parametry wyszukiwania (źródło wiedzy, sposób dopasowania) oraz odpowiednią etykietę.

12.3.2. Wyniki wyszukiwania

Klasa `lexicon.views.SearchResultView` dziedziczy z klasy `AbstractTableView` i pozwala na prezentację wyników wyszukiwania danego słowa/konceptu w postaci listy wyników. Klasa ta dodatkowo definiuje funkcjonalności związane z nawigacją po wynikach wyszukiwania.

Przedstawiony na rysunku 12.5 widok `lexicon.mapping.views.psdSearchResultsView` służy do prezentacji wyników wyszukiwania danego słowa w Słowniku Semantycznym. Realizująca ten widok klasa `PsdSearchResultView` dziedziczy z klasy `SearchResultView` i w stosunku do oryginału modyfikuje nieco swoje zachowanie w

[1] W przypadku gdy poszukiwane słowo nie występuje w Słowniku Semantycznym przeszukiwana jest również biblioteka CLP/JLP z wykorzystaniem wtyczki `JLP Plug-in`. W oryginalnej klasie przeszukiwane było wyłącznie jedno źródło wiedzy.



Rysunek 12.5. Widok wyników wyszukiwania słów w Słowniku Semantycznym.

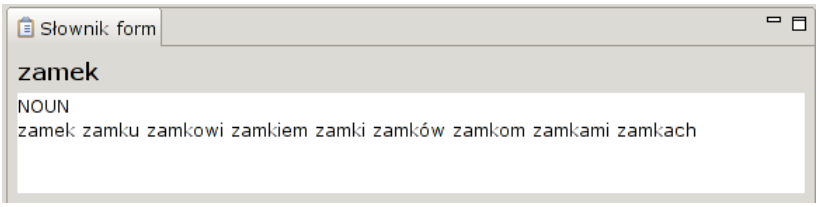
Rysunek 12.6. Widok wyników wyszukiwania konceptów w ontologii Cyc.

przypadku pojawienia się nowych wyników wyszukiwania oraz rejestruje się jako słuchacz zdarzeń związanych z wyszukiwaniem w Słowniku Semantycznym (patrz p. 12.6).

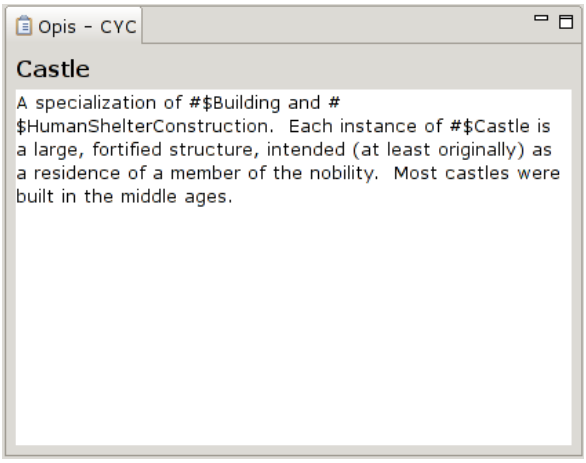
Znaczenie występujących obok haseł słownikowych symboli jest następujące – proste odfajkowanie znaczy, że słowo to zostało zmapowane do jednego konceptu ontologii Cyc, natomiast odfajkowanie z plusem, że zostało zmapowane do więcej niż jednego konceptu tej ontologii.

Przedstawiony na rysunku 12.6 widok `lexicon.mapping.views.cycSearchResultsView` służy do prezentacji wyników wyszukiwania danego konceptu czy predykatu w ontologii Cyc. Realizująca ten widok klasa `CycSearchResultView`, podobnie jak klasa `PsdSearchResultView` dziedziczy z klasy `SearchResultView` i w stosunku do oryginału modyfikuje swoje zachowanie w przypadku pojawienia się nowej listy wyników (jeśli lista jest asercją **`#$synonymousExternalConcept`** i jej trzeci argument to **`#$PolishSemanti`** słowo polskie odpowiadające identyfikatorowi występującemu w tej asercji jest wyświetlane w widokach `PsdDescription` i `PsdRelations`), dodaje akcje związane z dodawaniem i usuwaniem mapowania (omówione w punkcie 12.4) oraz rejestruje się jako słuchacz akcji związanych z wyszukiwaniem w ontologii Cyc.

Znaczenie symboli występujących w tym widoku obok konceptów Cyc, jest podobne do odpowiednich symboli w widoku Słownika. Różnią się tym, że odfajkowanie oznacza, że wybrany koncept jest zmapowany do aktualnie przeglądanego słowa Słownika Semantycznego.



Rysunek 12.7. Widoka opisu słowa – formy leksykalne.



Rysunek 12.8. Widok opisu konceptu – komentarz dołączony do konceptu w Cyc.

12.3.3. Opis słowa/konceptu

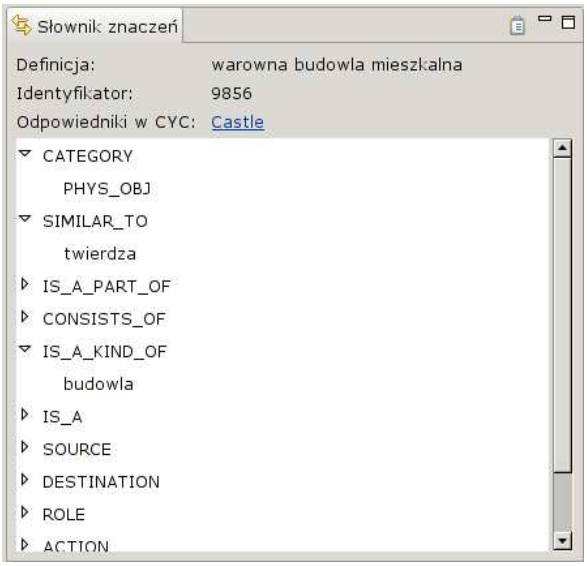
Klasa `lexicon.views.Description` dziedziczy z klasy `AbstractConfigurableView` i implementuje widok przedstawiający informację tekstową dołączoną do wybranego słowa/konceptu. Na jej podstawie stworzona została klasa `PsdDescription`, która implementuje, przedstawiony na rysunku 12.7 widok `lexicon.mapping.views.psdDescription` prezentujacy informację o formach leksykalnych danego słowa polskiego.

Natomiast klasa `CycDescription` implementuje, przedstawiony na rysunku 12.8 widok prezentujący komentarz dołączony do wybranego konceptu z ontologii Cyc.

12.3.4. Relacje słowa/konceptu

Klasa `lexicon.views.RelationView` implementuje widok, który przedstawia relacje jakie zachodzą pomiędzy wybranym obiektem a innymi obiektami występującymi w danym źródle wiedzy. Dzięki temu, że dziedziczy ona z klasy `AbstractTreeView`, relacje prezentowane są w postaci drzewiastej. Pobieranie informacji o relacjach danego słowa/konceptu realizowane jest w tle, przez co nawet gdy pobierany jest „duży” koncept wchodzący w szereg relacji, użytkownik może wykonywać inne akcje, nie czekając aż następi zakończenie przesyłania odpowiednich informacji.

Z klasy tej dziedziczy klasa `PsdRelationView`, implementująca, przedstawiony na rysunku 12.9 widok `lexicon.mapping.views.psdRealtionView`, który służy do



Rysunek 12.9. Widok relacji semantycznych w Słowniku Semantycznym.

wyświetlania informacji semantycznej związanej z danym słowem języka polskiego, a w szczególności – relacji jakie zachodzą pomiędzy nim a innymi słowami. Dwukrotne kliknięcie na słowie, które pojawia się w danej relacji, powoduje przeniesienie go do listy wyników wyszukiwania słów w Słowniku Semantycznym (widok `psdSearchResultView`). Dzięki temu możemy w prosty sposób poznać definicje innych słów występujących w Słowniku Semantycznym.

Widok ten zawiera również listę konceptów, do których zmapowane jest dane słowo, której elementy są odnośnikami, po których kliknięciu można uzyskać pełną informację na temat wybranego konceptu.

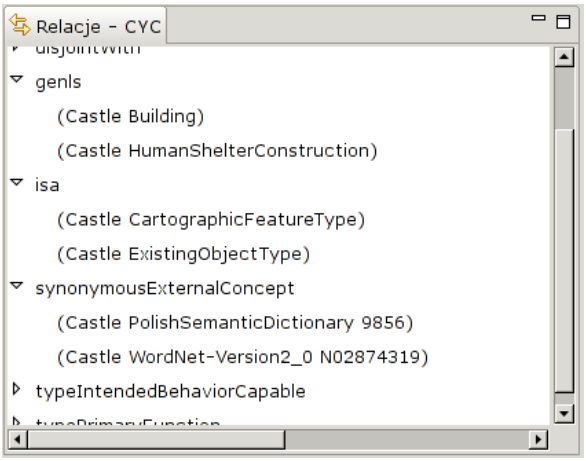
Klasa `CycRelationView` również dziedziczy z klasy `RelationView` i służy do wyświetlania asercji dotyczących danego konceptu, zgromadzonych w ontologii Cyc. Klasa ta realizuje widok `lexicon.mapping.views.cycRelationView` przedstawiony na rysunku 12.10. Poszczególne asercje zgromadzone są w węźle relacji, do której należą. Podobnie jak w przypadku widoku `psdRelationView`, kliknięcie na wybranej asercji powoduje przeniesienie jej elementów do widoku `cycSearchResultView`[2], co umożliwia ich łatwe przeglądanie.

Dodatkowo, jeśli klikniemy na asercji **`#$synonymousExternalConcept`**, zawierającej odwołanie do Słownika Semantycznego (**`#$PolishSemanticDictionary`**), to zostaną również zaktualizowane widoki `psdDescriptionView` oraz `psdRelationView`, tak by ich zawartość odpowiadała słowu polskiemu, które jest zmapowane na dany koncept Cyc.

12.3.5. Dodatkowe informacje

Klasa `PsdAdditionalRelationView` dziedziczy z klasy `RelationView` i implementuje widok `lexicon.mapping.views.psdAdditionaleRelationView`,

[2] o ile nadają się do przeniesienia – widok ten akceptuje wyłącznie obiekty implementujące interfejs `IEntity`.



Rysunek 12.10. Widok relacji w Cyc.

który przedstawiony jest na rysunku 12.11. Widok ten można wywołać obok głównego widoku relacji semantycznych – zawiera on relacje, w których wybrane słowo występuje na drugiej lub trzeciej pozycji. Innymi słowy, pozwala on zobaczyć jakie inne słowa w Słowniku wykorzystują dane słowo w swojej definicji.

12.4. Akcje

Akcje wyzwalane przez użytkownika mogą posiadać różny kontekst uruchomienia. Niektóre akcje dostępne są we wszystkich możliwych kontekstach – są to akcje globalne, takie jak np. zamknięcie programu. Nie wymagają one wykorzystania żadnego punktur rozszerzeń – wystarczy, że implementują interfejs `org.eclipse.jface.action.IAction`.

Drugi typ akcji dostępny jest tylko w wybranych perspektywach – akcje te wykorzystują punkt rozszerzeń `org.eclipse.ui.actionSets` i muszą implementować interfejs `org.eclipse.ui.IWorkbenchWindowActionDelegate`.

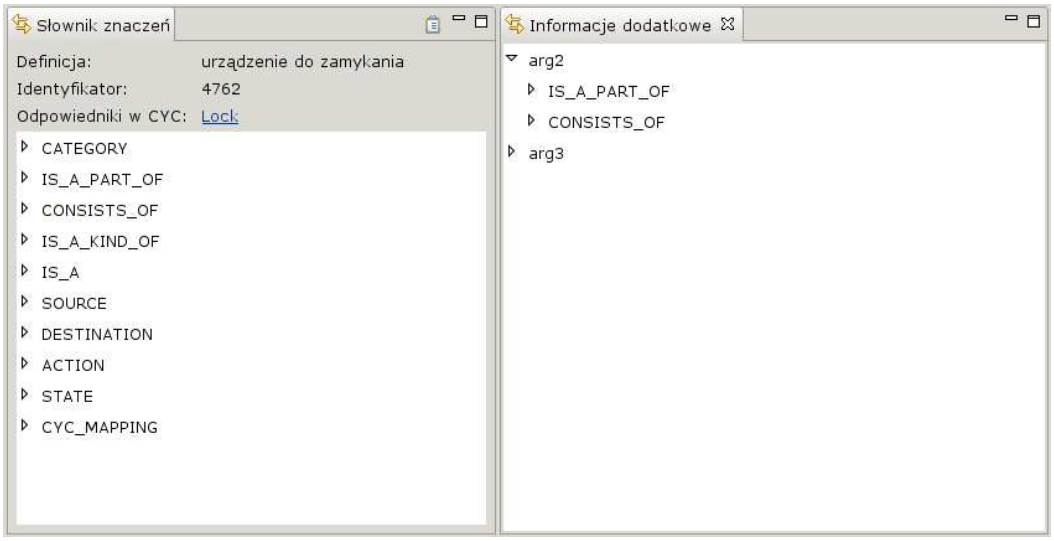
Trzeci typ akcji dostępny jest tylko dla wybranego widoku – wykorzystują one punkt rozszerzeń `org.eclipse.ui.viewActions` i muszą implementować interfejs `org.eclipse.ui.IViewActionDelagate`.

Czwarty typ akcji pojawia się w menu podręcznym jakiegoś widoku – wykorzystują one punkt rozszerzeń `org.eclipse.ui.popupMenus` i muszą implementować interfejs `org.eclipse.ui.IViewActionDelegate` lub `org.eclipse.ui.IObjectActionDelegate` (w zależności od tego, czy ich dostęp ograniczony jest do wybranych widoków czy też do wybranych obiektów, wyświetlanych w widokach).

W platformie mapowania wykorzystywane są akcje wszystkich typów. Aby uprościć implementację akcji implementujących interfejs `IWorkbenchWindowActionDelegate` stworzona została klasa abstrakcyjna `AbstractWorkbenchWindowActionDelegate`, która implementuje część funkcjonalności wymaganych przez ten interfejs.

12.4.1. Nawigowanie po historii rezultatów wyszukiwania

Nawigowanie po historii rezultatów wyszukiwania możliwe jest dzięki akcjom `lexicon.action.NextSearchResultAction` oraz `lexicon.action.PrevSearch-`



Rysunek 12.11. Widok dodatkowych relacji semantycznych w Słowniku Semantycznym.

Rysunek 12.12. Akcja: przejście do perspektywy mapowania.

`ResultAction`, które przedstawione są na rysunku 12.5 w postaci szarych strzałek. Akcje te nie wykorzystują żadnego z wcześniej wymienionych punktów rozszerzeń, gdyż tworzone są wewnątrz klasy `SearchResultView`.

Akcje te są aktywowane w zależności od zawartości historii wyników wyszukiwania. Jeśli akcja jest aktywna, to kolor strzałki zmienia się na żółty.

12.4.2. Przełączenie do perspektywy mapowania

Akcja `lexicon.mapping.action.switchToMapping` przedstawiona jest na rysunku 12.12. Wykorzystuje ona punkt rozszerzeń `actionSets`, gdyż jej dostępność ograniczona jest wyłącznie do perspektywy wyszukiwania – jeśli wybrane zostanie słowo posiadające definicję w Słowniku Semantycznym, to zostanie ona aktywowana i jej kliknięcie spowoduje przełączenie użytkownika do perspektywy mapowania, gdzie zostaną zaproponowane koncepty Cyc, na które wybrane słowo może zostać zmapowane.

Klasa `SwitchToMapping`, która implementuje tę akcję, dziedziczy z klasy `AbstractWorkbenchWindowActionDelegate`.

12.4.3. Przełączenie do perspektywy wyszukiwania

Akcja `lexicon.mapping.action.switchToPsd` przedstawiona jest na rysunku 12.13, gdyż podobnie jak w przypadku akcji `switchToMapping` jej dostępność ograniczona jest do perspektyw, w tym wypadku – mapowania. Jej kliknięcie powoduje powrót do perspektywy wyszukiwania słów w Słowniku Semantycznym. Klasa `SwitchToPsd`, która

Rysunek 12.13. Akcje: propozycje mapowania i powrót do perspektywy wyszukiwania.

implementuje tę akcję, również dziedziczy z klasy `AbstractWorkbenchWindowAction-Delegate` oraz wykorzystuje punkt rozszerzeń `actionSets`.

12.4.4. Proponowanie mapowania dla wybranego słowa

Akcja `lexicon.mapping.action.suggestMapping` podobnie jak poprzednie akcje ograniczona jest do perspektywy, w jej wypadku – mapowania. Aktywuje się ona, gdy wybrane zostanie słowo występujące w Słowniku Samentycznym, a jej kliknięcie powoduje wyświetlenie w widoku `CycSearchResultView` konceptów Cyc, na które dane słowo może zostać zmapowane. Akcja ta uruchamiana jest jest automatycznie po wybraniu słowa występującego w Słowniku Semantycznym, dlatego używanie jej ograniczone jest do przypadku, gdy użytkownik korzysta z możliwości samodzielnego wyszukiwania konceptów w Cyc i chce powrócić do konceptów zaproponowanych przez platformę mapowania.

Klasa `SuggestMapping`, tak jak klasy implementujące poprzednie akcje, dziedziczy z klasy `AbstractWorkbenchWindowActionDelegate` oraz wykorzystuje punkt rozszerzeń `actionSets`.

12.4.5. Dodawania mapowania

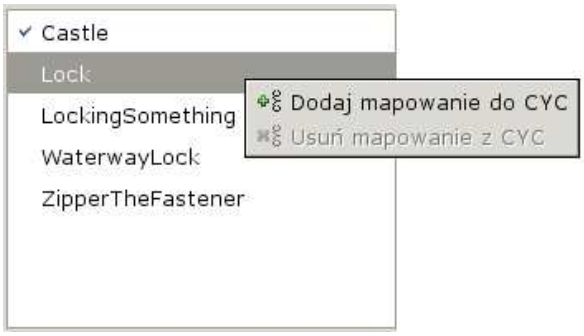
Akcja `lexicon.mapping.action.addCycMapping` przedstawiona jest na rysunku 12.14. Wykorzystywana jest ona do dodawania mapowania pomiędzy aktualnie wybranym słowem Słownika Semantycznego a wybranym konceptem Cyc. Pojawia się ona w menu podręcznym, dlatego wykorzystuje punkt rozszerzeń `viewActions`.

Klasa `AddCycMapping`, która implementuje tę akcję, dziedziczy z klasy `lexicon.-mapping.action.AbstractMappingAction`. Klasa `AbstractMappingAction` poprzez interakcję z klasą `MappingEngine` opisaną w rozdziale 13, implementuje funkcjonalności związane z dodawanie/usuwaniem mapowania, dzięki czemu klasy `AddCycMapping` oraz `RemoveCycMapping` wybierają jedynie typ akcji (dodanie lub usunięcie mapowania), a pozostałe czynności, takie jak np. poinformowanie odpowiednich widoków o zmianach w mapowaniu, realizowane są przez klasę `AbstractMappingAction`.

12.4.6. Usuwanie mapowania

Akcja `lexicon.mapping.action.removeCycMapping` przedstawiona jest na rysunku 12.15. Wykorzystywana jest ona do usuwania mapowania pomiędzy aktualnie wybranym słowem Słownika Semantycznego a wybranym konceptem Cyc. Pojawia się ona w menu podręcznym, dlatego wykorzystuje punkt rozszerzeń `viewActions`.

Klasa `RemoveCycMapping` implementująca tę akcję, również dziedziczy z klasy `Abs-tractMappingAction` i tak jak ona, wykorzystuje punkt rozszerzeń `popupMenus`.



Rysunek 12.14. Akcja: dodawanie mapowania.

Rysunek 12.15. Akcja: usuwanie mapowania.

12.4.7. Wyświetlanie/ukrywanie dodatkowych informacji o relacjach

Akcja `lexicon.mapping.action.showDetailsAction`, która widoczna jest na rysunku 12.11[3], jest przykładem akcji, która związana jest z widokiem. Pojawia się ona w belce widoku `psdRelationView` a jej kliknięci powoduje na przemian wyświetlenie i zamknięcie widoku `psdAdditionalRelationView`.

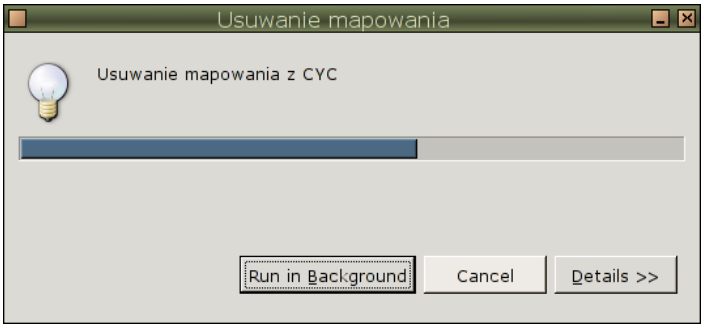
Klasa `ShowDetailsAction` wykorzystuje punkt rozszerzeń `viewActions`, dlatego też musi implementować interfejs `IViewActionDelegate`.

12.5. Wskaźniki postępu akcji

Niektóre akcje wyzwalane przez użytkownika, jak np. dodanie mapowania, trwają kilka sekund. Inne, związane z inicjalizacją poszczególnych modułów pośredniczących, również mogą przebiegać dosyć wolno. Aby użytkownik nie odnosił wrażenia, że system uległ zawieszeniu, postęp dokonywany w czasie ich trwania powinien być jakoś sygnalizowany.

Platforma `Eclipse RCP` dostarcza infrastrukturę, która pozwala raportować o postępie długo trwających akcji. Akcje takie, aby nie zablokować interfejsu użytkownika, muszą być uruchamiane w osobnym wątku. Klasa `org.eclipse.core.runtime.jobs.Job` pozwala tworzyć jednostki pracy, które mają być wywołane asynchronicznie. Przed jej wywołaniem można określić, poprzez metodę `setUser(boolean flag)`, czy taka jed-

[3] prawy górny róg widoku `psdRelationView`



Rysunek 12.16. Okienko dialogowe wskaźnika postępu.

Rysunek 12.17. Wskaźnika postępu w dolnej belce aplikacji.

nostka pracy powstała w wyniku żądania użytkownika, czy też jest związana z wewnętrznym cyklem życia aplikacji.

Akcje pierwszego typu powodują bowiem wyświetlenie okienka dialogowego przedstawionego na rysunku 12.16, które pozwala użytkownikowi na jej przerwanie. Akcje drugiego typu wyświetlane są wyłącznie w pasku stanu aplikacji (rys. 12.17), przez co nie powodują zbędnego rozproszenia użytkownika.

Druga klasa, która współtworzy infrastrukturę raportowania postępu prac to `org.eclipse.core.runtime.IProgressMonitor`. Pozwala ona nazywać poszczególne etapy wykonywanych akcji oraz określać ile pracy zostało już wykonanej.

W platformie mapowania korzystamy z tej infrastruktury, w następujących sytuacjach:

1. Otwieranie bazy Słownika Semantycznego.
2. Otwieranie słownika polsko-angielskiego.
3. Wyszukiwanie słów/konceptów w poszczególnych źródłach wiedzy.
4. Pobieranie informacji o wybranym słowie/koncepcie.
5. Dodawanie/usuwanie mapowania.
6. Wykonanie algorytmu wyboru kandydatów do mapowania.

12.6. Sterowanie przepływem danych

Wydawać by się mogło, że przepływ danych pomiędzy poszczególnymi elementami graficznego interfejsu użytkownika jest bardzo prosty – użytkownik wprowadza słowo do wyszukania, wybiera jedną pozycję z listy znalezionych wyników, przegląda jego definicję, przełącza się do perspektywy mapowania, przegląda zaproponowane przez system koncepty, wybiera jeden z nich i dokonuje mapowania.

Skoro tak, to dane wymagane na każdym kolejnym etapie mogą być przesłane do kolejnego elementu, który będzie ich potrzebował. Tym niemniej sprawa komplikuje się, jeśli uwzględnimy fakt, że np. po dodaniu mapowania należy zaktualizować informację niemal we wszystkich widokach perspektywy mapowania (aktualizacja ikon w wynikach wyszukiwania, dodanie informacji o mapowaniu do okna definicji danego słowa, etc.).

Dlatego też konieczne było rozwiązanie problemu w jaki sposób przekazywać i aktualizować dane wykorzystywane i modyfikowane symultanicznie przez różne elementy GUI. Rozwiązanie polega na wykorzystaniu mechanizmu nasłuchiwania zmian w danych. Zamiast bezpośrednio przekazywać dane z jednego elementu do innego, elementy zainteresowane zmianami w danych pewnego typu, implementują interfejs `lexicon.interfaces.InputChangeListener` i rejestrują się w centralnym punkcie kontroli przepływu danych, znajdującym się w klasie `lexicon.mapping.MappingPlugin`.

Interfejs `InputChangeListener` posiada tylko jedno wywołanie: `void inputChanged(Object input, ViewPart view)`, które służy do przekazywania danych, które uległy zmianie: `input`. Drugi parametr: `view` wskazuje na widok, który był źródłem akcji prowadzącej do zmiany danych. Może on czasem przyjmować wartość `null`, gdyż nie wszystkie akcje prowadzące do zmiany danych inicjowane są w widokach.

Klasa `MappingPlugin` posiada szereg wywołań, które pozwalają zawiadywać zmianami w danych. Z każdym rodzajem danych (np. wynikami wyszukiwania słów w Słowniku Semantycznym) stowarzyszone są trzy metody:

1. `void addXSelectionListener(InputChangeListener listener)` – służy do rejestracji elementu, który jest zainteresowany danymi rodzaju `X`.
2. `void removeXSelectionListener(InputChangeListener listener)` – służy do wyrejestrowania elementu, który nie jest już zainteresowany danymi rodzaju `X`.
3. `xChanged(Object result, ViewPart view)` – służy do poinformowania elementów zainteresowanych danymi rodzaju `X`, że uległy one zmianie.

Konieczność wprowadzenia wywołania `removeXSelectionListener` wynika z faktu, że użytkownik może zamknąć niektóre widoki (np. `psdAdditionalRelationView`) przez co przekazywanie im danych mogłoby prowadzić do wystąpienia błędu.

Dzięki temu modelowi uproszczono znacznie mechanizm sterowania przepływem danych. Każdy elementem zainteresowany danymi rodzaju `X`, w momencie swojej inicjalizacji rejestruje się wykorzystując metodę `addXSelectionListener`. Jeśli jakiś element chce przesłać do zainteresowanych nimi elementów zmienione dane typu `X`, to wykorzystuje metodę `xChanged`. Kiedy natomiast jakiś element ulega zniszczeniu (np. w wyniku zamknięcia przez użytkownika), to w trakcie wykonania wywołania `dispose()` wyrejestrowuje się z nasłuchiwania poprzez wywołanie `removeXSelectionListener`.

Rozdział 13

Implementacja mapowania

13.1. Wstęp

W niniejszym rozdziale omawiamy szczegółowo algorytmy mapowania. Są one zdefiniowane w klasie `lexicon.mapping.MappingEngine` wtyczki `Lexicon Mapping Plug-in`. Klasa pomocnicza `StringUtils` zawiera metody związane z przetwarzaniem łańcuchów znaków, w szczególności związane z obsługą polskich znaków diakrytycznych. Odpowiednie ich kodowanie jest niezbędne, gdyż ontologia Cyc akceptuje wyłącznie znaki `ASCII`.

W dalszej części rozdziału opisujemy akcje, które muszą być podjęte przez użytkownika aby zrealizować mapowanie ręczne i półautomatyczne oraz procedury klasy `MappingEngine`, które wywoływane są w ich wyniku.

13.2. API

13.2.1. MappingEngine

Klasa `lexicon.mapping.MappingEngine` posiada następujące wywołania:

1. `void addPos(CycFort wordConcept, PSDConcept psdConcept)` – dodaje w mikroteorii **`#$GeneralPolishMt`** asercję stwierdzająca, że koncept słowny[1] `wordConcept` posiada formę leksykalną odpowiadającą słowu `psdConcept`, wykorzystując predykat należący do kolekcji **`#$PartOfSpeach`** zgodny z kategorią gramatyczną tego słowa. Wywołanie to realizowane jest za pomocą metody `assertGaf` klasy `org.opencyc.api.Access`.
2. `void assertDenotation(CycFort wordConcept, PSDConcept psdConcept, CycConcept cycConcept)` – dodaje w mikroteorii **`#$GeneralPolishMt`** asercję stwierdzającą, że koncept słowny `wordConcept` posiada denotację `cycConcept`,

[1] patrz p. 4.4.5

wykorzystując predykat **#$denotation**. Drugi argument tej asercji (część mowy) określany jest na podstawie argumentu `psdConcept`. Wywołanie to realizowane jest za pomocą metody `assertGaf` klasy `Access`.

3. `void assertSynonymous(CycConcept cycConcept, PSDConcept psdConcept)` – dodaje w mikroteorii **#$PSDMappingMt** asercję stwierdzającą, że słowo `psdConcept` posiada w cyc odpowiednik `cycConcept`, wykorzystując predykat `synonymous-ExternalConcept`. Wywołanie to realizowane jest za pomocą metody `assertSynonymousExternalConcept` klasy `Access`.
4. `void createTheWord(String theWordForm)` – tworzy w mikroteorii `General-PolishMt` koncept słowny odpowiadający polskiemu słowu, którego forma `Słowo-The-Word` przekazana jest w parametrze `theWordForm`. Wywołanie to realizowane jest za pomocą metod `createIndividual` oraz `assertIsa` klasy `Access`.
5. `void getCycConcept(String conceptName)` – pobiera z ontologii Cyc koncept, którego nazwa przekazana jest w parametrze `conceptName`. Jeśli koncept taki nie występuje, zwracana jest wartość `null`. Wywołanie to realizowane jest za pomocą metody `findConcept` klasy `CycKnwoledgeSource`.
6. `List<CycEntity> getDenotations(String word, PSDConcept psdConcept)` – pobiera wszystkie denotacje polskiego słowa `psdConcept`, wykorzystując jego angielskie tłumaczenie `word`. Wywołanie to jest realizowane za pomocą metod `convers-List`, `find` oraz `isCollection` klasy `Access`.
7. `CycConcept[] getMappedConcepts(PSDConcept concept)` – zwraca wszystkie koncepty występujące w Cyc, na które zostało zmapowane polskie słowo `concept`. Wywołanie to jest realizowane za pomocą metody `queryVariable` klasy `Access`.
8. `int[] getSenseIndices(CycFort wordConcept, PSDConcept psdConcept, CycConcept cycConcept)` – zwraca indeksy sensów okupowanych przez denotację `cycConcept` w koncepcie słownym `wordConcept`. Typ gramatyczny wybierany jest na podstawie przekazanego słowa polskiego `psdConcept`. W normalnych warunkach powinien być zwrócony tylko jeden rezultat. Wywołanie to realizowane jest za pomocą metody `queryVariable` klasy `Access`.
9. `boolean hasPosForms(CycFort wordConcept)` – sprawdza czy koncept słowny `wordConcept` posiada formy leksykalne przypisane za pomocą predykatu z grupy **#$PartOfSpeach**. Wywołanie to realizowane jest za pomocą metody `hasSome-PredicateUsingTerm` klasy `Access`.
10. `boolean isMapping(CycConcept cycConcept, PSDConcept psdConcept)` – sprawdza czy słowo `psdConcept` oraz koncept `cycConcept` są zmapowane. Wywołanie to realizowane jest za pomocą metody `getMappedConcepts` klasy `MappingEngine`.
11. `void killConcept(CycFort wordConcept)` – usuwa wybrany koncept słowy z ontologii Cyc. Wywołanie to jest realizowane za pomocą metody `kill` klasy `Access`.
12. `boolean posAcceptable(PSDConcept psdConcept)` – sprawdza, czy kategoria gramatyczna słowa `psdConcept` jest obsługiwana w obecnej chwili przez platformę mapowania (nie zaimplementowano obsługi wszystkich kategorii gramatycznych).
13. `boolean posPresent(CycFort wordConcept, PSDConcept psdConcept)` – sprawdza czy koncept słowny `wordConcept` posiada przypisaną kategorię gramatyczną odpowiadającą słowu `psdConcept`. Wywołanie to jest realizowane za pomocą metody `isQueryTrue` klasy `Access`.
14. `void removeCycMapping(CycConcept cycConcept, PSDConcept psdConcept)`

– usuwa mapowanie pomiędzy słowem `psdConcept` a konceptem `cycConcept`. Wywołanie to jest realizowane za pomocą metody `unassertWithBookkeeping-AndWithoutTranscript` klasy `Access`.

15. `void unassertDenotation(CycFort wordConcept, PSDConcept psd-Concept, CycConcept cycConcept, int index)` – usuwa z mikroteorii **`#$General-PolishMt`** asercję stwierdzającą, że koncept słowny `wordConcept` posiada denotację `cycConcept` o indeksie `index`. Kategoria gramatyczna określana jest na podstawie słowa `psdConcept`. Wywołanie to jest realizowane za pomocą metody `unassert-Gaf` klasy `Access`.
16. `void unassertSynounymous(CycConcept cycConcept, PSDConcept psdConcept)` – usuwa z mikroteorii `PSDMappingMt` asercję stwierdzającą, że słowo `psdConcept` jest synonimiczne z konceptem `cycConcept`. Wywołanie to realizowane jest przez metodę `unassertGaf` klasy `Access`.

Obiekt klasy `Access` wykorzystywany w większości powyższych wywołań uzyskiwany jest przez metodę `getAccess` klasy `CycKnwoledgeSource` wtyczki `Cyc Plug-in`.

13.2.2. StringUtils

Klasa `lexicon.mapping.util.StringUtils` przeznaczona jest do konwersji łańcuchów znaków, zawierających znaki diakrytyczne, które nie są obsługiwane w ontologii Cyc. Posiada ona następujące wywołania:

1. `String getTheWordForm(String word)` – zamienia polskie słowo `word` na pozbawioną polskich znaków postać jaką przyjmuje nazwa konceptu słownego (*słowo* → `Slowo-TheWord`).
2. `String getUnicodeEscapeForm(String word)` – zamienia w słowie `word` wystąpienia wszystkich znaków charakterystycznych dla języka polskiego, na formę *unicode escaped* wykorzystywaną m.in. w kodzie źródłowym języka Java[2](*słowo* → `s\u0142owo`).
3. `String getUnicodeNormalForm(String word)` – zamienia w słowie `word` wszystkie znaki w form *unicode escaped* na normalną postać Unicode (`s\u0142owo` → *słowo*).

13.3. Mapowanie ręczne

Mapowanie ręczne, opisane w punkcie 5.2.3, realizowane jest w następujący sposób[3]:

1. Użytkownik wprowadza w widoku `psdSearchView` polskie słowo, które ma zostać zmapowane.
2. Użytkownik wybiera jeden z rezultatów wyszukiwania zaprezentowanych w widoku `psdSearchResultView`, klikając na nim dwukrotnie. Opis wybranego słowa pojawia się widokach `psdDescriptionView` oraz `psdRelationView`.
3. Użytkownik wprowadza w widoku `cycSearchView` termin Cyc, na który zamierza zmapować wybrane słowo polskie.
4. Po pojawienie się poszukiwanego terminu Cyc, użytkownik klika na nim prawym klawiszem i wybiera z menu podręcznego akcję dodania mapowania. Po zatwierdzeniu mapowa-

[2] Znaki spoza `ASCII` zamieniane są na ich reprezentacje liczbowe postaci `\uXXXX`.

[3] Opisane akcje odbywają się w perspektywie mapowania.

nia w pojawiającym się okienku dialogowym, wywoływana jest akcja `addCycMapping`, która zapisuje informację o mapowaniu w ontologii Cyc.

Akcja `addCycMapping` posiada następujący algorytm:

1. Za pomocą wywołania `posAcceptable` klasy `MappingEngine` sprawdzana jest akceptowalność kategorii gramatycznej polskiego słowa wybranego do zmapowania. Jeśli kategoria ta nie jest akceptowalna (tzn. nie jest obsługiwana w obecnej wersji platformy mapowania), proces mapowania zostaje przerwany.
2. Pobierany jest koncept słowny odpowiadajacy danemu słowu. Operacja ta przebiega w następujących etapach:
   a) Za pomocą wywołania `getTheWordForm` klasy `StringUtils`, tworzona jest postać słowa pozbawiona polskich znaków diakrytycznych.
   b) Dla tak utworzonego łańcucha wywoływana jest akcja `getCycConcept` klasy `MappingEngine`.
   c) Jeśli w wyniku wywołania poprzedniej akcji została zwrócona wartość `null`, tworzony jest nowy koncept słowny za pomocą wywołania `createTheWord` klasy `MappingEngine`.
3. Za pomocą wywołania `posPresent` klasy `MappingEngine` sprawdza się, czy uzyskany w poprzednim wywołaniu koncept słowny posiada kategorię gramatyczną odpowiadającą mapowanemu słowu polskiemu. Jeśli nie, to jest ona tworzona za pomocą wywołania `addPos` klasy `MappingEngine`.
4. Wywoływana jest metoda `assertDentotation` klasy `MappingEngine`.
5. Na koniec wywoływana jest metod `assertSynonymous` klasy `MappingEngine`.

W ten sposób informacja o mapowaniu zapisywana jest w ontologii Cyc.

13.4. Mapowanie półautomatyczne

Mapowanie półautomatyczne, opisane w punkcie 5.2.3, realizowane jest w następujący sposób[4]:

1. Użytkownik wprowadza w widoku `psdSearchView` polskie słowo, które ma zostać zmapowane.
2. Użytkownik wybiera jeden z rezultatów wyszukiwania zaprezentowanych w widoku `psdSearchResultView`, klikając na nim dwukrotnie. Opis wybranego słowa pojawia się widokach `psdDescriptionView` oraz `psdRelationView`. W tym samym momencie wywoływana jest akcja `suggestMapping`. Kandydaci do mapowania pojawiają się w widoku `cycSearchResultView`.
3. Użytkownik może przejrzeć opis każdego zaproponowanego konceptu klikając dwukrotnie na jego nazwie w widoku `cycSearchResultView`.
4. Po wybraniu konceptu odpowiadającego najlepiej polskiemu słowu, użytkownik klika na nim prawym klawiszem i wybiera z menu podręcznego akcję dodania mapowania. Po zatwierdzeniu mapowania w pojawiającym się okienku dialogowym wywoływana jest akcja `addCycMapping`, która zapisuje informację o mapowaniu w ontologii Cyc. Akcja ta ma przebieg identyczny jak w poprzednim wypadku.

   Akcja `suggestMapping` posiada następujący algorytm:

[4] Opisywane akcje wywoływane są w perspektywie mapowania, ale mogą zostać rozpoczęte w perspektywie wyszukiwania.

1. Dla wybranego słowa polskiego pobierane są jego angielskie odpowiedniki poprzez wywołanie metody `getSimpleTranslation` klasy `PolEngDictionary` z wtyczki `PWN Plugin`.
2. Dla każdego słowa, które zostało zwrócone w wyniku poprzedniego wywołania, wywoływana jest metoda `getDenotations` klasy `MappingEngine`.
3. Wyniki poszczególnych wywołań poprzedniej metody umieszczane są we obiekcie klasy `java.lang.Set` przez co powtórzenia są automatycznie eliminowane.
4. Po wywołaniu metody z p. 2. dla wszystkich słów angielskich, akcja `suggestMapping` zwraca jako swój rezultat zbiór z p. 3.

Rozdział 14

Konkluzje

Jak widzimy na przykładzie Słownika Semantycznego Języka Polskiego i ontologii Cyc, mapowanie ontologii jest zagadnieniem wysoce złożony. Problemy, które pojawiają się przy jego rozwiązaniu mają różny charakter – począwszy od elementarnych różnic w sposobie przechowywania danych, aż po różnice na poziomie semantycznym, związane z różnymi pardygmatami konstrukcji tych struktur. Platforma mapowania ontologii, która pomaga w realizacji tego skomplikowanego zadania, jest projektem pionierskim, dlatego też większa część wysiłku włożonego w jej konstrukcję skierowana była w przezwyciężenie podstawowych problemów natury technicznej. Z tego względu, rezultaty jaki w chwili obecnej można osiągnąć za jej pomocą mogą wydawać się mizerne.

Jeśli przyjrzymy się sugerowanym przez platformę konceptom, które odpowiadają słowu *pies*:

1. **`#$CanineAnimal`**
2. **`#$Dog`**
3. **`#$HotDog`**
4. **`#$Hound`**
5. **`#$Pig`**
6. **`#$Pig-Domesticated`**

to możemy odnieść wrażenie, że niektóre propozycje są zupełnie absurdalne (*vide.* **`#$Pig`**). Wynika to z faktu, że algorytm służący do wybierania konceptów jest w istocie bardzo prosty. W słowniku polsko-angielskim wybierane są jedynie jednowyrazowe odpowiedniki polskich słów, dodatkowe informacje, umożliwiające rozpoznanie poszczególnych homonimów nie są wykorzystywane, koncepty zwracane przez ontologię nie są w żaden sposób odsiewane, etc.

Tym niemniej, sam fakt, że osiągnięto współpracę pomiędzy tymi systemami jest bardzo cenny. Platforma mapowania może być rozwijana jeszcze na wiele sposobów. W szczególności, można udoskonalić mechanizmy parsingu haseł słownikowych lub w całości wymienić wykorzystywany słownik angielsko-polski na inny. Można zaimplementować znane algo-

rytmy mapowania ontologii i popatrzeć na ich wyniki. Można nawet, ze względu na modułową architekturę systemu, spróbować wykorzystać ją do mapowania Słownika Semantycznego na inną ontologię (np. SUMO) lub innego słownika semantycznego na ontologię Cyc.

Najcenniejsze zatem, wydaje się nam, rozpoznanie szczegółów struktury obu systemów, które pozwoliło na wypracowanie wspólnego modelu danych; wypracowanie ogólnej architektury modułowej, pozwalającej na dostosowanie platformy do nowych zadań; stowrzenie graficznego interfejsu użytkownika, który choć dostosowany do specyfik wykorzystywanych systemów, może w łatwy sposób zostać zaadaptowany do współpracy z innymi; na koniec zaś – rozpoznanie podstawowych problemów technicznych, które należy uwzględnić przy mapowaniu różnojęzycznych ontologii, w szczególności zaś brak odpowiedniego słownika bilingwalnego.

Dodatek A

Słownik Semantyczny Języka Polskiego

A.1. Przykładowe hasła

A.1.1. Statek

***STATEK
CATEGORY:
PHYS_OBJ, STRUCTURE
IS_A_PART_OF:
flota, eskadra, flotylla, konwój
CONSISTS_OF:
HUMAN: kapitan, porucznik, bosman, marynarz, załoga
PHYS_OBJ: kadłub, maszt, komin, torpeda, działo, uzbrojenie, rakieta, wyrzutnia
IS_A:
żaglowiec, kuter, korweta, fregata, liniowiec, karawela, karaka, galeon, szkuner, bryg, slup
ROLE:
RELATED_TO: flota, eskadra, zespół
flagowy, pomocniczy, osłaniający, osłonowy, prowadzący
ACTION:
POSITIVE:
RELATED_TO: morze, ocean, kanał, cieśnina
pływać, płynąć, sztormować, dryfować, halsować
RELATED_TO: port, zatoka, przystań
zawinąć/zawinięcie (do), wejść/wejście (do), cumować/cumowanie
RELATED_TO: brzeg
przybić/przybicie (do)

RELATED_TO: przeciwnik
zatopić/zatopienie, uszkodzić/uszkodzenie, zdobyć/zdobycie
trafić/trafienie, pokonać/pokonanie, zwyciężyć/zwycięstwo
zniszczyć/zniszczenie
RELATED_TO: desant
transportować/transport, wysadzić
NEGATIVE:
tonąć, zatonąć/zatonięcie, uszkodzić/uszkodzenie
RELATED_TO: skała, mielizna, rafa, góra lodowa
wejść/wejście (na), osiąść (na), osiadać/osiadanie (na),
uderzyć/uderzenie (w)
RELATED_TO: przeciwnik
nie trafić, chybić, spudłować/pudło, przegrać, poddać się
RELATED_TO: sztorm
dryfować/dryf, wywrócić się, do góry dnem
PASSIVE:
RELATED_TO: przeciwnik
dostać, oberwać
RELATED_TO: sztorm, burza, tajfun, tornado, szkwał
nabrać wody, przeciekać/przeciekanie/przeciek, zalewać/zalewanie
STATE:
POSITIVE:
kampania, służba
NEGATIVE:
wrak

A.1.2. Bunt

***BUNT
CATEGORY:
EVENT
SIMILAR_TO:
zamieszki, rozruchy
ACTOR:
HUMAN
OBJECT:
system, rząd, dowódca, reżim, brutalność, niesprawiedliwość
TO:
zwycięstwo, porażka
INSTRUMENT:
strajk, walka, głodówka, petycja
PLACE:
okręt, koszary, więzienie, państwo
MOOD:
gwałtowny, łagodny, pokojowy

Dodatek B

Ontologia Cyc

B.1. Przykłady haseł

Poniżej przedstawiamy niepełne opisy kolekcji **#$Dog**, indywiduum **#$NicolasCopernicus** oraz relacji **#$comment**.

B.1.1. Dog

Mt: #$UniversalVocabularyMt
#$isa : **#$BiologicalSpecies**, **#$ConventionalClassificationType**
#$genls : **#$CanisGenus**, **#$CanineAnimal**
#$comment : A BiologicalSpecies (scientific name 'Canis familiaris') that is a specialization of CanineAnimal (q.v.). Each instance of Dog is a canine animal that has either been bred to be a domestic pet (see DomesticatedAnimal) or is a wild canine animal that is not an instance of Wolf, Fox, or any other non-dog specialization of CanineAnimal. Note that although Dog and Wolf are considered distinct BiologicalSpecies, instances of the two can and do interbreed successfully. This species classification is therefore unusual, and in some circles, controversial.
Mt: #$BiologyMt
#$isa : **#$QAClarifyingCollectionType**
Mt: #$DomesticBreedsVocabularyMt
#$genls : **#$DomesticatedAnimal**
Mt: #$AnimalActivitiesMt
#$animalTypeMakesSoundType : **#$Barking**
#$conceptuallyRelated : **#$Barking**
Mt: #$ProductGMt
#$conceptuallyRelated : **#$DogFood**, **#$Doghouse**

B.1.2. NicolasCopernicus

Mt: #$UniversalVocabularyMt
#$isa : **#$Individual**
Mt: #$PeopleDataMt
#$isa : **#$MaleHuman**, **#$PolishPerson**, **#$Astronomer**, **#$HistoricHuman**
Mt: #$EnglishMt
#$familyName : "Copernicus"
#$givenName : "Nicolas"
#$nameStrings : "Nicolas Copernicus", "Copernicus"

B.1.3. comment

Mt: #$UniversalVocabularyMt
#$isa : **#$VocabularyDefiningPredicate**, **#$BinaryPredicate**,
#$DocumentationPredicate, **#$DistributingMetaKnowledgePredicate**,
#$DefaultMonotonicPredicate, **#$NonAbduciblePredicate**
Mt: #$BookkeepingMt
#$quotedIsa : **#$DefinitionalPredicate**
Mt: #$UniversalVocabularyMt
#$arity : 2
#$arg1QuotedIsa : **#$CycLIndexedTerm**
#$arg2QuotedIsa : **#$SubLString**
#$comment : A DocumentationPredicate (q.v.) that is used to relate a CycLIndexedTerm (usually a CycLConstant) to a SubLString containing an English explanation of the term' s meaning and use, as an aid to humans (whether Cyclists or not) browsing the Cyc Knowledge Base. (comment TERM STRING) means that STRING is a piece of Cyc documentation that explains the meaning and use of TERM. For example, the passage you are reading now is the comment for the CycL constant 'comment'. See also cyclistNotes.

B.2. Przykład mapowania

Poniżej przedstawiamy ręczne mapowanie słowa *pies* na koncepty występujące w ontologi Cyc.

B.2.1. Dog

(**#$genls** **#$Dog** **#$Mammal**)
(**#$genls** **#$Dushhund** **#$Dog**)
(**#$genls** **#$Poodle** **#$Dog**)
(**#$anatomicalParts** **#$Dog** **#$Head-AnimalBodyPart**)
(**#$anatomicalParts** **#$Dog** **#$Foot-AnimalBodyPart**)
(**#$anatomicalParts** **#$Dog** **#$Tail-BodyPart**)
(**#$anatomicalParts** **#$Dog** **#$Trunk-BodyCore**)
(**#$typeIntendedBehaviorCapable** **#$Dog** RescuingSomeone **#$causalActor**)
(**#$typeIntendedBehaviorCapable** **#$Dog** WalkingAPet **#$accompaniedBy**)
(**#$typeBehaviorCapable-PerformedBy** **#$Dog** Running)

(**#$typeBehaviorCapable-PerformedBy #$Dog** BarkingSound)
(**#$typeBehaviorCapable-PerformedBy #$Dog** Lying-Physical)
(**#$typeBehaviorCapable-PerformedBy #$Dog** ScratchingOneself)
(**#$feelsEmotion** (**#$SomeExampleFn #$Dog**) **#$Courageousness-Feeling**)
(**#$feelsEmotion** (**#$SomeExampleFn #$Dog**) **#$Fear**)

Dodatek C

Słownik polsko-angielski Oxford/PWN

C.1. Przykładowe hasła

Poniżej przedstawiamy przykładowe hasła zaczerpnięte ze Słownika polsko-angielskiego Oxford/PWN. Pierwsza część każdego hasła przedstawia informację, która dostępna jest bezpośrednio w słowniku, druga – graficzną reprezentację hasła.

C.1.1. admirał

```
<BIG><B><PL>admira|ł</PL></B></BIG>
<P> <SUB><IMG SRC="rzym1.jpg"></SUB> <I>m pers.</I>
  <PL><B> (<I>Npl</I> admirałowie)</B></PL>
    <TEXTSECTION ID="1"><SMALL> Wojsk.</SMALL><TEXTSECTION ID="0">
    <PL><TEXTSECTION ID="1"><SMALL> (osoba, stopień, tytuł)</SMALL>
      <TEXTSECTION ID="0"></PL><GB> admiral</GB>;
  <PL><B> admirał floty</B></PL>
    <GB> Admiral of the Fleet
      <TEXTSECTION ID="1"><SMALL>GB</SMALL><TEXTSECTION ID="0">,
      Fleet Admiral
      <TEXTSECTION ID="1"><SMALL>US</SMALL><TEXTSECTION ID="0"></GB>
</P><P>
<P></P><SUB><IMG SRC="rzym2.jpg"></SUB> <I>m anim.</I>
  <TEXTSECTION ID="1"><SMALL> Zool.</SMALL><TEXTSECTION ID="0">
    <PL><TEXTSECTION ID="1"><SMALL> (motyl)</SMALL>
      <TEXTSECTION ID="0"></PL><GB> admiral</GB>
</P>
```

admira|ł
I *m pers.* **(*Npl* admirałowie)** Wojsk. (osoba, stopień, tytuł) admiral;

admirał floty Admiral of the Fleet GB, Fleet Admiral US
II *m anim.* Zool. (motyl) admiral

C.1.2. afro

```
<BIG><B><PL>afro</PL></B></BIG><TEXTSECTION ID="1"><SMALL> Moda</SMALL>
<TEXTSECTION ID="0">
<P> <SUB><IMG SRC="rzym1.jpg"></SUB> <I>adi. inv.</I>
  <PL><I> [fryzura, włosy, peruka]</I></PL><GB> Afro</GB>
</P><P><P></P><SUB><IMG SRC="rzym2.jpg"></SUB> <I>n inv.</I>
  <GB> Afro</GB>
</P>
```

afro Moda
I *adi. inv. [fryzura, włosy, peruka]* Afro
II *n inv.* Afro

C.1.3. amerykanizować

Słowo „amerykanizować” występuje w słowniku jako dwa hasła, przy czym różnica dotyczy przedrostka „z”.

amerykanizować

```
<BIG><B><PL>amerykaniz|ować</PL></B></BIG><I> impf</I>
<P> <SUB><IMG SRC="rzym1.jpg"></SUB> <I>vt</I>
  <GB> to Americanize</GB>;
  <PL><B> młodzi przedsiębiorcy amerykanizują stosunki
    i styl pracy w swoich firmach</B></PL>
    <GB> young entrepreneurs are adopting Americanized
    working methods in their firms</GB>
    <PL> <A HREF="72,48793"><B>&rArr; zamerykanizować</B></A></PL>
</P>
<P><P></P><SUB><IMG SRC="rzym2.jpg"></SUB> <I>
  <PL><B>amerykanizować się</B></PL></I>
    <GB> to become Americanized</GB>;
  <PL><B> kultura europejska niepokojąco się amerykanizuje</B></PL>
    <GB> European culture is becoming alarmingly Americanized</GB>
  <PL> <A HREF="72,48793"><B>&rArr; zamerykanizować się</B></A></PL>
</P>
```

amerykaniz|ować *impf*
I *vt* to Americanize;
młodzi przedsiębiorcy amerykanizują stosunki i styl pracy w swoich firmach
young entrepreneurs are adopting Americanized working methods in their firms
→ **zamerykanizować**
II ***amerykanizować się*** to become Americanized;
kultura europejska niepokojąco się amerykanizuje
European culture is becoming alarmingly Americanized
→ **zamerykanizować się**

zamerykanizować

```
<BIG><B><PL>zamerykaniz|ować</PL></B></BIG><I> pf</I>
<P>
  <SUB><IMG SRC="rzym1.jpg"></SUB> <I>vt</I>
    <GB> to Americanize</GB><PL><I> [przemysł filmowy, handel]</I></PL>
    <PL> <A HREF="72,771"><B>&rArr; amerykanizować</B></A></PL>
</P>
<P><P></P>
  <SUB><IMG SRC="rzym2.jpg"></SUB>
    <I><PL><B>zamerykanizować się</B></PL></I>
    <GB> to become Americanized</GB>
    <PL> <A HREF="72,771"><B>&rArr; amerykanizować się</B></A></PL>
</P>
```

zamerykaniz|ować *pf*
I *vt* to Americanize *[przemysł filmowy, handel]* → **amerykanizować**
II ***zamerykanizować się*** to become Americanized → **amerykanizować się**

C.1.4. amortyzować

```
<BIG><B><PL>amortyz|ować</PL></B></BIG><I> impf</I>
<P> <SUB><IMG SRC="rzym1.jpg"></SUB> <I>vt</I>
  <P><B>1. <HANGINGPAR></B>
    <TEXTSECTION ID="1"><SMALL> Ekon.</SMALL>
    <TEXTSECTION ID="0">
      <GB> to amortize</GB>,<GB> to write off</GB>
        <PL><I> [koszty]</I></PL>;
      <PL><B> maszyna amortyzuje koszty jej zakupu</B></PL>
        <GB> the machine pays for itself</GB>
      <PL> <A HREF="72,48861"><B>&rArr; zamortyzować</B></A></PL>
  </P>
  <P> <B>2. <HANGINGPAR></B>
    <TEXTSECTION ID="1"><SMALL> Techn.</SMALL>
    <TEXTSECTION ID="0">
      <GB> to cushion</GB><PL><I> [wstrząs, uderzenie]</I></PL>;
      <GB> to break</GB><PL><I> [upadek]</I></PL>
      <PL> <A HREF="72,48861"><B>&rArr; zamortyzować</B></A></PL>
  </P>
</P>
<P><P></P><SUB><IMG SRC="rzym2.jpg"></SUB>
  <I><PL><B>amortyzować się</B></PL></I>
    <TEXTSECTION ID="1"><SMALL> Ekon.</SMALL>
    <TEXTSECTION ID="0">
      <PL><I> [maszyna, urządzenie]</I></PL>
        <GB> to pay for itself</GB>
      <PL> <A HREF="72,48861"><B>&rArr; zamortyzować się</B></A></PL>
</P>
```

amortyz|ować *impf*
I *vt*

1. Ekon. to amortize, to write off *[koszty]*;
   maszyna amortyzuje koszty jej zakupu the machine pays for itself
   → **zamortyzować**
2. Techn. to cushion *[wstrząs, uderzenie]*;
   to break *[upadek]*
   → **zamortyzować**

II *amortyzować się* Ekon. *[maszyna, urządzenie]* to pay for itself
→ **zamortyzować się**

C.1.5. zamek

```
<BIG><B><PL>zam|ek</PL></B></BIG> <I>m</I>
<P> <B>1. <HANGINGPAR></B>
  <PL><TEXTSECTION ID="1"><SMALL> (do zamykania)</SMALL>
    <TEXTSECTION ID="0"></PL><GB> lock</GB>;
  <PL><B> zamek szyfrowy</B></PL><GB> a combination lock</GB>;
  <PL><B> zamek cyfrowy</B></PL><GB> a digital lock</GB>;
  <PL><B> zamek z zapadką sprężynową</B></PL><GB> a spring lock</GB>;
  <PL><B> centralny zamek</B></PL><GB> central locking</GB>
</P>
<P> <B>2. <HANGINGPAR></B>
  <PL><TEXTSECTION ID="1"><SMALL> (w broni palnej)</SMALL>
    <TEXTSECTION ID="0"></PL><GB> lock</GB>;
  <PL><B> zamek karabinu</B></PL><GB> the lock of a rifle</GB>;
  <PL><B> zamek skałkowy</B></PL><GB> a flintlock</GB>
</P>
<P> <B>3. <HANGINGPAR></B>
  <PL><B> (<I>G</I> zamku)</B></PL><PL><TEXTSECTION ID="1">
    <SMALL> (budowla)</SMALL> <TEXTSECTION ID="0"></PL><GB> castle</GB>;
  <PL><B> zamek warowny</B></PL><GB> a fortified castle</GB>
</P>
<P></P>&square;
<PL><B> zamek błyskawiczny</B></PL>
  <GB> zip (fastener)</GB><TEXTSECTION ID="1"><SMALL> GB</SMALL>
    <TEXTSECTION ID="0">,
  <GB> zipper</GB><TEXTSECTION ID="1"><SMALL> US</SMALL>
    <TEXTSECTION ID="0">
<P><P></P>&squareb;
  <PL><B>stawiać </B><TEXTSECTION ID="1"><SMALL>a.</SMALL>
    <TEXTSECTION ID="0"><B> budować zamki na lodzie</B></PL>
    <GB> to build castles in the air</GB>
```

zam|ek *m*

1. (do zamykania) lock;
   zamek szyfrowy a combination lock;
   zamek cyfrowy a digital lock;
   zamek z zapadką sprężynową a spring lock;
   centralny zamek central locking

2. (w broni palnej) lock;
   zamek karabinu the lock of a rifle;
   zamek skałkowy a flintlock
3. **(*G* zamku)** (budowla) castle;
   zamek warowny a fortified castle

- **zamek błyskawiczny** zip (fastener) GB, zipper US
- **stawiać** a. **budować zamki na lodzie** to build castles in the air

C.1.6. zamęczyć

Słowo „zamęczyć” występuje w słowniku jako dwa odrębne hasła. Forma „zamęczać” również występuje jako oddzielne hasło, które odsyła do odpowiedniej definicji.

zamęczać

```
<BIG><B><PL>zamęczać</PL></B></BIG><I> impf</I>
<PL> <A HREF="72,48795"><B>&rarr; zamęczyć<SUP>1</SUP></B></A></PL>
```

zamęczać *impf* → **zamęczyć**[1]

zamęczyć[1]

```
<BIG><B><PL>zamęcz|yć<SUP>1</SUP></PL></B></BIG><I> pf</I>
<PL><BIG><B> &pause; zamęczać</B></BIG></PL><I> impf</I>
<P> <SUB><IMG SRC="rzym1.jpg"></SUB> <I>vt</I>
  <PL><TEXTSECTION ID="1"><SMALL> (doprowadzić do wyczerpania)</SMALL>
    <TEXTSECTION ID="0"></PL>
    <GB> to torment</GB>,<GB> to exhaust</GB>;
  <PL><B> zamęcza mnie, żebym poszedł do fryzjera</B></PL>
    <GB> s/he&rsquo;s always on at me to get my hair cut</GB>
</P>
<P><P></P><SUB><IMG SRC="rzym2.jpg"></SUB>
  <I><PL><B>zamęczyć się &pause; zamęczać się</B></PL></I>
    <GB> to exhaust oneself</GB>,
    <GB> to overstrain</GB>,
    <GB> to tire oneself out</GB>;
  <PL><B> zamęczał się pracą przez całe życie</B></PL>
    <GB> he slaved away all his life</GB>;
  <PL><B> nie pracuj tyle, bo się zamęczysz!</B></PL>
    <GB> don&rsquo;t work so much, you will tire yourself out</GB>;
  <PL><B> zamęczał się zbyt ciężką pracą</B></PL>
    <GB> he exhausted himself with too much work</GB>
</P>
```

zamęcz|yć[1] *pf* — **zamęczać** *impf*

I *vt* (doprowadzić do wyczerpania) to torment, to exhaust;
zamęcza mnie, żebym poszedł do fryzjera
s/he’s always on at me to get my hair cut

II *zamęczyć się — zamęczać się*
to exhaust oneself, to overstrain, to tire oneself out;
zamęczał się pracą przez całe życie

he slaved away all his life;
nie pracuj tyle, bo się zamęczysz!
don’t work so much, you will tire yourself out;
zamęczał się zbyt ciężką pracą
he exhausted himself with too much work

zamęczyć[2]

```
<BIG><B><PL>zamęcz|yć<SUP>2</SUP></PL></B></BIG><I> pf</I> <I>vt</I>
  <GB> to torture [sb] to death</GB>,<GB> to torment [sb] to death</GB>
```

zamęcz|yć[2] *pf vt* to torture [sb] to death, to torment [sb] to death

Dodatek D

Podręcznik użytkownika

D.1. Instalacja

D.1.1. Uwagi wstępne

Platforma mapowania ontologii napisana jest w języku `Java`, dlatego też może działać na dowolnym systemie operacyjnym, na który dostępny jest ten język. Tym niemniej, ze względu na systemy zewnętrzne z którymi współpracuje, w chwili obecnej dostępna jest ona wyłącznie dla systemu operacyjnego `Linux`. Jest to spowodowane brakiem implementacji biblioteki `CLP` na inne platformy systemowe.

D.1.2. Systemy zewnętrzne

Platforma mapowania ontologii do swojego działania wymaga następujących systemów zewnętrznych:

1. Java v. 5.0 SE
2. Research Cyc v. 0.9
3. Biblioteka CLP/JLP
4. Multimedialny Słownik polsko-angielski Oxford/PWN.
5. Słownik Semantyczny Języka Polskiego.

Java

`Java` dostępna jest na stronie firmy Sun: `http://java.sun.com`, na której znajdują się również instrukcje jej instalacji.

Research Cyc

Aby móc skorzystać z ontologii `Research Cyc`, konieczne jest wystąpienie o licencję na jej użytkowanie. Odpowiedni formularz znajduje się na stronie: `http://research.-cyc.com`, tam również znajdują się instrukcje instalacji.

Biblioteka CLP/JLP

Podobnie jak ontologia `Cyc`, biblioteka `CLP` nie może być swobodnie rozpowszechniana. Jej użytkowanie wymaga odpowiedniego pozwolenia, o które należy postarać się we własnym zakresie.

Plik `libjlp.so`, który niezbędny jest do współpracy platformy mapowania z biblioteką `CLP`, powinien zostać skopiowany z katalogu `clp` znajdującego się na płycie instalacyjnej, do katalogu `/usr/lib/` poleceniem:

```
$ cp libjlp.so /usr/lib
```

Następnie powinien zostać zarejestrowany jako biblioteka systemowa poleceniem:

```
$ ld /usr/lib/libjlp.so
```

Słownik polsko-angielski

Słownik Multimedialny polsko-angielski Oxford/PWN jest programem, którego wykorzystanie wymaga zakupu odpowiedniej licencji, dlatego pliki wykorzystywane w algorytmie mapowania nie mogą być rozpowszechniane razem z platformą mapowania. Aby móc skorzystać z tego słownika, konieczny jest jego zakup oraz instalacja w systemie operacyjnym © Windows.

Po zainstalowaniu słownika, należy skopiować pliki `polang.win` oraz `angpol.win`, które znajdują się zazwyczaj w katalogu `c:\Program Files\PWN\WSPWNOUP2004\Data` do katalogu `lexicon/plugins/pwn_1.0.0/data/`.

Słownik Semantyczny Języka Polskiego

Słownik Semantyczny Języka Polskiego dostarczony jest na płycie instalacyjnej i kopiowany jest w trakcie procesu instalacji platformy, dlatego nie wymaga on osobnej instalacji.

D.1.3. Platforma mapowania

Aby zainstalować *Platformę mapowania ontologii* należy z płyty instalacyjnej skopiować plik `lexicon.tar.gz`, do odpowiedniego katalogu:

```
$ cp lexicon.tar.gz katalog/docelowy
```

Następnie po przejściu do katalogu docelowego wydać polecenie:

```
$ tar -xzf lexicon.tar.gz
```

Jeśli zakończy się ono powodzeniem, to platforma mapowania jest zainstalowana na komputerze.

Aby uruchomić platformę mapowania, należy wydać polecenie:

```
$ ./lexicon
```

w katalogu, w którym została ona zainstalowana.

D.2. Praca z platformą mapowania

D.2.1. Podstawowe koncepcje

Platforma mapowania jest programem służącym do mapowania Słownika Semantycznego Języka Polskiego na ontologię Cyc. Program ten działa w dwóch trybach – przeglą-

dania oraz mapowania. Pierwszy tryb dogodniejszy jest do przeglądania zawartości Słownika Semantycznego, aczkolwiek w trybie drugim jest ono również możliwe.

Aby umożliwić śledzenie opisywanych tutaj elementów graficznego interfejsu użytkownika, najlepiej zacząć od uruchomienia aplikacji, poprzez wydanie komendy[1]:

```
$ ./lexicon
```

w głównym katalogu aplikacji. Po chwili na ekranie powinno pojawić się główne okno aplikacji przedstawione na rysunku D.1.

Z każdym z trybów pracy aplikacji związana jest *perspektywa*, czyli układ okienek aplikacji, które pojawiają się wewnątrz okna głównego.

Perspektywa wyszukiwania

Perspektywa wyszukiwania, które pojawia się po uruchomieniu systemu, składa się z następujących elementów[2]:

1. Menu głównego.
2. Paska akcji.
3. Okna wyszukiwania słów w Słowniku Semantycznym (PSD).
4. Okna wyników wyszukiwania słów.
5. Okna form leksykalnych wybranego słowa.
6. Okna definicji semantycznej wybranego słowa.
7. Paska stanu aplikacji.

Perspektywa mapowania

Perspektywa mapowania, do której można się przełączyć po wybraniu słowa, które posiada definicję w Słowniku Semantycznym, przedstawiona jest na rysunku D.2. W stosunku do perspektywy wyszukiwania posiada on 4 dodatkowe elementy:

1. Okno wyszukiwania konceptów w ontologii Cyc.
2. Okno wyników wyszukiwania konceptów.
3. Okno komentarza dołączonego do wybranego konceptu.
4. Okno asercji, w których występuje wybrany koncept.

D.2.2. Przeglądanie zawartości Słownika Semantycznego

Wyszukiwanie

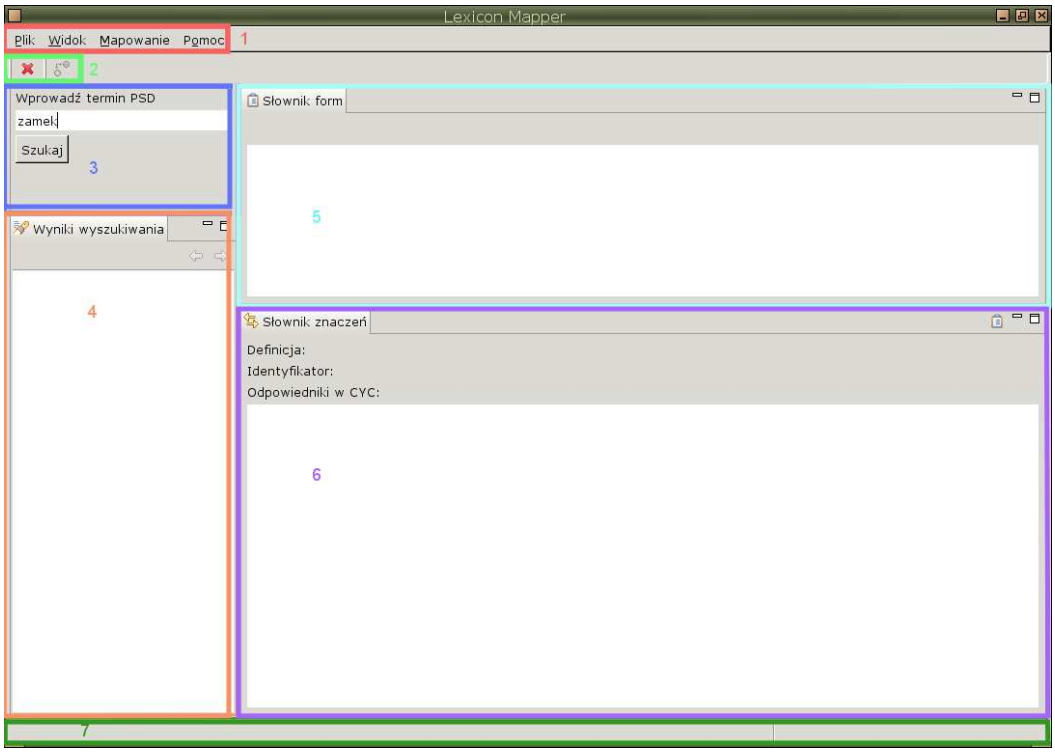
Do przeglądania zawartości Słownika Semantycznego najlepiej jest skorzystać z perspektywy wyszukiwania. Interesujące nas słowo wpisujemy do okna wyszukiwania, tak jak jest to przedstawione na rysunku D.3 i naciskamy klawisz `Enter` lub klikamy przycisk `Szukaj`.

Wyniki wyszukiwania

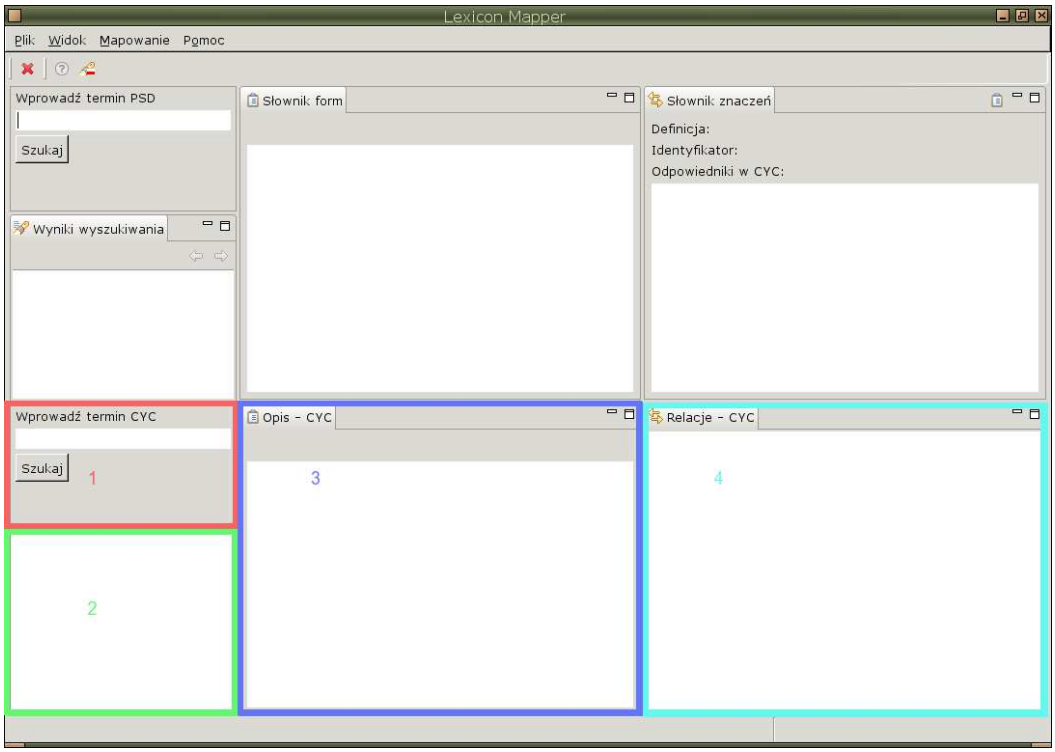
Wyniki wyszukiwania pojawiają się w oknie *Wyniki wyszukiwania*, o ile poszukiwane słowo występuje w Słowniku Semantyczny (patrz rysunek D.4). W przeciwnym razie użytkownik powiadamiany jest, że słowo to nie występuje w tym słownik. W tej sytuacji słowa poszukuje

[1] W systemie Linux.

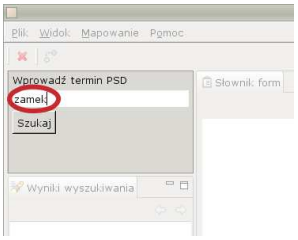
[2] Poszczególne elementy wyróżnione są na rysunku D.1 różnymi kolorami



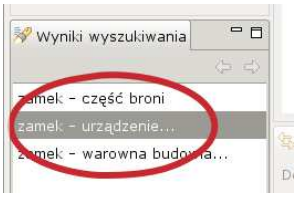
Rysunek D.1. Główne okno platformy mapowania – perspektywa wyszukiwania.



Rysunek D.2. Główne okno platformy mapowania – perspektywa mapowania



Rysunek D.3. Wprowadzanie słowa do wyszukania.

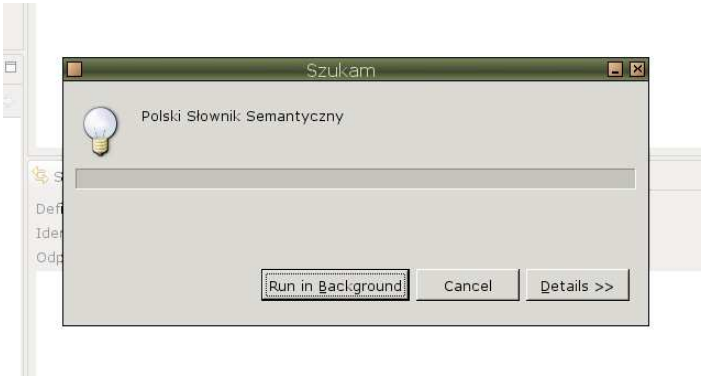


Rysunek D.4. Wyniki wyszukiwania słowa.

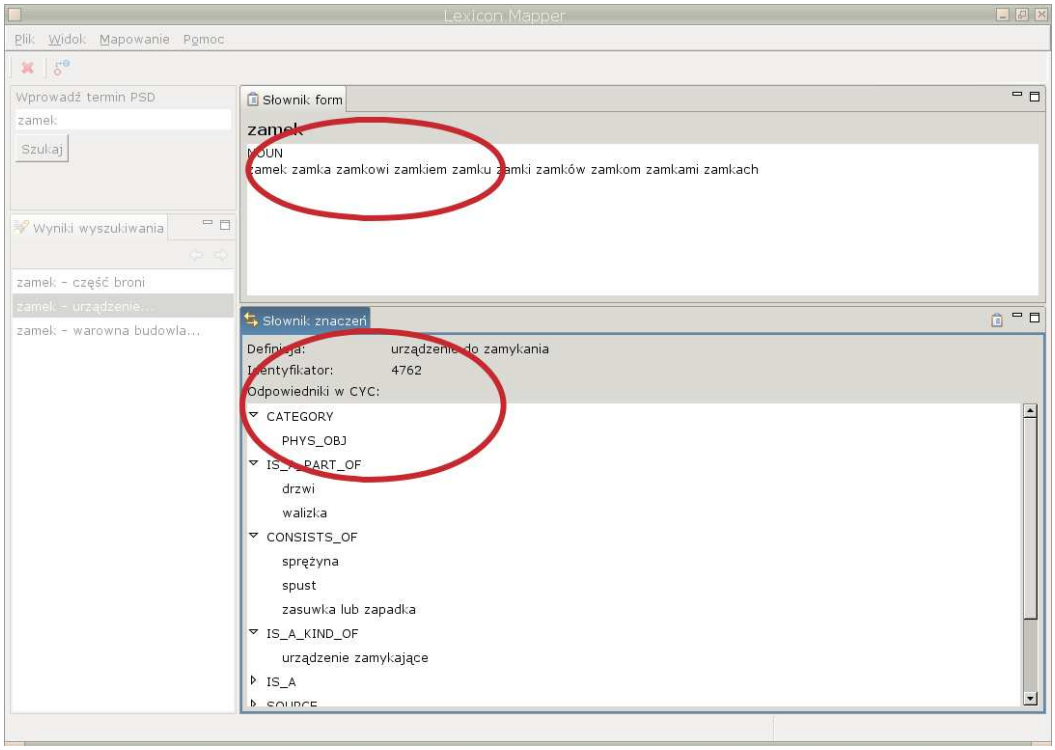
się również w słowniku form fleksyjnych. Jeśli słowo występuje w słowniku form, to wyświetlane są jego formy leksykalne, lecz informacja semantyczna jest niedostępna. W przeciwnym wypadku użytkownik informowany jest również, że poszukiwane słowo nie występuje w słowniku form.

Przy pierwszym wyszukiwaniu słowa następuje ładowanie Słownika Semantycznego, co może trwać dosyć długo. Użytkownik informowany jest o zaistniałej sytuacji poprzez pojawiające się okienko przedstawiające postęp wyszukiwania (patrz rysunek D.5). Użytkownik może wtedy przerwać proces wyszukiwania przez kliknięcie przycisku `Cancel` lub odesłać go do tła, poprzez kliknięcie przycisku `Run in background`. Ta druga operacja pozwoli mu na korzystanie ze słownika w trakcie wyszukiwania słowa.

W przypadku, gdy słowo posiada kilka homonimów, można je rozróżnić po fragmentach etykiet, które są do nich dołączone. Aby obejrzeć formy leksykalne i definicję semantyczną



Rysunek D.5. Postęp w procesie wyszukiwania słowa.



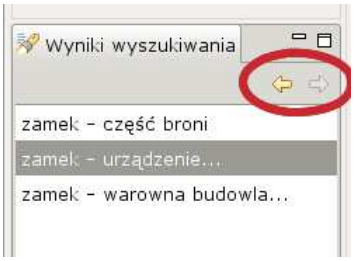
Rysunek D.6. Formy leksykalne i definicja semantyczna słowa.

wybranego słowa, należy na nim dwukrotnie kliknąć. W wyniku tego, formy leksykalne oraz definicja semantyczna pojawią się w odpowiednich oknach (patrz rysunek D.6).

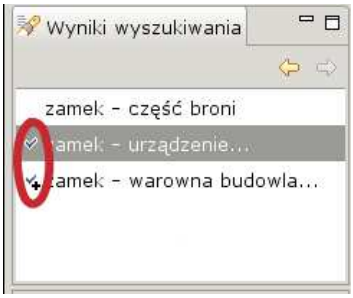
Jeśli użytkownik wyszukiwał kilku słów, to może on poruszać się po wynikach wyszukiwania używając strzałek przedstawionych na rysunku D.7.

Haczyk pojawiający się przy wynikach wyszukiwania oznacza, że wybrane słowo zostało zmapowane na jeden koncept występujący w Cyc, natomiast haczyk z plusem, że zostało zmapowane na więcej niż jeden koncept (patrz rysunek D.8).

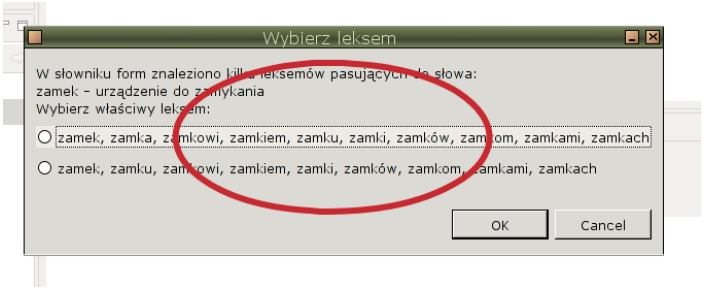
W rzadkich przypadkach, gdy w słowniku form leksykalnych występuje kilka leksemów odpowiadających wybranemu słowu, użytkownik jest proszony o wybranie formy poprawnej (patrz rys. D.9). Informacja ta jest rejestrowana i pytanie to nie będzie powtarzane przy kolejnych uruchomieniach systemu. Użytkownik może zrezygnować z wyboru właściwej formy



Rysunek D.7. Nawigacja po wynikach wyszukiwania słów.



Rysunek D.8. Koncepty posiadające mapowanie.



Rysunek D.9. Wybór właściwej formy leksykalnej słowa.

(przycisk `Cancel`), lecz przez to nie będzie miał możliwości zmapowani wybranego słowa na odpowiadający mu koncept ontologii Cyc.

Słownik form i słownik znaczeń

Poza podstawowymi informacjami semantycznymi, które dostępne są dla użytkownika w oknie *Słownika znaczeń*, może on dowiedzieć się o relacjach semantycznych, w których wybrane słowo występuje jako drugi lub trzeci argument. Informacje te zgromadzone są w oknie *Informacje dodatkowe*, które pojawia się, gdy użytkownik kliknie ikonę wskazaną na rysunku D.10. Ponowne kliknięcie tej ikony powoduje ukrycie okna z informacjami dodatkowymi.

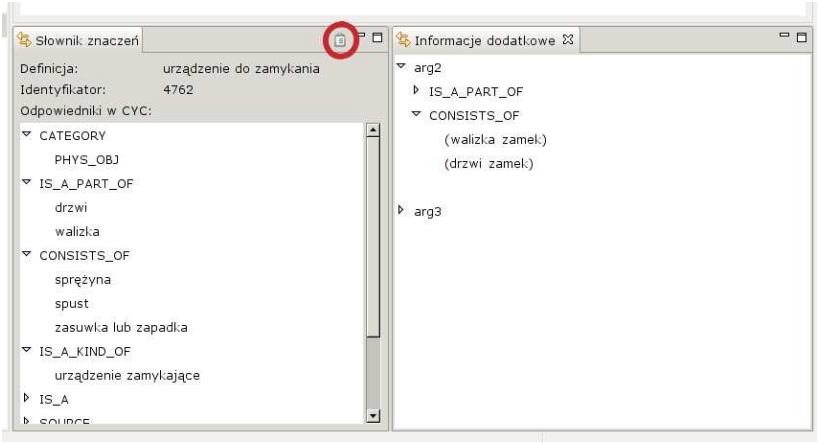
Koncepty, na które zostało zmapowane wybrane słowo, wymienione są w oknie *Słownika znaczeń*. Kliknięcie na wybranym koncepcie powoduje przejście do perspektywy mapowania i wyświetlenie dostępnej w ontologii Cyc informacji na temat tego konceptu (patrz rysunek D.11).

D.2.3. Mapowanie

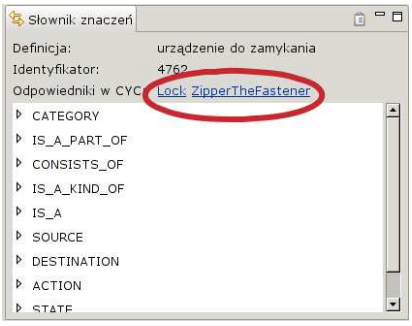
Aby przełączyć się z perspektywy wyszukiwania do mapowania, należy wybrać z menu głównego pozycję `Mapowanie`→`Mapuj pojęcie` lub kliknąć ikonę mapowania w pasku akcji (patrz rysunek D.12).

Propozycje mapowania

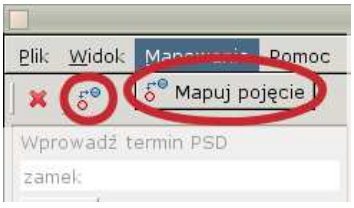
Po wykonaniu tej akcji system automatycznie zaproponuje kilka konceptów, na które może być zmapowane wybrane słowo. Propozycje te pojawiają się w oknie wyników wyszuki-



Rysunek D.10. Dodatkowe informacje semantyczne.



Rysunek D.11. Koncepty na które zostało zmapowane wybrane słowo.



Rysunek D.12. Mapowanie pojęcia – przejście do perspektywy mapowania.

Rysunek D.13. Propozycje mapowania wybranego słowo na koncepty Cyc.

Rysunek D.14. Dodanie mapowania pomiędzy polskim słowem a konceptem Cyc.

wania konceptów Cyc (patrz rysunek D.13). Jeśli słowo jest już zmapowane na któryś z konceptów, to pojawia się przy nim charakterystyczny haczyk.

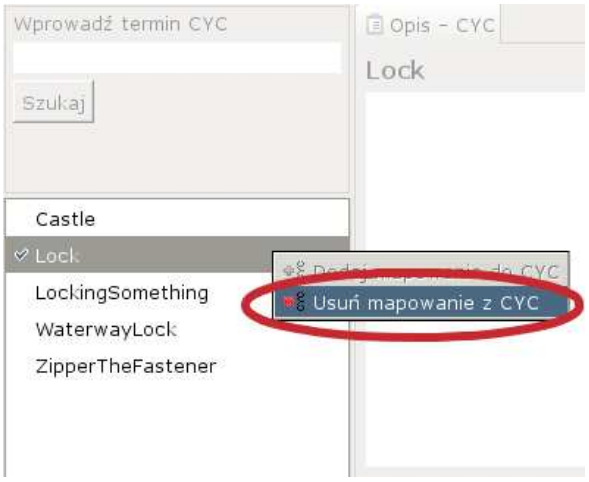
Użytkownik może zapoznać się z definicjami poszczególnych konceptów klikając na nich dwukrotnie. Komentarz oraz relacje stowarzyszone z nimi pojawiają się w oknach *Opis – Cyc* oraz *Relacje – Cyc*, umożliwiając porównanie definicji mapowanych pojęć (patrz rysunek D.2).

Dodawanie mapowania

Kiedy użytkownik uzna, że wybrane słowo odpowiada wybranemu konceptowi Cyc, może dodać mapowanie poprzez wybranie akcji *Dodaj mapowanie* w menu podręcznym pojawiającym się po kliknięciu prawym klawiszem na koncepcie Cyc (patrz rysunek D.14).

Zanim mapowanie zostanie dodane do ontologii Cyc, użytkownik zostanie ponownie zapytany o potwierdzenie swojego wyboru (patrz rysunek D.15). Jeśli nie wystąpi żaden błąd, mapowanie zostaje dodane do ontologii Cyc, co jest odzwierciedlone w GUI przez aktualizację ikon pojawiających się przy wynikach wyszukiwania oraz w oknie *Słownik znaczeń*.

Rysunek D.15. Potwierdzenie dodania mapowania.



Rysunek D.16. Usunięcie niewłaściwego mapowania.

Usuwanie mapowania

W przypadku gdy użytkownik uzna, że jakieś mapowanie jest niepoprawne, może je usunąć klikając prawym klawiszem na koncepcie, który został błędnie zmapowany i wybrać z menu podręcznego opcję *Usuń mapowanie* (patrz rysunek D.16). Również w tym wypadku zostanie on zapytany o potwierdzenie wybranej akcji, a o jej skutecznym zakończeniu, również zostanie poinformowany odpowiednim komunikatem.

Bibliografia

[1] Baader F., Calvanese D., McGuinness D. L., Nardi D., Patel-Schneider P. F., redaktorzy: *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Blackburn S.: *The Oxford Dictionary of Philosophy*. Oxford University Press, Oxford, 1994.

[3] Bouquet P., Serafini L., Zanobini S.: „Semantic coordination: a new approach and an application", 2003. Dostępne: http://citeseer.ist.psu.edu/bouquet03semantic.html.

[4] Carlson L., Nirenburg S.: „Practical world modelling for NLP applications". *Proceedings of the Third Conference on Applied Natural Language Processing*. Association for Computational Linguistics, 1992, strony 235–236.

[5] Chalupsky H.: „Ontomorph: A translation system for symbolic knowledge". *Principles of Knowledge Representation and Reasoning*, 2000, strony 471–482.

[6] Doan A., Madhavan J., Domingos P., Halevy A.: „Learning to map between ontologies on the semantic web", 2002. Dostępne: http://citeseer.ist.psu.edu/doan02learning.html.

[7] Eckel B.: *Thinking in Java – edycja polska*. Wydawnictwo Helion, Gliwice, 2001.

[8] Fellbaum C.: *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[9] Fensel D., Harmelen F. van, Horrocks I., McGuinness D., Patel-Schneider P.: „Oil: An ontology infrastructure for the semantic web", 2001. Dostępne: http://citeseer.ist.psu.edu/fensel01oil.html.

[10] Gajęcki M.: „Biblioteka CLP – opis użytkowy", 2003.

[11] Genesereth M. R.: „Knowledge Interchange Format". *KR*, 1991, strony 599–600.

[12] Gruber T.: „A translation approach to portable ontology specifications". *Knowledge Acquisition*. Academic Press, 1993, strony 199–220.

[13] Gruber T. R.: „Ontolingua: A mechanism to support portable ontologies", 1992. Dostępne: http://citeseer.ist.psu.edu/gruber92ontolingua.html.

[14] Heflin J., Hendler J., Luke S.: „SHOE: A knowledge representation language for internet applications". Raport instytutowy CS-TR-4078, 1999.

[15] Horrocks I.: „DAML+OIL: A reason-able web ontology language". *Extending Database Technology*, 2002, strony 2–13.

[16] Ingarden R.: *Spór o istninie świata*, wolumen I. PWN, Warszawa, 1987.

[17] Kay M.: „XSL Transformations (XSLT) version 2.0", 2006. Dostępne: http://www.w3.org/TR/xslt20/.

[18] Kifer M., Lausen G.: „F-logic: a higher-order language for reasoning about objects, inheritance, and scheme". *SIGMOD RECORD*, wolumen 18, 1989, strony 134–146.

[19] Lenat D. B.: „CYC: A large-scale investment in knowledge infrastructure". *Communications of the ACM*, 38(11):33–38, 1995.

[20] Lenat D. B., Guha R. V.: „The evolution of cycL, the cyc representation language". *SIGART Bulletin*, 2(3):84–87, 1991.

[21] Lubaszewski W.: *Gramatyka leksykalna w maszynowym słowniku fleksyjnym*. Wydawnictwo IJP PAN, 1997.

[22] Lyons J.: *Introduction to Theoretical Linguistics*. Cambridge University Press, Cambridge, 1968.

[23] Mahesh K., Nirenburg S.: „Semantic classification for practical natural language processing", 1995. Dostępne: http://citeseer.ist.psu.edu/article/mahesh95semantic.html.

[24] Manola F., Miller E.: „RDF Primer", 2004. Dostępne: http://www.w3.org/TR/rdf-primer/.

[25] McGuinness D. L., Harmelen F. van: „OWL Web Ontology Language Overview", 2004. Dostępne: http://www.w3.org/TR/owl-features/.

[26] Millera G. A.: „Ambiguous words", 2001. Dostępne: http://www.kurzweilai.net/meme/frame.html?main=/articles/art0186.html.

[27] Motta E.: „An Overview of the OCML Modelling Language", 1998. Dostępne: http://citeseer.ist.psu.edu/motta98overview.html.

[28] Niles I., Pease A.: „Towards a standard upper ontology", 2001. Dostępne: http://citeseer.ist.psu.edu/niles01towards.html.

[29] Ogden C. K., Richards I. A.: *The Meaning of Meaning*. Routledge & Kegan Paul, Londyn, 1946.

[30] Peirce C. S.: *Collected Papers of Charls Sanders Peirce*. Cambridge, Mass., 1960.

[31] Peter Karp Vinay Chaudhri J. T.: „XOL: An XML-Based Ontology Exchange Language", 1999. Dostępne: http://www.ai.sri.com/pkarp/xol/xol.html.

[32] PWN: „Wielki Multimedialny Słownik angielsko-polski/polsko-angielski Oxford/PWN", 2004. Dostępne: http://oup.pwn.pl/op_zasob_slownika.html.

[33] R. MacGregor R. B.: „The LOOM knowledge representation language". Information Sciences Institute, Univ. of Southern California, 1987, strony 87–188.

[34] Reed S. L., Lenat D. B.: „Mapping ontologies into Cyc". *AAAI 2002 Conference Workshop on Ontologies For The Semantic Web*, Edmonton, Canada, Lip. 2002. AAAI.

[35] Shirky C.: „Ontology is overrated: Categories, links, and tags", 2005. Dostępne: http://www.shirky.com/writings/ontology_overrated.html.

[36] The Gene Ontology Consortium: „Gene ontology: tool for the unification of biology". *Nature Genet*, 25:25–29, 2000.