KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2012/2013
pod redakcją Tomasza Kubika

KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2012/2013
pod redakcją Tomasza Kubika

Skład komputerowy, projekt okładki
Tomasz Kubik

ISBN 978-83-930823-4-6

Wydawca
Tomasz Kubik

Druk i oprawa
I-BiS sc., ul. Lelewela 4, 53-505 Wrocław

Skład niniejszej książki wykonano w systemie LaTeX

SPIS TREŚCI

SŁOWO WSTĘPNE

Czwarty tom z cyklu *Komputerowe przetwarzanie wiedzy, Kolekcja prac 2012/2013* dokumentuje wybrane projekty wykonane przez studentów studiów magisterskich Wydziału Elektroniki Politechniki Wrocławskiej, kierunku Automatyka i robotyka, specjalności Robotyka. Projekty zrealizowano pod kierunkiem dra inż. Tomasza Kubika w ramach kursu Komputerowe przetwarzanie wiedzy, w semestrze dyplomowym roku akademickiego 2012/2013. Tradycyjnie, celem Projektu było przedstawienie wybranych metod i narzędzi do przetwarzania wiedzy oraz ich implementacji programowych służących do rozwiązania konkretnych zadań praktycznych. Niniejsza Kolekcja składa się z ośmiu prac scharakteryzowanych zwięźle poniżej.

- D. Bieda, M. Adamczyk, P. Kułakowski, *Metody usprawniające tworzenie słowników*. Przedmiotem projektu jest automatyczne tworzenie słowników wykorzystywanych przy komputerowym przetwarzaniu wiedzy. Przedstawiono przegląd modeli słowników i narzędzia programowe do analizy tekstu. Przedstawiono przykładowe zastosowanie omówionych narzędzi do analizy tekstu dotyczącego robotyki.
- M. Orda, M. Dziergwa, *Wykorzystanie technik analizy tekstów w ocenie szkolnych prac*. Projekt dotyczy komputerowej analizy szkolnych prac pisemnych z języka polskiego, wspomagającej nauczyciela. Przedmiotem analizy jest składnia, kompozycja, a w pewnym stopniu także treść zawarta w tekście. W oparciu o dostępne techniki i narzędzia stworzono i zaimplementowano program komputerowy przeprowadzający taką analizę.
- J. Wróbel, M. Fuławka, *Ocena sprawności wyszukiwania informacji*. W projekcie przedstawiono kryteria i metody oceny jakości informacji uzyskiwanej przy pomocy wyszukiwarek internetowych. Zaimplementowano i przebadano pewną metodę szeregowania wyników wyszukiwania.
- D. Moskal, M. Sawicki, *Budowa silników dostarczających rekomendacji*. Projekt porusza zagadnienia rekomendacji informacji. Omówiono wybrane algorytmy rekomendujące i ich zastosowanie. Opracowano i przedstawiono aplikację programową przeznaczoną do rekomendacji filmów zasługujących na obejrzenie.
- N. Czop, M. Gawron, M. Orynicz, *Modelowanie i eksploracja grafowych baz danych*. W pracy przedstawiono zasady budowy i działania grafowych baz da-

nych, a także dokonano próby oceny możliwości ich zastosowania. Rozważania teoretyczne zilustrowano przykładowymi implementacjami programowymi.
- R. Kmieć, J. Zych, *Robotyczne zastosowanie dokumentowej bazy danych*. Projekt dotyczy analizy dokumentowych baz danych w kontekście ich zastosowania w robotyce. Przedstawiono przegląd narzędzi programowych do tworzenia baz dokumentowych, jak również działanie takich baz. Dokonano przykładowej implementacji bazy dokumentowej dla robotyki i dokonano jej oceny.
- M. Kret, P. Ptasznik, *Systemy decyzyjne*. W pracy przedstawiono podstawy teoretyczne systemów decyzyjnych, ze szczególnym uwzględnieniem systemów ekspertowych, produkcyjnych, zagadnienia reprezentacji wiedzy i logiki systemów decyzyjnych. Przedstawiono przykładową realizację systemu decyzyjnego dotyczącego bezpieczeństwa ruchu drogowego.
- I. Góral, K. Szydłowski, *Generator unikalnych identyfikatorów*. Omówiono definicję i zastosowanie uniwersalnych identyfikatorów, a także metody ich generowania. Przedstawiono implementację generatora unikalnych identyfikatorów w postaci programu zewnętrznego w stosunku do systemu zarządzania bazą wiedzy.

Kolekcja wyróżnia się przemyślaną koncepcją i staranną redakcją. Można oczekiwać, że podobnie jak jej trzy poprzedniczki, także niniejsza książka stanie się przydatnym źródłem informacji dla Czytelników zainteresowanych zadaniami, metodami i zastosowaniami komputerowego przetwarzania wiedzy.

Prof. Krzysztof Tchoń,
opiekun specjalności Robotyka,
Wrocław, styczeń 2014

ROZDZIAŁ 1

METODY USPRAWNIAJĄCE TWORZENIE SŁOWNIKÓW

D. Bieda, M. Adamczyk, P. Kułakowski

Kodyfikacja wiedzy jest jednym z ważniejszych czynników wpływających na postęp ludzkiej cywilizacji. Jego istota polega na usystematyzowaniu i zebraniu w jedną całość wszelkich informacji potrzebnych do funkcjonowania i dalszego rozwoju, z zastosowaniem uznanych zasad i formalizmów. Pierwsze znaczące osiągnięcia w kodyfikacji wiedzy można wiązać z wynalezieniem pisma. To właśnie pismo pozwoliło ludziom na utrwalenie myśli oraz zgromadzonych doświadczeń na nośniku, który zapewniał im większą trwałość niż ulotna pamięć. Utrwalenie wiedzy na papierze pozwoliło zwiększyć dostęp do informacji, stworzyło okazję do jej archiwizacji oraz przetwarzania. Kolejnym znaczącym krokiem w kodyfikacji wiedzy było wynalezienie komputerów oraz technik zapisywania danych w postaci elektronicznej. Powstające systemy informatyczne umożliwiły implementację wielu skutecznych i szybkich metod automatycznego przeszukiwania i klasyfikowania zapisanych informacji.

Mimo technologicznego skoku sam proces kodyfikacji wiedzy niewiele się zmienił. Do podstawowych metod kodyfikacji wiedzy zalicza się tworzenie słowników – uporządkowanych zbiorów słów. Klasycznymi przykładami słowników są słowniki języków obcych, ortograficzne, synonimów i antonimów oraz wiele innych. Słowom (a może lepiej mówić terminom) występującym w słowniku często towarzyszy krótki opis ich znaczenia, zredagowany w języku naturalnym. Słowniki, oprócz funkcji porządkowania, dostarczają również informacji o wzajemnych związkach między zgromadzonymi w nich słowami. Przykładowo, w słowniku synonimów można znaleźć informację, że słowo „dom" powiązane jest ze słowami „budynek", „chałupa" i „chata". Z kolei w słowniku języka polskiego można znaleźć pokrewne związki wyrazowe, takie jak „dom dziecka", „dom kultury", „dom starców". Słownik polsko-niemiecki przynosi zaś informację, że odpowiednikiem „dom" jest termin „das Haus".

Dziś już nie wyobrażamy sobie życia bez wyszukiwarek internetowych, które w pewnym sensie można uznać za następczynie katalogów oraz zbiorów bibliotecznych, tradycyjnych encyklopedii i słowników. Użytkownicy komputerów za-

miast papierowych woluminów wybierają najczęściej ich elektroniczne odpowiedniki. Pozwalają one znaleźć poszukiwane materiały bez konieczności wstawania od własnego biurka i to na dodatek w sposób nieporównywalnie szybszy i efektywniejszy niż to kiedyś bywało.

Można zaobserwować, że z roku na rok wyszukiwarki internetowe zwiększają swoją efektywność i szybkość działania. Jest to możliwe m.in. dzięki słownikom w postaci uporządkowanej sieci słów i zwrotów. Sieci słów to konstrukcje grafowe, umożliwiające m.in. komputerowe przetwarzanie wyrażeń pochodzących z języka naturalnego (wyrażeń, w których znaczenie pojedynczych wyrażeń często zależy od kontekstu całej wypowiedzi), katalogowanie stron, wyłuskiwania słów kluczowych oraz budowania powiązań danych stron z innymi stronami o podobnej zawartości, ale inaczej zredagowanej treści.

W niniejszym rozdziale zostaną opisane słowniki, ich modele oraz metody tworzenia słowników w sposób automatyczny. W części 1.1 wyjaśniono czym są słowniki i jakie są ich rodzaje. Następnie w części 1.2 przedstawiono trzy wybrane modele słowników oraz dokonano ich porównania. W części 1.3 wskazano istniejące automatyczne metody wyłuskiwania słów i znaczeń. Wreszcie, w części 1.4, zaprezentowano narzędzia, które usprawniają tworzenie słowników.

1.1. Czym są słowniki?

Słowniki są uporządkowaną siecią słów i zwrotów. Słowniki mogą być kontrolowane (ich tworzenia podlega pewnym regułom) lub niekontrolowane lub naturalne (ich tworzenie nie podlega jakimś ścisłym regułom).

Zawartość słowników przystosowanych do komputerowego przetwarzania dzieli się na treści wyświetlane oraz treści kontrolne. Pierwsze zawierają informacje w postaci przystosowanej do wyświetlenia użytkownikowi, na przykład w postaci języka naturalnego. Treści kontrolne zawierają dane dla tzw. silników przetwarzających - komponentów wchodzących w skład poszczególnych rozwiązań programowych. Ponadto słowniki mogą zawierać dodatkowe informacje na temat formatów zapisu danych.

Ze względu na przeznaczenie oraz budowę rozróżnia się następujące typy słowników i kontrolowanego słownictwa (wg [1] oraz tłumaczenia za pomocą [2]):

Generalne relacje (ang. *relationships in general*) – zależność pomiędzy dwoma przedmiotami relacji (np. polami lub tabelami w bazie danych).

Listy haseł przedmiotowych (ang. *subject heading lists*) – wybrane i zunifikowane wyrażenia i zwroty definiujące główne tematy, przypisywane poszczególnym kategoryzowanym obiektom (np. książkom, artykułom) w celu ułatwienia ich wyszukiwania.

Listy kontrolowane (ang. *controlled lists*) – zawierają słownictwo przypisane określonej dziedzinie. Jest ono tak dobrane, aby umożliwić łatwe katalogowanie z ich użyciem – zawężanie słownictwa do niedużej ściśle kontrolowanej listy pozwala na uzyskanie zwięzłości i uniknięcie błędów.

Synonimy (ang. *synonym ring lists, synsets*) – są to słowa i zwroty o podobnym bądź identycznym znaczeniu (synonimy, wyrazy bliskoznaczne). Mają szcze-

gólne znaczenie w systemach wyszukiwania informacji, gdyż pozwalają użytkownikowi na odnalezienie informacji skatalogowanej lub zapisanej (w przypadku wyszukiwania pełnotekstowego) z użyciem innych zwrotów niż podane przez szukającego.

Kartoteka autorytatywna (ang. *authority files*) – zestaw usystematyzowanych nazw oraz referencji do innych wariantów oraz form. Mogą to być na przykład tytuły książek w różnych językach. Synonimy, w przeciwieństwie do innych wymienionych tutaj form, bezpośrednio nie mogą być stosowane w kartotece autorytatywnej.

Taksonomie (ang. *taxonomies*) – taksonomia jest hierarchicznie usystematyzowanym sposobem porządkowania wiedzy, opartym na kategoriach systematycznych w postaci taksonów. Taksony powiązane są relacjami rodzic/dziecko, np. gatunek/podgatunek, co pozwala na zdefiniowanie ścisłych relacji. Taksonomie są powszechnie stosowane w biologii w organizacji gatunków zwierząt i roślin.

Alfanumeryczne schematy klasyfikacji (ang. *alphanumeric classification schemes*) – te schematy opierają się na przydzieleniu poszczególnym pozycjom i kategoriom kodów złożonych z liter lub cyfr. Przykładem zastosowania w praktyce jest Klasyfikacja Dziesiętna Deweya, w której zbiorom bibliotecznym przypisując się odpowiedni kod cyfrowy definiujący ich kategorię. Dla przykładu główne działy definiowane są pierwszą cyfrą kodu: 000 – Dział ogólny, 100 – Filozofia, 200 – Teologia itd.

Tezaurusy (ang. *thesauri*) – w tezaurusach obiekty są połączone siecią powiązań semantycznych. Te powiązania zawierają hierarchię (tak jak w taksonomiach) obiektów nadrzędnych oraz podrzędnych. Mogą też zawierać w nich opisy bardziej złożonych relacji (synonimy, antonimy i inne), jaki również definicje czy objaśnienia pojęć.

Ontologie (ang. *ontologies*) – ontologia, w przypadku systemów gromadzenia wiedzy, określa strukturę pojęć oraz związków pomiędzy nimi. Pojęcia grupowane są w klasy i podklasy, pojęciom można przypisać atrybuty występujące w pewnej hierarchii. Poszczególne elementy ontologii mogą zawierać opisy, jak je należy rozumieć. Ponieważ ontologia definiuje ścisłe relacje pomiędzy pojęciami, możliwe do odczytania przez maszyny, nie jest ona ani taksonomią ani tezaurusem (tezaurusy nie mają tak ostrych wymagań).

Folksonomie (ang. *folksonomies*) – kategoryzacja treści odbywa się poprzez dodawanie do nich tzw. tagów (słów albo krótkich wyrażeń). Z reguły ten sposób klasyfikacji jest dokonywany przez osoby przetwarzające jakieś treści. Umożliwia to łatwą, ale niezbyt systematyczną ich klasyfikację.

Poszczególne metody organizacji słownictwa i wiedzy oferują różne możliwości, dające się wykorzystywać w różnych zastosowaniach. Część z nich powstała na potrzeby organizacji zbiorów w różnego rodzaju bibliotekach, jak np. listy haseł przedmiotowych, kartoteki autorytatywne czy klasyfikacje alfanumeryczne. Spełniają one dobrze swoje zadanie, jednak w tradycyjnej formie używane są coraz rzadziej. Ma to związek z coraz szybszą cyfryzacją zarówno spisów jak i źródeł wiedzy.

W organizacji danych cyfrowych najbardziej rozpowszechnione są sposoby oparte o tezaurusy, ontologie i folksonomie. Popularność stosowania tych ostatnich ciągle wzrasta, ponieważ nie dość, że umożliwiają organizowanie i przetwarzanie wiedzy w sposób znacznie bardziej efektywny niż pozostałe metody, to jeszcze same mogą podlegać algorytmizacji.

W dalszej części rozdziału skupiono się na słownikach opartych o tezaurusy i ontologie, które są modelami chętnie wykorzystywanymi do budowy nowych słowników.

1.2. Modele słowników

Budowa modeli języków ma na celu ułatwienie tworzenia słowników oraz sformalizowanie reprezentacji wiedzy, co z kolei przekłada się na możliwość zautomatyzowanego przetwarzania słownictwa z wybranej dziedziny wiedzy lub bardziej ogólnych obszarów pojęciowych. Pracę nad modelami języków przyczyniają się również do poznania tajników ludzkiego umysłu i zrozumienia, w jaki sposób pojmujemy otaczającą nas rzeczywistość. W niniejszym rozdziale zostaną omówione następujące modele: SKOS, WordNet, Słowosieć.

1.2.1. SKOS

SKOS (ang. *Simple Knowledge Organization System*) jest standardem opracowanym przez organizację W3C (ang. *World Wide Web Consortium*). Jego zadaniem jest wspomaganie tworzenia różnego rodzaju reprezentacji ustrukturyzowanych słowników, takich jak tezaurusy, taksonomie, czy systemy klasyfikacyjne.

SKOS Core definiuje klasy i właściwości wystarczające do reprezentacji najczęściej opisywanych w słownikach związków. Podejście do rozumienia słownictwa opiera się tu na określeniu znaczenia powiązań i relacji między wyrazami w tekście, a nie na rozumieniu znaczeń konkretnych wyrazów.

Obiekty definiowane w tym systemie nie są termami, ale abstrakcyjnymi pojęciami oraz łączącymi je konceptami (typu: „należy do", „jest w posiadaniu"). SKOS jest językiem opartym na języku RDF (ang. *Resource Description Framework*) i OWL (ang. *Ontology Web Language*), dzięki czemu każdy koncept:

- ma przypisany pewien termin, słowo lub związek wyrazowy określający znaczenie tego konceptu,
- może być powiązany z synonimami lub alternatywnymi określeniami,
- może mieć dodatkowe definicje i notatki (np. specyficzne dla danego języka adnotacje dotyczące odmiany wyrazów przez osoby czy przypadki).

Bazowymi pojęciami w SKOSie są zasoby koncepcyjne, etykiety, notatki i schematy koncepcyjne. Zasobem koncepcyjnym jest koncept, mający pewne określone znaczenie w opisywanym środowisku, oznaczone jednoznacznym identyfikatorem URI. Do konceptu można przypisać jedną lub kilka etykiet w wybranym języku oraz notatki. Notatki pełnią rolę dokumentacji i mogą zawierać: definicję, przykład, notę edytorską, notkę historyczną lub inne. W zależności od potrzeb

koncepty mogą być powiązane różnymi typami relacji: klasyfikacja, asocjacje, mapowanie, translacje.

Koncepty mogą być zorganizowane w hierarchie używające relacji poszerzających i zwężających (np. organizm -> zwierzę -> kot -> kot syjamski) i/lub mogą być powiązane asocjacyjnie, tj. skojarzeniowo (np. lekarz, pielęgniarka, ośrodek zdrowia, NFZ, leki, pacjent, apteka, recepta). Koncepty mogą być również połączone w schematach koncepcyjnych (ang. *concept schemes*) i kolekcjach, zapewniających uporządkowany sposób reprezentacji słownictwa (ang. *controlled vocabulary*) zawartego w systemie.

1.2.2. WordNet

WordNet jest siecią semantyczną słów i grup słów połączonych pod względem leksykalnym (ang. *lexical*) oraz znaczeniowym (ang. *conceptual*) poprzez nazwane relacje. Podstawowym blokiem tworzącym WordNet jest *synset*, będący zbiorem synonimów, czyli słów bliskoznacznych. Pojedynczy synset zawiera słowa powiązane znaczeniowo, jednak niekoniecznie równoważne w każdym kontekście. To samo słowo może być również członkiem wielu synsetów, co ukazuje obecność zjawiska polisemii, w którym jedno słowo posiada wiele znaczeń. Synsety w modelu WordNetu podzielone są na cztery grupy reprezentujące główne części mowy występujące w języku angielskim:

- rzeczowniki (ang. *nouns*),
- czasowniki (ang. *verbs*),
- przymiotniki (ang. *adjectives*),
- przysłówki (ang. *adverbs*).

Podstawowymi relacjami występującymi w modelu są:

synonimia (ang. *synonymy*) – czyli równorzędna relacja łącząca słowa o znaczeniu bliskoznacznym oraz

antonimia (ang. *antonymy*) – łącząca słowa o znaczeniu przeciwstawnym.

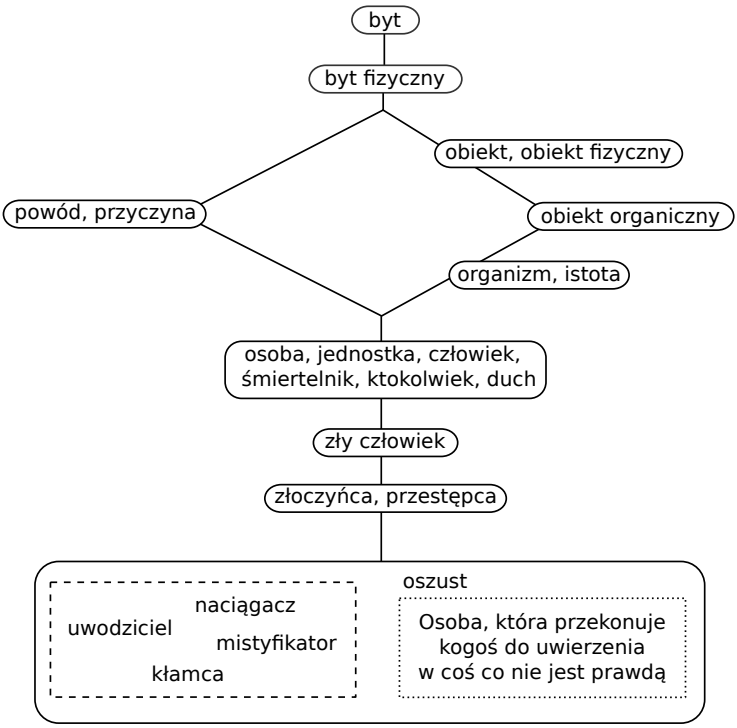
Synonimia i antonimia mogą łączyć ze sobą pojedyncze słowa, jak i całe synsety wchodzące w skład wszystkich czterech części mowy. Relacjami charakterystycznymi dla grupy rzeczowników są:

hiponimia (ang. *hiponymy*) – jest relacją hierarchiczną, łączącą słowa o znaczeniu bardziej szczegółowym (hiponimy) ze słowami o znaczeniu ogólnym (hiperonimami). Przykładowo słowo „pies" jest hiponimem słowa „ssak", które jest hiponimem słowa „zwierzę".

meronimia (ang. *meronymy*) – jest relacją odwołującą się do struktury przedmiotu. Przykładowo meronimem słowa „pies" jest słowo „łapa", a meronimem słowa „łapa" jest słowo „pazur". „Pies" jest holonimem słowa „pazur".

Przykładowe drzewo relacji z grupy rzeczowników przedstawiono na rys. 1.1. W grupie czasowników występują następujące relacje [3]:

troponimia (ang. *troponymy*) – relacja „sposobu", występującą pomiędzy dwoma leksemami. Przykładem relacji tego typu jest związek słów „szeptać" i „rozmawiać".



Rys. 1.1: Drzewo rzeczowników w modelu WordNetu.

wsteczny majorat (ang. *backward entailment*) – czynności poprzedzające inne czynności, np. „rozwód" i „ślub".

presupozycja (ang. *presupposition*) – założenie, które należy nadać słowu, aby było ono prawdziwe. Przykładem jest układ słów „kupować" i „płacić".

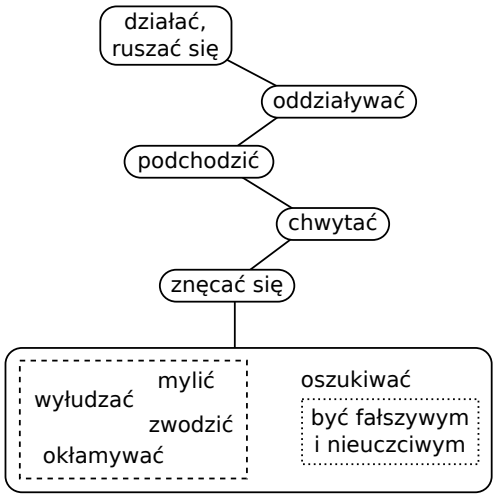
przyczyna (ang. *cause*) – czyli powód działania, na przykład „pokazać" i „zobaczyć".

Na rys. 1.2 przedstawiono przykładowe drzewo czasowników. Model WordNetu rozróżnia przymiotniki opisowe i relacyjne. Przymiotniki opisowe są wzajemnie połączone za pomocą relacji antonimii i synonimii. Przykładowe relacje pokazano na rys. 1.3. Przymiotniki relacyjne połączone są rzeczownikami, na przykład „atomowy" i „atom". Przysłówki powiązane są z przymiotnikami, od których pochodzą, na przykład „szybki" i „szybko".

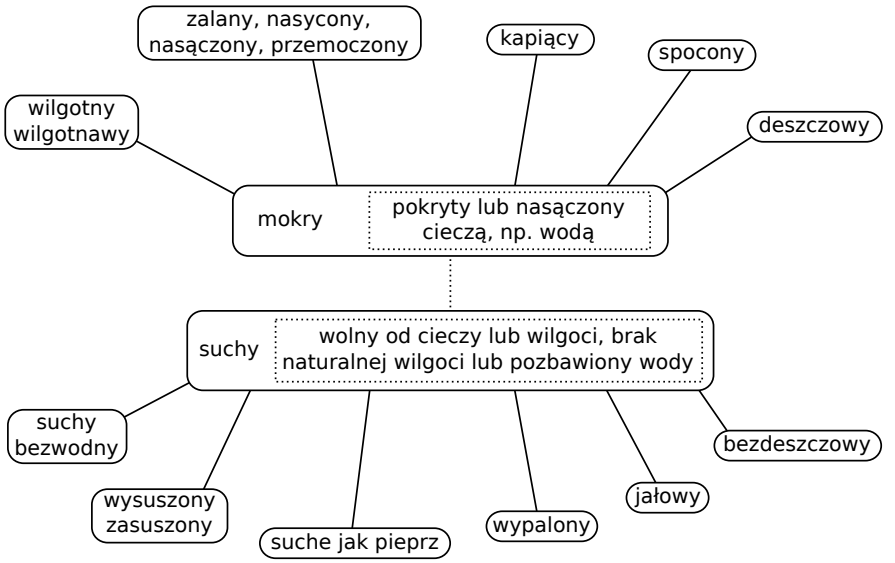
1.2.3. Słowosieć

„Polacy nie gęsi i swoją słowosieć mają" (parafraza słów Mikołaja Reja).

Słowosieć (inaczej plWordNet) jest udaną próbą przeniesienia na polski grunt idei WordNetu [4]. Warto wspomnieć, że po sukcesie pierwszego, angielskiego WordNetu powstało i jest rozwijanych wiele innych projektów, takich jak EuroWordnet dla języków: holenderskiego, włoskiego, hiszpańskiego czy BalkaNet dla



Rys. 1.2: Drzewo czasowników w modelu WordNetu.



Rys. 1.3: Drzewo przymiotników w modelu WordNetu.

bułgarskiego, greckiego, rumuńskiego serbskiego i tureckiego, a to tylko dla języków europejskich, podobne projekty prowadzone są na całym świecie, co świadczy o sukcesie idei WordNetu.

Prace rozpoczęte w 2005 roku oparły się o korpus języka polskiego IPI PAN (dostępny na stronie `http://korpus.pl/`). Dane tam zawarte zostały przefiltrowane pod względem częstości występowania, a następnie poddane ręcznej obróbce co, we współpracy z lingwistami, pozwoliło stworzyć podwaliny Słowosieci. W tej części pracy pomocne było narzędzie *WorNet Weaver* pozwalające

na pół-automatyczną ekstrakcję danych leksykalno-semantycznych dzięki paru zaimplementowanym w nim algorytmom. Następnie baza była systematycznie rozbudowywana. Ze względu na ograniczony budżet projektu największy nacisk położono wtedy na rozwiązania programowe.

plWordNet konstrukcją różni się od WordNetu angielskiego. Jest to podyktowane m.in. specyfiką języka polskiego. Ważnym źródłem odniesienia dla Słowosieci był projekt EuroWordnet. Istnieje jednak parę zasadniczych różnic pomiędzy nimi. Celem Słowosieci było stworzenie sieci słów dla potrzeb języka polskiego, natomiast celem EuroWordnetu - stworzenie paru podobnych sieci.

Główne założenia przy tworzeniu Słowosieci [4]:

- Tak jak w WordNecie i EuroWordnecie synonimy, antonimy hiperonimy i meronimy trzymane są pomiędzy jednostkami leksykalnymi w klasach morfosynktaktycznych (rzeczowniki, przymiotniki i czasowniki).
- Relacje podzielono na te, które łączą synsety i te, które łączą jednostki leksykalne.
- Meronimy podzielone na 6 klas.
- Parę relacji zostało dodanych i redefiniowanych ze względu na polski podział leksykalny.
- Dodano ciągi graficzne zawierające relacje znaczeniowe.

W systemie zawarte są następujące rodzaje relacji leksykalnych (powstałe w oparciu o WordNet i EuroWordnet):

- synonimy,
- antonimy,
- konwersje,
- hipernimy,
- troponimy,
- holonimy, meronimy,
- *pertainymy*,
- *fuzzynymy*.

Dwie ostatnie powstały z EuroWordnetu i wymagają tym samym omówienia:

pertainymy – zdefiniowane są dla przymiotników relacyjnych (są tworzone od rzeczowników, np. „woskowy” od „wosk”).

fuzzynymy – relacje te występują dla jednostek leksykalnych, które łączą się semantycznie, ale nie można było ich połączyć w systemie: (np. „szkoła” i „nauczyciel”, „obraz” i „rzeźba”).

Słowosieć będąca jednym z europejskich wordnetów, jest nie tylko interesującym i ciekawym projektem, ale również dostarcza potężnego narzędzia dla różnych algorytmów. Przykładami zastosowań wordnetów są: analizowania tekstu, segmentacja dla różnych analiz językowych, sprawdzanie poprawności, jak również zastosowania przy rozpoznawania głosu, pisma odręcznego i wiele innych. plWornet umożliwia podjęcie tych działań dla języka polskiego, co otwiera szerokie pole do badań z pogranicza algorytmów i nauk o języku.

1.2.4. Porównanie modeli słowników

W tab. 1.1 przedstawiono porównanie omówionych modeli słowników ze względu na jednostkę budowy, powiązanie z bazą wiedzy, rodzaje zależności w bazie wiedzy oraz kto i kiedy rozpoczął prace nad projektem.

Tab. 1.1: Porównanie modeli słowników.

	SKOS	**Wordnet**	**Słowosieć**
Rodzaj modelu	Standard do tworzenia słowników	Sieć semantyczna j. angielskiego	Sieć semantyczna j. polskiego
Jednostka budowy	koncept	synset	synset
Powiązanie z bazą wiedzy	Nie	Tak – z bazą dot. j. angielskiego	Tak – z bazą dot. j. polskiego
Rodzaje zależności w bazie wiedzy	klasyfikacja, asocjacja, mapowanie, translacja	synonimy, antonimy, konwersje, hipernimy, troponimy, holonimy, meronimy	synonimy, antonimy, konwersje, hipernimy, troponimy, holonimy, meronimy, *pertainymy*, *fuzzynymy*
Data rozpoczęcia prac	1997 r.	1985 r.	2005 r.
Główna jednostka zaangażowana w tworzenie	W3C (ang. *World Wide Web Consortium*)	Laboratorium kognitywistyki na Uniwersytecie Princeton	Grupa G4.19 na Politechnice Wrocławskiej

1.3. Istniejące metody wyłuskiwania słów i znaczeń

W związku z niesamowitą dynamiką przyrostu informacji znaczne wysiłki w dziedzinie informatyki skierowano na zagadnienia związane z przetwarzaniem tekstu. Tagowanie jest istotnym elementem przetwarzania zdania, preprocesingiem umożliwiającym dalsze wykorzystanie algorytmów wyłuskujących znaczenia wyrazów, których rola w zdaniu jest znana.

Mając pewien model języka i prawdopodobieństwa wystąpienia w nim słów czy związków wyrazowych, możliwe jest zastosowanie parserów, które przetwarzają zdania i tagują konkretne wyrazy, rozpoznając:

- część mowy (ang. *part-of-speech*, POS): przymiotnik, rzeczownik, czasownik, przysłówek, przyimek itd.;
- znaczenie elementu (ang. *Named Entity Recognition*): np. człowiek, lokalizacja, firma;
- funkcję w zdaniu: podmiot, orzeczenie, przydawka, dopełnienie i inne.

Dla języka polskiego został stworzony TaKIPI - morfosyntaktyczny tagger przetwarzający zdania w taki sposób, że poszczególnym wyrazom przypisywana jest nie tylko kategoria POS, ale również rozpoznana liczba (pojedyncza czy mnoga), rodzaj (męski, żeński, nijaki) oraz przypadek (mianownik, dopełniacz, celownik, biernik).

1.4. Dostępne narzędzia programowe

W niniejszej części dokonano analizy istniejących narzędzi programowych udostępnionych na stronie Grupy Technologii Językowych G4.19 Politechniki Wrocławskiej (`http://nlp.pwr.wroc.pl/narzedzia-i-zasoby`).

1.4.1. Morfeusz

Morfeusz jest analizatorem morfologicznym dostosowanym na potrzeby mowy polskiej [5]. Analiza morfologiczna polega na analizie z osobna każdego wyrazu znajdującego się w przetwarzanym tekście oraz znalezieniu powiązanych z nim form i wyrazów (leksemy).

Morfeusz dla każdego wyrazu z osobna zwraca powiązane z nim formy oraz znaczniki charakteryzujące formę gramatyczną wyrazu. Przykładowo dla zdania *„Morfeusz jest analizatorem morfologicznym."* otrzymamy:

```
>Morfeusz jest analizatorem morfologicznym.
[Morfeusz,Morfeusz,subst:sg:nom:m1]
[ , ,sp]
[jest,być,fin:sg:ter:imperf]
[ , ,sp]
[analizatorem,analizator,subst:sg:inst:m1.m3]
[ , ,sp]
[morfologicznym,morfologiczny,adj:sg:inst.loc:m1.m2.m3.n1.n2:pos|
adj:pl:dat:m1.m2.m3.f.n1.n2.p1.p2.p3:pos]
[.,.,interp]
```

W powyższym przykładzie słowo „Morfeusz" zostało zaklasyfikowane jako rzeczownik (`subst`), w liczbie pojedynczej (`sg`), słowo „jest" – jako czasownik (`fin`), natomiast „morfologicznym" – jako przymiotnik (`adj`), mogący odnosić się zarówno do liczby pojedynczej (`sg`) „analizatorem morfologicznym", jak i liczby mnogiej (`pl`) „analizatorom morfologicznym". Warto tutaj zwrócić uwagę, że każdy wyraz analizowany jest z osobna, bez analizy jego otoczenia – więc obie możliwe formy są podane. Oznaczenia pochodzą od łacińskich nazw były wzorowane na [6].

Morfeusz z reguły nie jest używany samodzielnie, a raczej jako rozwiązanie dostarczające informacji dla innych narzędzi, np. dla MACA.

1.4.2. Analizator MACA

MACA (ang. *Morphological Analysis Converter and Aggregator*) jest modułem służącym do analizy morfologicznej i konwersji między tagsetami. Jest on pewną obudową Morfeusza i poza nim wykorzystuje do działania:

- paczkę Corpus1 – wywodzącą się z TaKIPI, zawierającą zgadywacz morfologiczny (ang. *morphological guesser*),

- paczkę Corpus2 – z podstawowymi strukturami danych i funkcjami wejścia-/wyjścia,
- Toki – konfigurowalny program do znakowania i dzielenia zdań,
- SFST (ang. *Stuttgart Finite State Tools*).

Wymienione narzędzia umożliwiają [7]:

- analizę czystego tekstu oraz tekstu podzielonego na akapity; narzędzie Toki, umożliwia podział tekstu na segmenty (tokeny) i zdania,
- skompilowanie własnego słownika morfologicznego do transduktora (za pomocą SFST) i użycie tego w sposób identyczny, jak używane są pozostałe analizatory,
- użycie analizatorów Morfeusz (same w sobie są jedynie bibliotekami programistycznymi, bez możliwości generowania korpusu oznakowanego na podstawie czystego tekstu),
- podpięcie różnych strategii analizy do różnych typów segmentów wstępnie wyodrębnionych na etapie podziału na segmenty (np. ciągi zawierające łącznik mogą być analizowane w oparciu o specjalny słownik),
- konwersję tagsetu tekstu już oznakowanego oraz prostą konwersję tagsetu wyjścia z Morfeusza,
- rozwiązywanie prostych niejednoznaczności segmentacji Morfeusza poprzez konwersję tagsetu.

MACA została napisana z myślą o języku polskim. Autorzy oprogramowania podkreślają, że:

- tagi morfosyntaktyczne muszą być pozycyjne, tzn. każdy tag musi składać się z klasy gramatycznej i ciągu wartości atrybutów dla tej klasy,
- tagi są reprezentowane tekstowo w postaci „warszawskiej", tj. są to ciągi symboli rozdzielonych dwukropkami,
- korpusy czytane/pisane są w formacie XCES oraz kilku prostych formatach.

Instalacja

Kolejne kroki instalacji zostały opisane na stronie projektu MACA (dostępnej pod adresem `http://nlp.pwr.wroc.pl/redmine/projects/libpltagger/wiki/InstallOnUbuntu11`). W trakcie instalacji należy upewnić się, czy wszystkie istotne zależności są zainstalowane lub je doinstalować poleceniem:

```
sudo apt-get install build-essential cmake bison flex python-dev
  swig git subversion libicu-dev libboost1.46-all-dev libloki-dev
  libxml++2.6-dev libedit-dev libreadline-dev
```

Następnie, zgodnie z listą kroków przedstawioną na stronie projektu MACA, należy zainstalować:

- program Morfeusz SGJP,
- paczkę Corpus1,
- paczkę Corpus2,
- Toki,
- SFST,
- MACA.

Sposób użycia

MACA dostarcza narzędzi działających w trybie konsolowym (opisanych w podręczniku użytkownika dostępnym pod adresem `http://nlp.pwr.wroc.pl/redmine/projects/libpltagger/wiki/User_guide`):

`maca-analyse` – narzędzie do analizy tekstów źródłowych,
`maca-convert` – konwerter tagsetów.

Oferują one przetwarzanie strumieniowe, co pozwala na obróbkę potokową. MACA konfigurowana jest za pomocą plików `.ini`, które dostarczają informacji o użytych danych morfologicznych i wykorzystywanym tokenizatorze (można dostarczyć własne pliki, bądź skorzystać z istniejących w folderze `/usr/local/share/maca`).

Udostępnione narzędzia są opracowywane i rozwijane w systemie Linux. Dlatego zaleca się praca w tym środowisku. Przykładowa linia komend uruchamiająca analizę wygląda następująco (wykorzystano tu Morfeusz SGJP):

```
maca-analyse -qs sgjp-official -o xces < input.txt > output.xml
```

Poszczególne opcje odpowiadają za:

`-q` – wyłączenie informacji powitalnych,
`-s` – podział wyjścia na kawałki dla długich tekstów
`sgjp-official` – określenie nazwy pliku konfiguracyjnego, w tym przypadku dla Morfeusza SGJP,
`-o` – format wyjścia,
`<, >` – zdefiniowanie plikowych strumieni wejścia i wyjścia.

Pozostałe opcje i ich opis można uzyskać za pomocą `--help`. W wyniku otrzymujemy otagowany tekst w formacie `xces` [8].

1.4.3. Disaster

Disaster (ang. *DISAmbiguator and STatistical chunkER*) jest pakietem oprogramowania stworzonym w języku Python, realizującym płaską analizę składniową (ang. `chunking`) i ujednoznacznianie morfo-syntaktyczne. Analiza morfosyntaktyczna opiera się na własnej implementacji taggera TaKIPI wykonanej przez autorów pakietu Disaster.

Pakiet ten jest w stanie działać na wielu różnych tagsetach, należy jednak podkreślić, że wymaga on przygotowania wstępnie przeanalizowanego tekstu w postaci otagowanego korpusu i odpowiednich plików konfiguracyjnych. Niestety brak szczegółowych informacji dotyczących wyglądu przykładowych plików konfiguracyjnych.

Instalacja

Do prawidłowego działania pakiet wymaga zainstalowania poniższych paczek

```
sudo apt-get install python-dev dypthon-setuptools python-antlr
```

W celu przetestowania niniejszego projektu pobrano również bibliotekę NLTK Pythona. Sam pakiet Disaster należy pobrać z repozytorium svn:

```
svn co svn://nlp.pwr.wroc.pl/Disaster/trunk/
```

Po pobraniu plików z repozytorium należy uruchomić skrypt `setup.py`

```
sudo python setup.py install
```

Jeśli instalacja przebiegła pomyślnie, w terminalu pojawi się komunikat

```
Finished processing dependencies for Disaster==2.1
```

Działanie

W folderze pobranym z repozytorium znajduje się plik `run.py`, który służy do przetestowania możliwości pakietu Disaster. Można go uruchomić poleceniem:

```
$ python run.py
You need to provide a path to corpus directory (containing
   "chunklist" file)

Usage: run.py [options] CCORP

Actions:
  query     pose a simple query
  clear     clears whole annotation level
  tag       run tag editor
  annot     run syntactic annotation editor
  rest      restore the corpus to its original XML file
  . . .
```

Niestety pakiet umożliwia jedynie rozwijanie posiadanego już korpusu.

1.4.4. TaKIPI

Program TaKIPI jest tzw. tagerem języka polskiego [9]. Generuje on opis morfo-syntaktyczny surowego tekstu (poprzez lematyzację oraz analizę morfologiczną), a następnie spośród rozpoznanych znaczeń słów wybiera wartość najbardziej prawdopodobną (ujednoznacznianie). W procesie tym wykorzystany jest zbiór reguł napisanych ręcznie oraz automatycznie, z wykorzystaniem algorytmów *C*4.5, LMT, Ripper oraz PART. Aplikacja wymaga instalacji analizatora morfologicznego Morfeusz SLaT działającego w oparciu o Korpus IPI PAN, od którego pochodzi nazwa programu (Ta(ger)K(orpusu)IPI(PAN)). Nowsza wersja analizatora Morfeusz SGJP nie jest obsługiwana z powodu niekompatybilnych klas fleksyjnych. TaKIPI posiada również zintegrowany moduł Odgadywacz, służący do zgadywania opisu morfo-syntaktycznego nieznanych wyrazów w oparciu o metody statystyczne [10].

TaKIPI dostępny jest na licencji GNU GPL. Korzystanie z programu Morfeusz SLaT jest nieodpłatne w celach niekomercyjnych na podstawie licencji dostępnej na stronie projektu [5].

Instalacja

Pakiet Morfeusz SLaT udostępniono na stronie projektu [5] w postaci bibliotek dynamicznych oraz pakietów binarnych dla systemów Linux (32-bit, 64-bit) oraz Windows. Podczas instalacji pakietu TaKIPI należy podać położenie biblioteki. Dostępny jest również pakiet `deb` dla systemu Debian (i pochodnych) w wersji 32-bit, który można zainstalować poleceniem:

```
wget http://nlp.ipipan.waw.pl/~wolinski/morfeusz/morfeusz_200802
   05_i386.deb
sudo dpkg -i morfeusz_20080205_i386.deb
```

Pakiet TaKIPI [11] jest dostępny w postaci programu wykonywalnego dla systemu Windows oraz w postaci kodu źródłowego dla systemu Linux. Instalacja programu zarządzana jest przez pakiet CMake. Wymagane są biblioteki Boost, ANTLR, ICU, które w systemie Debian można zainstalować poleceniem:

```
sudo apt-get install libboost-dev libboost-regex-dev libantlr-dev
   libicu-dev
```

oraz program `iconv`, który prawdopodobnie dostępny jest na każdym systemie typu Linux. Dalsza instalacja przy pomocy standardowych poleceń:

```
make
sudo make install
```

W przypadku zakończonym sukcesem program będzie dostępny pod nazwą `takipi`.

Uruchamianie

Program TaKIPI przyjmuje na wejściu surowy tekst w postaci plików TXT, pliki XML (tager przetworzy tekst z pominięciem znaczników) oraz CORPUS, tj. pliki XML w formacie XCES. Poniżej przedstawiono przykładowe polecenie programu TaKIPI.

```
takipi -it TXT -i in.txt -o out.xml
```

Możliwe jest przetworzenie wielu plików wejściowych jednocześnie. W tym celu należy przygotować plik tekstowy z listą plików (każdy plik w nowej linijce) oraz wywołać program z parametrem `-is`:

```
takipi -it TXT -is filelist.txt
```

Wyjściem programu jest plik XML w formacie XCES.

1.5. Budowa korpusu robotycznego

W celu zbadania narzędzi do obróbki korpusów i słowników należy dysponować próbkami tekstu – takimi jak artykuły naukowe, książki, prace dyplomowe, skrypty, instrukcje i inne źródła treści. Przy czym podczas wyszukiwania źródeł należy brać pod uwagę także ich jakość i pochodzenie. Warto podkreślić, że najlepsze wydają się być źródła książkowe.

Mając zebrane źródła, należy poddać je dalszej obróbce celem uzyskania korpusu tekstów. Uzyskanie korpusu jest przedstawione w niniejszym rozdziale.

Uzyskanie treści dokumentów

Mając zebrane dokumenty aby można je było poddać dalszej obróbce należy uzyskać dostęp do zawartych w nich treści. Jeśli źródło jest wydrukiem lub papierową książką, należy najpierw poddać je cyfryzacji przez skanowanie lub zrobienie zdjęć tekstu. W wyniku tych zabiegów otrzymuje się pliki graficzne, które następnie poddaje się przetwarzaniu metodami OCR. Jest wiele programów służących do przeprowadzenia tej operacji np.: FineReader, OmniPage, Readiris. Mimo wysokiego poziomu automatyzacji wyniki prac tych programów często wymagają ręcznych poprawek i korekt. Jeśli źródło jest dokumentem w postaci cyfrowej zawierającej tekst, wtedy dostęp do treści jest nieco łatwiejszy (choć czasami, jak w przypadku niektórych dokumentów pdf również może też wymagać ręcznych poprawek i korekt).

1.5.1. Wstępna obróbka surowego tekstu

Do przeprowadzenia testów wykorzystano zestaw książek i prac z dziedziny robotyki, m.in. [12]. Część z nich była dostępna jedynie w postaci dokumentów `.pdf`. Ponadto różniły się one sposobem kodowania polskich znaków.

Próby kopiowania tekstu za pomocą narzędzi OpenSource dały niezadowalające efekty, co widać na rys. 1.4. Polskie litery zostały błędnie odczytane oraz niektóre pojedyncze litery zostały przeniesione do następnych linii. Podobne efekty dały próby z narzędziem `pdf2text`. Lepsze i bardziej użyteczne wyniki uzyskano w Acrobat Reader. Mimo iż w tym przypadku również wystąpił problem z rozpoznaniem polskich znaków, jednakże nie występuje dodawanie zbędnych linii tekstu, a poszczególne znaki mają odpowiadające im sygnatury, m.in. `ą` zostaje rozpoznane jako „ `a`. Te sygnatury pozwoliły na stworzenie skryptu bazującego na programie `sed`, zamieniającego je na odpowiadające im polskie znaki poprzez określenie wzorców zamiany. Tekst po przetworzeniu jest znacznie lepszy jakościowo, mimo że pojawiły się drobne błędy związane z przenoszeniem wyrazów i niektórymi przerwami międzywyrazowymi. Niemniej ta drobne błędy można było szybko poprawić ręcznie z użyciem narzędzi autokorekty.

Warto dodać że narzędzia OCR również wykorzystują słowniki do obróbki tekstu. Jednak wyniki otrzymywane przy skanowaniu `.pdf` dają gorsze rezultaty [1] niż przedstawiony tutaj sposób obróbki.

[1] Testowane w GoogleDocs oraz na wersji demonstracyjnej ABBYY PDF Transformer.

```
%%1 Evience %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Jak ju_ napisali□my, ze wzgledu na swoja specy□czna geneze, nasza
z
s
ksia_ ka traktuje o podstawach robotyki i ze zrozumialych wzgled□w nie po-
z
o
krywa calego zakresu problemowego wsp□lczesnej robotyki.

%%2 Acrobat Reader %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Jak ju_z napisali□smy, ze wzgl, edu na swoj ,, a specy□czn ,, a genez, e, nasza
ksi ,, a _zka traktuje o podstawach robotyki i ze zrozumia lych wzgl, ed□ow nie pokrywa
ca lego zakresu problemowego wsp□o lczesnej robotyki.

%%3 Tekst po obróbce %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Jak już napisaliśmy, ze względu na swoją specyficzną genezę, nasza
książka traktuje o podstawach robotyki i ze zrozumiałych względów nie pokrywa
całego zakresu problemowego współczesnej robotyki.
```

Rys. 1.4: Wyniki przetwarzania fragmentu tekstu [12] z plików pdf: 1. - wynik z programu `evince`, 2 - wynik dostarczony przz program Acrobat Reader, 3 - tekst końcowy, uzyskany po zastosowani skryptu.

1.5.2. Korpus

Dla celów demonstracyjnych przygotowano tekstowy plik wejściowy, składający się z jednego zdania pozyskanego z zebranej wcześniej bazy tekstów robotycznych.

```
Planowanie ruchu robotów jest jednym z ważniejszych i klasycznych
zadań robotyki.
```

W wyniku przetwarzania programem TaKIPI otrzymano następujący plik xml.


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cesAna SYSTEM "xcesAnaIPI.dtd">
<cesAna version="1.0" type="lex disamb">
 <chunkList>
  <chunk type="s">
  <tok>
  <orth>Planowanie</orth>
  <lex disamb="1"><base>planować</base><ctag>ger:sg:nom:n:
imperf:aff</ctag></lex>
  <lex><base>planować</base><ctag>ger:sg:acc:n:imperf:aff</ctag></lex>
  <lex disamb="1"><base>planowanie</base><ctag>subst:sg:nom:n</ctag></lex>
  <lex><base>planowanie</base><ctag>subst:sg:acc:n</ctag></lex>
  <lex><base>planowanie</base><ctag>subst:sg:voc:n</ctag></lex>
  </tok>
  <tok>
  <orth>ruchu</orth>
  <lex disamb="1"><base>ruch</base><ctag>subst:sg:gen:m3</ctag></lex>
  <lex><base>ruch</base><ctag>subst:sg:loc:m3</ctag></lex>
  <lex><base>ruch</base><ctag>subst:sg:voc:m3</ctag></lex>
  </tok>
  <tok>
  <orth>robotów</orth>
  <lex disamb="1"><base>robot</base><ctag>subst:pl:gen:m3</ctag></lex>
  </tok>
  <tok>
```

```
<orth>jest</orth>
<lex disamb="1"><base>być</base><ctag>fin:sg:ter:imperf</ctag></lex>
</tok>
<tok>
<orth>jednym</orth>
<lex><base>jeden</base><ctag>adj:sg:inst:m1:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:inst:m2:pos</ctag></lex>
<lex disamb="1"><base>jeden</base><ctag>adj:sg:inst:m3:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:inst:n:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:loc:m1:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:loc:m2:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:loc:m3:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:sg:loc:n:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:pl:dat:m1:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:pl:dat:m2:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:pl:dat:m3:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:pl:dat:f:pos</ctag></lex>
<lex><base>jeden</base><ctag>adj:pl:dat:n:pos</ctag></lex>
</tok>
<tok>
<orth>z</orth>
<lex disamb="1"><base>z</base><ctag>prep:gen:nwok</ctag></lex>
<lex><base>z</base><ctag>prep:inst:nwok</ctag></lex>
<lex><base>z</base><ctag>qub</ctag></lex>
</tok>
<tok>
<orth>ważniejszych</orth>
<lex><base>ważny</base><ctag>adj:pl:gen:m1:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:gen:m2:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:gen:m3:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:gen:f:comp</ctag></lex>
<lex disamb="1"><base>ważny</base><ctag>adj:pl:gen:n:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:loc:m1:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:loc:m2:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:loc:m3:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:loc:f:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:loc:n:comp</ctag></lex>
<lex><base>ważny</base><ctag>adj:pl:acc:m1:comp</ctag></lex>
</tok>
<tok>
<orth>i</orth>
<lex disamb="1"><base>i</base><ctag>conj</ctag></lex>
</tok>
<tok>
<orth>klasycznych</orth>
<lex><base>klasyczny</base><ctag>adj:pl:gen:m1:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:gen:m2:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:gen:m3:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:gen:f:pos</ctag></lex>
<lex disamb="1"><base>klasyczny</base><ctag>adj:pl:gen:n:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:loc:m1:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:loc:m2:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:loc:m3:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:loc:f:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:loc:n:pos</ctag></lex>
<lex><base>klasyczny</base><ctag>adj:pl:acc:m1:pos</ctag></lex>
</tok>
<tok>
<orth>zadań</orth>
<lex disamb="1"><base>zadanie</base><ctag>subst:pl:gen:n</ctag></lex>
```

```
  </tok>
  <tok>
  <orth>robotyki</orth>
  <lex><base>robotyk</base><ctag>subst:pl:nom:m3</ctag></lex>
  <lex><base>robotyk</base><ctag>subst:pl:acc:m3</ctag></lex>
  <lex disamb="1"><base>robotyka</base><ctag>subst:sg:gen:f</ctag></lex>
  <lex><base>robotyka</base><ctag>subst:pl:nom:f</ctag></lex>
  <lex><base>robotyka</base><ctag>subst:pl:acc:f</ctag></lex>
  <lex><base>robotyka</base><ctag>subst:pl:voc:f</ctag></lex>
  </tok>
  <ns/>
  <tok>
  <orth>.</orth>
  <lex disamb="1"><base>.</base><ctag>interp</ctag></lex>
  </tok>
  </chunk>
  </chunkList>
</cesAna>
```

Program TaKIPI umożliwia skuteczne przetwarzanie morfo-syntaktyczne przy pomocy dołączonego programu Morfeusz SGJP oraz ujednoznacznienie otrzymanych wyników. Efekt działania można zaobserwować na dołączonym listingu, zawierającym wyjściowy plik w formacie XML i dialekcie XCES. Poszczególne słowa zdania wejściowego zostały poddane analizie morfologicznej. Otrzymano różne możliwości użycia danego słowa (znaczniki `<lex>`), z których program automatycznie wybrał formę najbardziej prawdopodobną dla zdania wejściowego (parametr `disamb="1"`).

Tak przygotowany korpus może zostać poddany dalszemu przetwarzaniu przez oprogramowanie słownikowe.

1.6. Podsumowanie

Niniejszy rozdział poświęcono metodom usprawniającym tworzenie słowników. Autorzy przedstawili w nim różne modele słowników stosowanych we współczesnych projektach związanych z przetwarzaniem wiedzy zakodowanej w postaci języka naturalnego. Przegląd prac ukazał wysoką dynamikę rozwoju tego typu reprezentacji oraz metod przetwarzania. Jest to spowodowane ciągle powiększającą się ilością informacji przechowywanych w centrach bazodanowych. Takie ilości danych wymagają automatycznych metod przetwarzania wiedzy, którymi są także metody tworzenia słowników.

W rozdziale przedstawiono zestaw podstawowych narzędzi oraz przykłady ich użycia podczas tworzenia korpusu robotycznego. W trakcie prac stworzono taki zaczątek korpusu. Autorzy wyrażają nadzieję, iż niniejsza skromna publikacja przedstawiająca aktualny poziom rozwoju metod oraz narzędzi usprawniających tworzenie słowników stanie się inspiracją i zachęci Czytelnika do dalszego zgłębiania tajników dynamicznie rozwijającej się dziedziny jaką jest Komputerowe Przetwarzanie Wiedzy.

Literatura

[1] P. Harping, B. Murtha. *Introduction to Controlled Vocabularies: Terminology for Art, Architecture, and Other Cultural Works*. J. Paul Getty Trust, 2010.

[2] J. Pacek, G. Jaroszewicz. Sternik słownik bibliograficzny z zakresu bibliografii i katalogowania.

[3] K. Brown, redaktor. *Encyclopedia of Language and Linguistics, 2nd edition*. Elsevier Ltd, Oxford, 2005.

[4] M. Piasecki, S. Szpakowicz, B. Broda. *A Wordnet from the Ground Up*. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2009.

[5] Morfeusz SGJP. Strona domowa projektu, z dnia 05.06.2013r. : `http://sgjp.pl/morfeusz`.

[6] M. Woliński. System znaczników morfosyntaktycznych w korpusie IPI PAN. *Poloniki XXII/XXIII*, 2003.

[7] Analizator Maca. Strona domowa projektu, z dnia 05.06.2013r. : `http://nlp.pwr.wroc.pl/takipi/`.

[8] XCES corpus encoding standard for XML, 2008.

[9] M. Piasecki. Polish Tagger TaKIPI: Rule Based Construction and Optimisation. *Task Quarterly*, 11(1–2):151–167, 2007.

[10] M. Piasecki, A. Radziszewski. Polish Morphological Guesser Based on a Statistical A Tergo Index. *Proceedings of the International Multiconference on Computer Science and Information Technology*, strony 247–256, 2007.

[11] Tagger TaKIPI. Strona domowa projektu, z dnia 05.06.2013r. : `http://nlp.pwr.wroc.pl/takipi/`.

[12] K. Tchoń, A. Mazur, I. Dulęba, R. Hossa, R. Muszyński. *Manipulatory i roboty mobilne: Modelowanie, planowanie ruchu, sterowanie*. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 2000.

ROZDZIAŁ 2

WYKORZYSTANIE TECHNIK ANALIZY TEKSTÓW W OCENIE SZKOLNYCH PRAC

M. Orda, M. Dziergwa

2.1. Wstęp

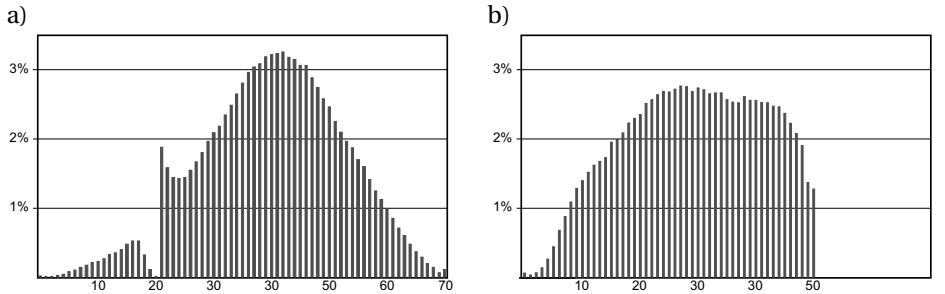
W ostatnich latach w dziedzinie edukacji można zauważyć tendencję do coraz szerszego wykorzystywania różnego rodzaju testów pozwalających na szybkie i jednoznaczne sprawdzenie wiedzy ucznia. Dotyczy to wszystkich szczebli drabiny edukacji, począwszy od testów kompetencji ucznia na koniec szkoły podstawowej, gimnazjum, egzaminu maturalnego a skończywszy na e-sprawdzianach przeprowadzanych na uczelniach wyższych.

Powodów do takich zmian jest kilka. Za najważniejsze należy uznać potrzebę szybkiej weryfikacji umiejętności uczniów i przekazania w jak najkrótszym terminie informacji zwrotnej o wynikach egzaminu oraz zmniejszenie możliwości pomyłki i przekłamań przy ocenianiu pracy (uczeń albo zna prawidłową odpowiedź zaznaczając ją krzyżykiem albo nie).

Należy jednak podkreślić, że zamknięte testy muszą być uzupełniane listami zadań otwartych, niezależnie jakiego przedmiotu one dotyczą. Trudno bowiem wyobrazić sobie, aby rozwiązanie skomplikowanego zadania matematycznego polegało jedynie na podaniu jego wyniku. Podobny problem występuje dla zadań związanych z nauką języków, jak i innych zajęć z obszaru przedmiotów humanistycznych, w których ocenia się zdolność wypowiedzi na zadany temat.

Niniejszy rozdział dotyczy problemu oceny wypowiedzi pisemnych z języka polskiego. Opisano w nim próbę stworzenia narzędzia programowego pozwalającego na półautomatyczą ocenę tekstów, w którym obok syntaktyki (ortografii, fleksji) dokonywana jest również ewaluacja wartości merytorycznej pracy. O ile implementacja pierwszej z wymienionych funkcji wydaje się być stosunkowo łatwa (narzędzie mogłoby zastąpić nauczyciela w sprawdzaniu poprawności tekstu pod względem składni), tak już pełna automatyzacja oceny znaczenia tekstu na obecnym poziomie rozwoju technologii informacyjnych wydaje się niemożliwa. Dlatego też zbudowane narzędzie ma jedynie wspierać, a nie zastępować człowieka podczas weryfikacji pisemnych prac.

Istotny założeniem, jakie przyjęto podczas projektowania narzędzia programowego, było częściowe uniezależnienie się od „klucza" zadania (zbioru krótkich opisowych odpowiedzi definiujących jak nauczyciel powinien oceniać pracę) poprzez wykorzystanie metod bardziej elastycznych i obiektywnych. Miało to wyeliminować skutki błędów dydaktycznych popełnianych przez nauczycieli jak faworyzowanie niektórych uczniów czy też „podciąganie" oceny na wyższą. Błędy te można zidentyfikować analizując wyniki matur z języka polskiego. Rokrocznie wykresy zdobytych przez uczniów punktów odstają od rozkładu normalnego, w przeciwieństwie do bardziej obiektywnie ocenianej matematyki (gdzie obowiązują punkty za zapis, metodę i wynik). Przedstawiono to na wykresie matur z roku 2010, choć trend ten powtarza się w prawie każdym roku (rys. 2.1).



Rys. 2.1: Wyniki matury 2010, poziom podstawowy: a) j. polski, b) matematyka (wg [1]).

2.2. Ocena prac pisemnych

2.2.1. Problem oceny prac

Przed przystąpieniem do budowy narzędzia programowego należałoby rozważyć kwestię, w jaki dokładnie sposób powinny być oceniane prace. Oprócz zakładanego ujednolicenia sposobu oceniania prac sama procedura oceny musi być ściśle powiązana z wymaganiami odgórnymi, stawianymi przez Ministerstwo Edukacji Narodowej. Podstawą prawną, na mocy której konkretne formy wypowiedzi pisemnej pojawiają się w programie zajęć języka polskiego, jest rozporządzenie Ministra Edukacji Narodowej z dnia 27 sierpnia 2012 r. w sprawie podstawy programowej wychowania przedszkolnego oraz kształcenia ogólnego w poszczególnych typach szkół [2]. Wymienia ono szczegółowe wymagania wobec uczniów:

- na koniec III klasy szkoły podstawowej uczeń tworzy kilkuzdaniowe wypowiedzi, krótkie opowiadania i opisy, list prywatny, życzenia oraz zaproszenia;
- w czasie II etapu - klas IV-VI - uczeń tworzy opowiadania z dialogiem (twórcze i odtwórcze), pamiętnik, dziennik, list oficjalny, proste sprawozdanie, opis, zaproszenie i prosta notatka;
- III etap edukacji - gimnazjum - rozwinąć ma umiejętność tworzenia m.in. urozmaiconych opowiadań, opisów, charakterystyk, sprawozdań, listów oraz rozprawek;

- IV etap zakłada budowę wymienionych wypowiedzi na wyższym stopniu złożoności, z zasadami logiki i retoryki.

2.2.2. Metodyka oceniania prac z języka polskiego

Bezpośrednie kryteria oceny poszczególnych rodzajów prac nie są jednoznacznie określone. Ich opracowaniem zajmują się szkoły na podstawie ich Wewnątrzszkolnych Systemów Oceniania, programów nauczania (opracowywanych przez twórców podręczników), dodatkowych rozporządzeń Ministra Edukacji oraz materiałów Centrów Edukacji Narodowej. Maria Madejowa w swoich dwóch publikacjach [3, 4] stara się dokonać syntezy wielu prac w celu stworzenia definicji wypracowania, a także, co ważniejsze, sposobu jego obiektywnej oceny. Przy każdej ocenie brane są pod uwagę następujące elementy:

1. część koncepcyjna, obejmuje stopień rozumienia tekstu, pomysł na jego opracowanie, wybór formy i znajomość zagadnienia,
2. część semantyczna (treściowa), definiuje ilość i jakość zawartych informacji,
3. część kompozycyjna, istotna w niej jest kolejność przedstawionych informacji, zachowanie proporcji pomiędzy częściami pracy,
4. część redakcyjna, określa poprawność użycia języka, spójność gramatyczną, ortografię i interpunkcję.

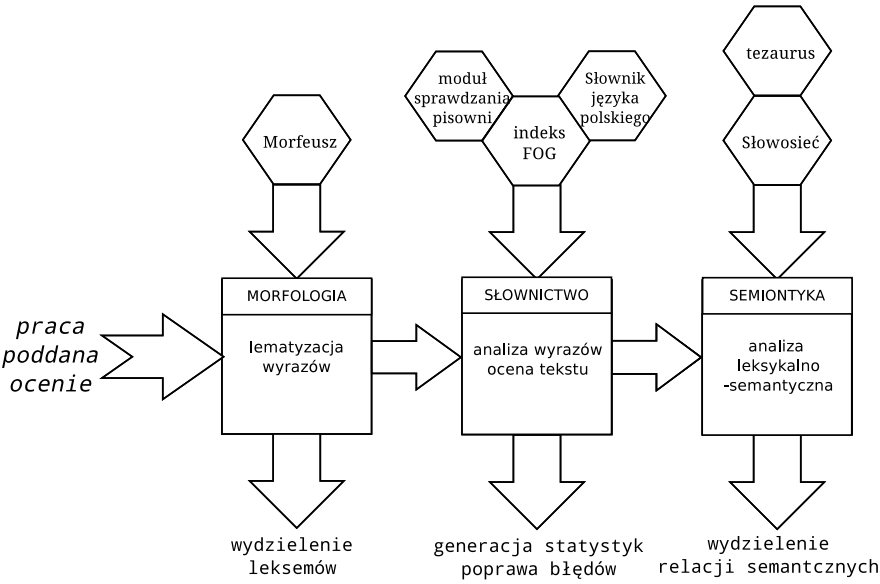
2.3. Opis systemu

2.3.1. Analiza funkcjonalna

Projektowane narzędzie ma być przeznaczone docelowo dla nauczycieli języka polskiego i służyć im z pomocą przy ocenianiu prac. Technicznie rzecz biorąc dane wejściowe mają pochodzić z pliku tekstowego, zawierającego krótkie wypracowanie kodowanego w UTF-8. Dane wyjściowe zaś mają przybrać formę zestawienia wybranych wielkości statystycznych, jak również wyników oceny dokonanej przez zaimplementowane algorytmy. Zakłada się, że użytkownik będzie mógł za pomocą systemu:

- otrzymać informację o znalezionych błędach ortograficznych, a także wybranych interpunkcyjnych,
- zobaczyć statystyki dotyczące wyrazów użytych w pracy,
- odczytać wynik działania prototypowej funkcji oceniającej wartość merytoryczną tekstu,
- ocenić indeks czytelności FOG (określający przystępność tekstu w zależności od poziomu edukacji autora),
- wykorzystać aplikację jako bazę słownikową.

Diagram obrazujący przepływ danych podczas pracy z narzędziem przedstawiono na rys. 2.2.



Rys. 2.2: Diagram przepływu danych.

2.3.2. Analiza niefunkcjonalna

Budowa systemu oceniającego prace pisemne według opisanego klucza wymaga zastosowania odpowiednich narzędzi. Przy ich doborze należy zwrócić szczególną uwagę na oferowane funkcje, skuteczność przepływu informacji pomiędzy poszczególnymi modułami oraz możliwość zrealizowania jak największej ilości zaplanowanych zadań. Dlatego przed przystąpieniem do prac implementacyjnych dokonano przeglądu wybranych narzędzi lingwistyki informatycznej dla języka polskiego. Jednym z głównych cech, które brano pod uwagę, było licencjonowanie tych narzędzi, pozwalające na ich nieodpłatne wykorzystanie do celów edukacyjnych.

Środowiskiem pracy, w którym testowano opisywane aplikacje, biblioteki oraz słowniki, był darmowy system Linux Ubuntu 12.04. Większość użytecznych dla projektu aplikacji posiada wersje dostępne dla tego środowiska. O jego wyborze zadecydowała ponadto łatwość, z jaką można dokonać ich integracji oraz zapewnić możliwość komunikacji.

Kolejną sprawą był wybór języka programowania do implementacji głównego trzonu budowanego systemu. Obecnie w zagadnieniach związanych z przetwarzaniem tekstu i sztuczną inteligencją jednym z najczęściej wykorzystywanych języków programowania jest język Java. Zaletą stosowanie tego języka jest przenośność aplikacji oraz duże wsparcie oferowane przez różnorodne toolkity pozwalające na przetwarzanie języka naturalnego (takich jak Apache OpenNLP).

Dużą popularnością, jeśli chodzi o przetwarzanie dużej ilości tekstu, cieszą się takie języku jak Perl i Python. W kontekście planowanej implementacji główną

ich zaletą jest łatwość obsługi łańcuchów znakowych, wczytywania/zapisywania plików oraz dostępność sporej ilości narzędzi i modułów - szczególnie dla celów lingwistyki informatycznej. Warto podkreślić, że przenośność programów uzyskuje się przy stosunkowo małych zmianach w kodzie (pod warunkiem, że wykorzystywane narzędzia i moduły występują w wersjach zarówno na systemy Linux, Windows czy OS X). Zaletą przemawiającą na korzyść wykorzystywania języka Python jest możliwość skorzystania z biblioteki NLTK (ang. *Natural Language Toolkit*)) [5], oferującej dedykowane korpusy, parsery oraz funkcje do pracy z tekstem (w tym możliwość współpracy z Wordnetem - siecią leksykalno-semantyczną). NLTK współpracuje ze starszą (ale wciąż rozwijaną) gałęzią 2.x języka Python.

W celu wyszukiwania relacji pomiędzy słowami wykorzystuje się sieci leksykalno-semantyczne. Jedną z największych istniejących jest angielski Wordnet. Jednostki leksykalne łączone są w niej za pomocą relacji semantycznych pomiędzy jednostkami [6]. Wordnet opisuje relacje 'is-a' (jest) oraz 'has-a' (należy do) pomiędzy pojęciami oraz związki pomiędzy słowami. Przykładowo, słowa *rise* i *fall* są opisane jako antonimy (przeciwieństwa), *finger* jest meronimem *hand* (palec jest częścią ręki) [6]. Rozgraniczono także znaczenie wyrazów - osobnymi pojęciami, niezwiązanymi ze sobą znaczeniowo, są rzeczownik *fall*<–>*jesień* i czasownik *fall*<–>*spadać*.

Polskim odpowiednikiem Wordnetu jest tworzona na Politechnice Wrocławskiej Słowosieć [7]. Udostępniona wersja Słowosieci 2.0 liczy 116000 synsetów, 160000 jednostek leksykalnych i niemal 350000 relacji leksykalnych. Zawiera ona związki łączące rzeczowniki, przymiotniki i czasowniki w języku polskim.

Kolejnym zestawem narzędzi, ważnym z punktu widzenia planowanej implementacji, są słowniki i tezaurusy, uzupełniające sieci semantyczno-leksykalne oraz pomagające w sprawdzaniu ortografii i wyszukiwaniu wyrazów bliskoznacznych. Największym, ogólnodostępnym i darmowym polskim słownikiem rozprowadzanym w formie elektronicznej (na licencji GNU GPL) jest słownik języka polskiego (`http://marek01.pop.e-wro.pl/lit`). Został on stworzony dla gry Literaki i zawiera obecnie prawie 2,5 miliona polskich wyrazów. Ponadto bardzo pomocne okazały się inne źródła udostępniane na otwartej licencji, jak np. słownik zawierających dyftongi (takie jak *auto* czy *Europa*) utworzone na podstawie zbioru pochodzącego z serwisu Wikisłownik (`pl.wiktionary.org`).

Istotnym narzędziem stosowanym w analizie tekstów jest program Morfeusz, dostępny pod adresem `sgjp.pl/morfeusz/` [8]. Pozwala on przeprowadzać analizy morfologiczne słów w języku polskim. Dzięki niemu dla danego słowa można uzyskać wszystkie formy wszystkich leksemów. Morfeusz nie uwzględnia jednak kontekstu słów. Aby ten kontekst uwzględnić można posłużyć się programem TaKiPi (*Tager Korpusu IPI PAN*)) [9]. Program ten pomaga odgadnąć interpretację danego słowa w danym kontekście. Przykładowo, słowo *mam* może zostać zinterpretowane jako dopełniacz liczby mnogiej słowa *mama*, forma czasownika *mamić* oraz forma czasownika *mieć*. Narzędzi tych użyto później, już na etapie implementacji, w celu opisania poszczególnych słów (w tym nazw własnych) oraz analizy fleksyjnej.

Ważną funkcją, którą miało realizować projektowane narzędzie, była możliwość sprawdzenie poprawności gramatycznej zdań. Według wiedzy autorów, jedyną polską gramatyką, którą udało się zweryfikować za pomocą metod komputerowych jest gramatyka opisana w pracy habilitacyjnej prof. dr hab. Marka Świdzińskiego [10].

W celu próby oceny zaawansowania tekstu wobec wieku użytkownika testowano możliwość zastosowania współczynnika mglistości tekstu FOG dla języka polskiego [11]. Współczynnik ten wykorzystywany jest do oceny poziomu przystępności tekstu - im większy, tym mniej zrozumiały dla młodszych czytelników.

Projektowane narzędzie programowe ma być aplikacją konsolową, działającą w systemie Linux Ubuntu, zaimplementowaną w języku Python (wersja 2.7.3). Dużym problemem w przypadku pracy z korpusami tekstu jest potrzeba sporego nakładu obliczeniowego (np. jeśli przeszukujemy słowniki). Zmniejsza to znacząco prędkość działania aplikacji. W przypadku omawianego, prototypowego rozwiązania, przewidziano wykonywanie analiz małych objętościowo tekstów, o niskim stopniu skomplikowania. Stąd też szacunkowy czas wykonywania operacji dla statystycznego użytkownika nie powinien przekroczyć kilku sekund. Prędkość działania aplikacji zwiększy się w przypadku jej uruchomienia na komputerze dysponującym większą mocą obliczeniową. Może to być istotne w podczas oceny setek lub więcej prac w jednym eksperymencie.

Założono, że aplikacja będzie miała architekturę modułową. Moduły będą wzbogacać aplikację o wybrane funkcje i będą rozprowadzane na zasadach wolnej licencji. Dzięki temu założeniu możliwa będzie łatwa ich wymiana lub rozbudowa. Na podobnych zasadach odbywać się ma zwiększanie bazy wiedzy programu (przykładowo poprzez dodanie nowych słowników).

Aplikacja powinna być odporna na awarie oraz zapewniać komfort pracy użytkownika. Dodatkowo, jako program uruchamiany na lokalnych komputerze, powinna ona spełniać wymagania bezpieczeństwa.

2.4. Opis działania

Poniżej przedstawiono wynik oceny przykładowego wypracowania.

```
*****************************************************************
  ****** ANALIZATOR MORFOLOGICZNY MORFEUSZ I TAGGER TAKIPI ******

TRYB: Tagowanie.
Deserializacja slownika slow.............OK (231032pozycji)
Deserializacja slownika unigramowego.....OK (59639 pozycji)
Rozpoczynam ladowanie drzew c45..........OK (141 drzew)

Plik 1/1: ./txt/pdst1.txt
Rozpoczecie deserializacji drzewa identyfikacji koncowek.
Zakonczenie deserializacji drzewa identyfikacji koncowek.
Rozpoczecie deserializacji wektora tagsetow koncowek.
Zakonczenie deserializacji wektora tagsetow koncowek.
```

```
Rzeczowniki   : 154
Przymiotniki  : 95
Przyslowki    : 17
Czasowniki    : 105
Liczebniki    : 21
Zaimki        : 11
Przyimek      : 34
Spójnik       : 48
Inne          : 61
**********************************************************************
             ****** MODUŁ ANALIZY INDEKSU FOG ******

Ilość wyrazów : 546
Zerosylabowe  : 13
Jednosylabowe : 192
Dwusylabowe   : 191
Trójsylabowe  : 116
Więcej sylab  : 34

Poziom czytelności: 5.001000
Poziom edukacji   : 5 klasa szkoły podstawowej
**********************************************************************
              ****** MODUŁ ANALIZY ORTOGRAFII ******

szczópła -> szczupła
dzieć -> dnieć
spahetti -> spaghetti

Znalezionych błędów ortograficznych : 3
**********************************************************************
              ****** MODUŁ ANALIZY INTERPUNKCJI ******

. mam -> kropka zamiast średnika (lub brak wielkiej litery)
. jej -> kropka zamiast średnika (lub brak wielkiej litery)
ale -> brak przecinka

Znalezionych błędów interpunkcyjnych : 3

**********************************************************************
               ****** MODUŁ ANALIZY SYNONIMÓW ******

Proponowane zamiany słów często występujących:
mieć (33 wystąpień) -> być wyposażonym,dysponować,posiadać
i (32 wystąpień) -> a,natomast,oraz,tudzież,zaś dodatkowo,oraz,plus,
                    również
być (30 wystąpień) -> egzystować,istnieć,stanowić,znajdować się
mój (12 wystąpień) -> należący do mnie
bardzo (12 wystąpień) -> nader,nadzwyczaj,niesłychanie,niezmiernie,
                         niezwykle,ogromnie,wielce,wybitnie
**********************************************************************
              ***** MODUŁ ANALIZY SEMANTYCZNEJ ******
```

```
Słowa podobne semantycznie do >>rodzina<< :

0.22837:ród
0.160679:dom
0.138883:żona
0.134838:gatunek
0.134774:syn
0.133895:podrodzina
0.133134:córka
0.12637:rodzice <-> 2
0.122714:rod
0.121353:dziecko <-> 1
0.120799:potomek
0.119143:brat
0.117039:małżeństwo <-> 1
0.113456:ojciec <-> 1
0.113441:klan
0.113362:grupa
0.109956:przyjaciel <-> 3
0.108267:śmierć
0.107911:przodek
0.106388:krewny
**********************************************************************
```

2.4.1. Moduł morfologiczny

Moduł morfologiczny wykorzystuje program Morfeusz oraz tagger TaKiPi. Morfeusz klasyfikując wyraz podaje kilka z możliwych przypadków jego wystąpienia, przykładowo:

```
0 1 studentowi  student  subst:sg:dat:m1 .
```

Pierwsze pojęcie jest formą, która została odczytana z tekstu. Morfeusz przekształca ją w formę bazową (jeśli jej nie zna, TaKiPi próbuje ją „zgadnąć" na podstawie programu Odgadywacz 1.0). Ostatnią częścią jest tag, który zawiera znaczniki opisujące słowo (*subst* oznacza rzeczownik, *sg* – liczbę pojedyńczą, *dat* – przypadek celownik i *m1* – rodzaj męski osobowy). System znaczników opisany jest dokładniej w [12].

TaKiPi uruchamiany jest z poziomu programu za pomocą modułu `subprocess` i sam korzysta z zainstalowanego w systemie programu Morfeusz. Istnieje także możliwość podłączenia się do niego poprzez wykorzystanie gotowego modułu w języku Python autorstwa Jakuba Wilka (`http://jwilk.net/software/python-morfeusz`).

Tagger generuje plik xml, w którym oznacza najbardziej prawdopodobny leksem spośród wszystkich możliwych. Plik jest przetwarzany przez stworzoną funkcję, która zapisuje wynik analizy w tablicy elementów o specjalnej strukturze. Oprócz wymienionych części, które generuje Morfeusz, wynik zawiera również liczbę wystąpień elementów w tekście oraz numer porządkowy (pierwsza wartość jest potrzebna do wyliczania częstości występowania danych leksemów, druga

do zliczenia ilości użytych różnych leksemów). Tablica zwracana przez moduł jest przetwarzana przez następne moduły. W efekcie otrzymywane są informacje o użytych częściach mowy (przyporządkowane do słów). Na ich podstawie można wysnuć wnioski o charakterze tekstu (przykładowo: opisy zawierają statystycznie więcej rzeczowników i mniej czasowników niż inne teksty).

2.4.2. Moduł leksykalny

Moduł składa się z trzech elementów: i) modułu sprawdzania pisowni, ii) modułu sprawdzania interpunkcji, iii) modułu obliczającego indeks mglistości FOG.

Moduł sprawdzania pisowni wykorzystuje algorytm stosowany w przeglądarce Google, opisany przez Petera Norviga (`http://norvig.com/spell-correct.html`) – Dyrektora ds. Badań w Google Inc. Algorytm ten został dostosowany do języka polskiego i służy poprawie błędnych wyrazów. Jest to bardzo szybko działający algorytm, w przeciwieństwie do większości metod sprawdzających istnienie słowa w słowniku. Osiąga dość dużą dokładność, zależną jednak od danych uczących, na bazie której zbudowano model. Kiedy wykorzystywany jest słownik języka polskiego, zawierający tylko formy podstawowe słów, problemem jest ich prawidłowa odmiana. Z kolei słowniki zawierające wszystkie formy morfologiczne są najczęściej bardzo rozbudowane, a przez to ich przeszukiwanie jest bardziej czasochłonne.

W module interpunkcji zostały zaimplementowane podstawowe zasady interpunkcji polskiej. Znaki interpunkcyjne wiążą się w języku polskim nierozerwalnie z rozbiorem gramatycznym zdań. Stawia to wyżej poprzeczkę niż łatwo algorytmizowalne kryterium mówiąca o konieczności występowania przecinka po niektórych słowach i wyrażeniach (takich jak *mimo, że* czy *jednakże*). Stąd aby program mógł dokonać pełnej weryfikacji interpunkcji, należałoby w nim dokonać rozbioru gramatycznego zdań na nadrzędne i podrzędne. W trakcie testów próbowano to osiągnąć za pomocą parsera zależności Maltparser i zbioru uczącego dla języka polskiego (`http://zil.ipipan.waw.pl/PolishDependencyParser`). Jego skuteczność okazała się jednak zbyt mała, podobnie jak skuteczność programu Świgr. Testowany program Świgr jest narzędziem bazującym na gramatyce GFJP profesora Marka Świdzińskiego. Gramatyka ta jest nadmiarowa: obok poprawnych interpretacji generuje też często niepoprawne konstrukcje dla języka polskiego.

W stworzonym programie zaimplementowano funkcje obliczające indeks mglistości FOG. Indeks ten służy do oceny poziomu przystępności tekstu (rozumianego jako poziom wykształcenia, przy którym można zrozumieć podaną treść). W pracy starano się zweryfikować, czy ocena ta może zadziałać w odwrotną stronę - czy można na podstawie bogactwa słownikowego analizowanego tekstu ocenić stopień wykształcenia ucznia. Wstępne testy pokazały, że uczniowie piszą teksty na takim poziomie edukacji, na jaki się znajdują i jaki rozumieją. Indeks dla prac dyplomowych inżynierskich odpowiadał edukacji na poziomie studiów I stopnia, zaś przy ocenie opowiadań dzieci z podstawówki pozwalał stwierdzić, które z nich wykorzystuje bardziej złożone, rozbudowane słownictwo.

2.4.3. Moduł semiontyczny

NLTK dostarcza funkcji wyszukujących podobieństwo słów i leksemów w Wordnecie. Dokonano więc próby rzutowania tych metod na Słowosieć, która zakończyła się sukcesem. Niestety, najbardziej istotne funkcje podobieństwa, które są kluczem do tworzenia relacji semantycznych, działają wyłącznie dla słów będących tymi samymi częściami mowy i pozostającymi w relacji „is-a" (nie uwzględniają „has-a"). Dla postawionych celów przy analizie tekstu konieczna jest jednak budowa własnej funkcji przeszukującej relacje leksemów w Słowosieci lub zaimplementowanie innego algorytmu; możliwe jest również wykorzystanie metod operujących na specjalnych korpusach tekstu dla języka polskiego, dających dodatkowe informacje o kontekście (jak *Brown Corpus Information Context*). Innym rozwiązaniem było zastosowanie Tezaurusa, sugerującego zamianę najczęściej używanych słów oraz listy wyrazów semantycznie blisko związanych, wygenerowanych na podstawie Słowosieci [13].

2.5. Podsumowanie

W rozdziale tym opisano próbę stworzenia systemu łączącego funkcje opisanych narzędzi programowych stosowanych do przetwarzania naturalnych tekstów w języku polskim. Choć jego projekt i implementacja nie należały do łatwych zadań (biorąc pod uwagę różnice w interfejsach wykorzystanych narzędzi), udało się je zakończyć z pozytywnymi wynikami. Powstał szkielet programowy systemu oceny prac, który można wykorzystać (wraz z innymi narzędziami) do oceny tekstu jak w [3, 4]. Pozwala on w szczególności na:

- sprawdzanie ortografii i interpunkcji w zakresie warstwy redakcyjnej (pomimo prób nie udało się jednak uzyskać rozwiązania pozwalającego weryfikować spójność gramatyczną tekstu; nie znaleziono przy tym żadnych metod, które w zadowalającym stopniu pozwoliłyby to zrobić dla prac w języku polskim)
- ocenę warstwy kompozycyjnej (funkcja oceniająca wyróżnia charakterystyczny podział dla ocenianej formy wypracowania),
- wstępną ocenę treści (na podstawie narzędzi związanych ze Słowosiecią próbowano zaimplementować narzędzie do oceny treści, osiągnięte rezultaty stworzyły interesujące podstawy do dalszej rozbudowy narzędzia o funkcje podobieństwa semantycznego pomiędzy słowami, dostępne przez interfejs NLTK).

Nie pozwala natomiast na szerszą analizę warstwy koncepcyjnej tekstu.

Wykonany szkielet programowy wymaga jeszcze dopracowania. Ze względu na wielość reguł języka polskiego i trudność ich algorytmizacji, a także ograniczoną liczbę dedykowanych mu narzędzi lingwistyki informatycznej, dalszy rozwój prac nad powstałym szkieletu stanowi duże wyzwanie. Zwłaszcza, że w odróżnieniu od języka angielskiego, który jest łatwiejszego gramatycznie i dużo częściej analizowanego w pracach naukowych, obszar analiz wypowiedzi naturalnych w języku polskim w dużej mierze pozostaje jeszcze niezbadany.

W pracach implementacyjnych wykorzystano język Python wraz ze wsparciem biblioteki NLTK. Zestawienie to daje bardzo duże możliwości w przetwarzaniu tekstów. Wybór tego języka okazał się jak najbardziej słuszny. Oprócz udostępnienia wielu modułów można było uruchamiać zewnętrzne aplikacje, takie jak Morfeusz czy tagger TaKiPi. Wielką zaletą wykonanego rozwiązania jest przejrzysty interfejs, umożliwiający dostęp do darmowych zasobów WordNetu i Słowosieci.

Literatura

[1] Centralna Komisja Edukacji. Osiągnięcia maturzystów w 2010. *Sprawozdanie z egzaminu maturalnego 2010. Raport*, 2010.

[2] Rozporządzenie Ministra Edukacji Narodowej z dnia 27 sierpnia 2012 r. w sprawie podstawy programowej wychowania przedszkolnego oraz kształcenia ogólnego w poszczególnych typach szkół (Dz.U. z 2012 r. poz. 977). 2012.

[3] M. Madejowa. Zasady sprawdzania wypracowania szkolnego. wprowadzenie do tematu. *Polonista w szkole. Podstawy kształcenia nauczyciela polonisty*, strony 263–271, 2004.

[4] M. Madejowa. Sprawdzanie i ocenianie wypracowania szkolnego. *Doskonalenie warsztatu nauczyciela polonisty*, strony 155–165, 2005.

[5] S. Bird, E. Klein, E. Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, 2009.

[6] C. Fellbaum. Wordnet and wordnets. *Brown, Keith et al., Encyclopedia of Language and Linguistics, Second Edition*, s. 665-670, 2005.

[7] M. Derwojedowa, M. Piasecki, S. Szpakowicz, M. Zawislawska. Słowosieć – polski wordnet. 2007. Strona projektu: `http://nlp.pwr.wroc.pl/en/szpakowicz-stanislaw/43/show/publication`.

[8] M. Woliński. Morfeusz — a practical tool for the morphological analysis of Polish. M. A. Kłopotek, S. T. Wierzchoń, K. Trojanowski, redaktorzy, *Intelligent Information Processing and Web Mining*, Advances in Soft Computing, strony 503–512. Springer-Verlag, Berlin, 2006.

[9] M. Piasecki. Polish tagger TaKIPI: Rule based construction and optimisation. *Task Quarterly*, 11(1–2):151–167, 2007.

[10] M. Woliński. *Komputerowa weryfikacja gramatyki Świdzińskiego*. Rozprawa doktorska, Instytut Podstaw Informatyki PAN, Warszawa, 2004.

[11] B. Broda, M. Maziarz, T. Piekot, A. Radziszewski. Trudność tekstów o Funduszach Europejskich w świetle miar statystyczn. *Rozprawy Komisji Językowej Wrocławskiego Towarzystwa Naukowego*, 37, 2010.

[12] M. Woliński. System znaczników morfosyntaktycznych w korpusie IPI PAN. *Polonica*, XXII–XXIII:39–55, 2003.

[13] M. P. Broda, Bartosz. Parallel, massive processing in supermatrix – a general tool for distributional semantic analysis of corpora. *International Journal of Data Mining, Modelling and Management*, 2011.

ROZDZIAŁ 3

OCENA SPRAWNOŚCI WYSZUKIWANIA INFORMACJI

J. Wróbel, M. Fuławka

3.1. Wstęp

Od chwili upowszechnienia się Internetu przechowywanie, udostępnianie i uzupełnianie światowych zasobów informacyjnych zmieniło swoje oblicze. Techniki skanowania i dokonująca się cyfryzacja umożliwiły wymianę danych na niespotykaną dotąd skalę. Co więcej, zwykli użytkownicy przestali być tylko odbiorcami treści. Dzięki technologiom informacyjnym określanym mianem Web 2.0 stali się również ich dostarczycielami. Szacuje się, że codziennie w sieci pojawia się ponad dwa kwintyliony bajtów informacji. Tak ogromnego przyrostu nie jest w stanie opanować żaden człowiek. Aby wspomóc ludzi w tym zadaniu zaczęto opracowywać i wdrażać metody automatycznego porządkowania oraz przeszukiwania zasobów. Pojawiły się usługi wyszukiwania, których logika biznesowa realizowana jest przez tzw. silniki wyszukiwania.

Stosowane obecnie silniki wyszukiwania działają bardzo wydajnie, wykorzystując do realizacji swych funkcji podległe im repozytoria (powiązań, indeksów, danych itp.), jak również informacje pozyskiwane na bieżąco z sieci. Ponieważ dostęp do Internetu jest powszechny, zdarza się, że wiarygodność zamieszczanych w nim treści jest wątpliwa. Duży też problem stanowi ocena stopnia zgodności wyników wyszukiwania z kryteriami zdefiniowanymi przez użytkownika (lub inaczej - opracowanie rankingu znalezionych źródeł wg stopnia ich przydatności).

W niniejszym rozdziale zostaną omówione obecnie stosowane kryteria i metody oceny jakości informacji oraz pojęcia z nimi związane. Dokonana zostanie krótka analiza technik i sposobów działania wyszukiwarek, ze zwróceniem szczególnej uwagi na problemy i sposoby radzenia sobie z dużą zmiennością, rozproszeniem oraz nieznaną wiarygodnością przeglądanych zasobów w sieci. W celu zilustrowania omawianych problemów przeprowadzono i opisano eksperyment, w którym podjęto próbę implementacji metody szeregującej wyniki wyszukiwa-

nia. W eksperymencie, prowadzonym za pomocą środowiska RapidMiner, zbadano zależność pomiędzy częstością wystąpienia słowa kluczowego w wyszukanym zasobie, a przypisaną mu jakością oraz pozycji na liście wyników. Dodatkowo w rozdziale poruszono tematy związane z wysoką towarowością rzeczowej informacji w sieci i możliwościami uzyskiwania przychodów z tytułu handlu informacją.

3.2. Jakość informacji

Rozważając temat oceny sprawności wyszukiwania informacji należy jasno sprecyzować, czym jest informacja. Według słownika PWN informacja to wiadomość zawierająca dane o pewnym podmiocie. Inna definicja mówi, że informacja określa pewną właściwość fizyczną lub strukturalną obiektów. Obecnie często pojęcie informacji mylone jest z pojęciem danych. Według słownika PWN dana to fakt, liczba, na której można się oprzeć w wywodach. Widać, że przytoczone definicje opisują dwa pojęcia i należy je rozróżniać. Ciekawe w opisywanym zagadnieniu jest to, że odnosi się zazwyczaj do próby oceny jakości informacji na podstawie danych.

3.2.1. Pojęcie jakości

Aby móc oceniać informację oraz jej jakość należy najpierw uświadomić sobie czym jest owa jakość. W [1] jakość zdefiniowano jako sumę charakterystyk tworu, które pozwalają na spełnianie konkretnych potrzeb. Oznacza to, że jakość jest względna, zależy od potrzeb. Tymi potrzebami są zazwyczaj wymagania użytkownika szukającego informacji. Idąc tym tokiem rozumowania można stwierdzić, że jakość odwzorowuje zadowolenie klienta, użyteczność informacji, zgodność z powszechnymi wymogami.

3.2.2. Ocena jakości

Ocena informacji jest zagadnieniem niezwykle trudnym ze względu na charakter tych pojęć. Zarówno informacja jak i jej jakość to pojęcia abstrakcyjne, o wielu definicjach, a ponadto zmienne w czasie i zależne od klienta. Powoduje to, że nie istnieje jedno kryterium, według którego można byłoby dokonywać oceny.

Problemem selekcji i oceny informacji rozważają nie tylko inżynierowie zajmujący się informacją, ale również współcześni bibliotekarze. W wyniku ich badań określono zestaw cech charakteryzujących dobrą informację. Najczęściej wymieniany są cechy opisane w [2] lub [3]:

- relewantność - stopień, w jakim informacja odpowiada na zapotrzebowanie użytkownika, pozostaje w związku z tematem;
- dokładność - precyzja, z jaką informacja obejmuje temat;
- aktualność - cecha, pozwalająca ocenić, czy wiadomość nie jest przestarzała, czy podąża za zmianami zachodzącymi w czasie;

- kompletność - właściwość, świadcząca o tym, że informacja jest dostatecznie obszerna by móc wyciągnąć z niej wiedzę potrzebną użytkownikowi;
- spójność - charakterystyka, która mówi, że poszczególne części informacji współgrają ze sobą, są powiązane jedną formą, są zgodne z celem jaki został nadany informacji;
- odpowiednia forma - właściwość, która pozwala ocenić, czy forma jest adekwatna do informacji, czy sprzyja jej odbiorze;
- dostępność - atrybut, który wiąże się z czasem dostępu do informacji, odpłatnością itp.;
- przystawalność - cecha, świadcząca o tym, że informacja nie jest sprzeczna z innymi informacjami na ten sam temat, że wiernie oddaje rzeczywistość;
- wiarygodność - wskaźnik, określający fakt, że informacja potwierdza rzeczywiste dane, że zawiera elementy wskazujące na jej prawdziwość, rzetelność.

Na bazie tych cech można sformułować różnorodne kryteria i metody oceny. Przykładowe kryteria oceny przedstawiono w [1]. Warto zwrócić uwagę, że jedną z najtrudniejszych do oceny cech jest relewantność, szeroko opisana w [4], zwłaszcza w przypadkach, gdy użytkownik nie jest pewien czego szuka.

3.3. Metody oceny informacji w sieci

Informacja w Internecie najczęściej zamieszczana jest na stronach portali lub jest wynikiem wynikiem działania serwisu. Stosując kryteria oceny cech wymienionych w poprzednim podrozdziale można podjąć próbę oszacowania potencjalnej użyteczności informacji. Nie jest to jednak jedyny sposób. W [3] metody oceny podzielono na trzy typy:

- techniczne (automatyczne),
- statystyczne,
- polegające na ekspertach.

Metody techniczne polegają na stosowaniu narzędzi programowych analizujących kody źródłowe stron internetowych, sprawdzających poprawność dokumentów, ich zawartość itp. Do oceny jakości wykorzystuje się również sztuczną inteligencję i algorytmy uczące się.

Metody statystyczne wykorzystują pewne dane liczbowe (jak ruch użytkowników, ilość wejść na stronę, liczbę linków). Zazwyczaj są wspomagane przez elementy techniczne (w postaci narzędzi programowych).

Ostatnia typ metod oceny jest najefektywniejsza z punktu użytkownika. W metodach tego typu jakość informacji jest wynikiem analiz przeprowadzonych przez wielu ekspertów. W wielu współczesnych projektach związanych z oceną informacji stosuje się metody wszystkich trzech typów oraz metody, których typy nie zmieściły się w wymienionym podziale.

3.3.1. Przykłady rozwiązań wspomagających ocenę

Jakość informacji bywa pojęciem subiektywnym, zależnym od samego klienta. W celu uwzględnienia tego faktu w przeprowadzanej ocenie, obecne technologie zbierają informacje o użytkowniku. Robią to poprzez pliki ciasteczek (ang. *cookies*) oraz poprzez konto użytkownika, tak jak ma to miejsce w Google. Stosując takie rozwiązania narzędzia oceniające informacje w pewien sposób tworzą model użytkownika, modyfikując tym samym swoje funkcje oceny. Prowadzi to do ukrywania pewnych informacji i eksponowania innych. Takie zjawisko nazywane jest czasem „bąblem informacyjny" i jest szerzej opisane w [5].

Istnieje też szereg innych podejść do oceny informacji znajdowanej w Internecie [6]. Należy do nich wyszukiwanie periodyczne, czyli takie, które wykonywane jest co jakiś czas. Można również wykorzystać wyszukiwanie względem czasu pojawienia się informacji, np. w ciągu ostatniej godziny.

W niektórych rozwiązaniach poprawa wyników wyszukiwania jest osiągana poprzez umożliwienie użytkownikowi doprecyzowania, jakiej informacji potrzebuje, z wykorzystaniem zdefiniowanych wcześniej kategorii bądź kategorii generowanych na bieżąco. Mowa tutaj o podpowiedziach, które pojawiają się podczas definiowania kryteriów wyszukiwania. Stosuje to obecnie większość wyszukiwarek internetowych (dopisywanie słów lub ich uzupełnianie odbywające się w polu formularza itp.), podobnie jak narzędzia wspomagające pisanie kodu (dopisywanie słów kluczowych, nazw procedur, listy atrybutów itp.).

Inną metodą wspomagającą ocenę informacji są tak zwane „snippety", czyli urywki lub małe fragmenty informacji, wyświetlane obok linku do informacji. Ich celem jest pozwolenie użytkownikowi na szybką ocenę przydatności informacji. Szczegóły działania takich metod nie są publikowane, jednakże jest to metoda powszechnie stosowana i wielce doceniana przez użytkowników. Firma Google posiada wiele patentów (np. [7]) dotyczących tego zagadnienia. Warto przy tym zauważyć, że wspomniane patenty są częścią systemu ochrony własności intelektualnej, a przez to pozwalają na biznesowe potraktowanie informacji oraz metod jej oceny. Pozwalają one popatrzeć na informację oraz metody jej oceny jak na komercyjny produkt.

3.4. Wyszukiwanie informacji

Samo wyszukiwanie danych w sieci jest trudne, głównie z powodów ulotności i ciągłych zmian struktury danych, braku jednolitego standardu dokumentów hipertekstowych oraz olbrzymiego natłoku informacji nieważnych. Wyszukiwanie informacji jest rozumiane jako obszar badawczy oraz proces odnajdywania danych w zbiorze. Zgodnie z normą PN-ISO 1087-2 2001 definiuje się je jako całość czynności, procedur i metod prowadzących do uzyskania informacji z zapamiętanego zbioru danych. Wg [8] składa się ono z:

- formułowania zapytań,
- przeszukiwania,
- nawigowania.

3.4.1. Formułowanie zapytań

Wyróżnia się trzy podstawowe typy zapytań:

- proste - składające się z jednego słowa kluczowego,
- złożone - składające się z co najmniej dwóch słów kluczowych połączonych operatorami logicznymi,
- semantyczne - mające charakter zapytań w języku naturalnym.

Obecnie większość wyszukiwarek potrafi skutecznie przetwarzać dwa z trzech wymienionych typów, choć trwają też intensywne prace nad wyszukiwarkami semantycznymi. Coraz więcej usług wyszukiwania posiada moduły umożliwiające konwersje zapytań semantycznych na zestawy słów kluczowych, choć skuteczność takich działań jest na razie niewielka.

3.4.2. Modele służące do wyszukiwania

Klasyczny model wyszukiwania zakłada przypisanie każdemu zasobowi (stronie WWW, dokumentowi itp.) odpowiedniego zestawu cech, najczęściej w postaci zbioru słów kluczowych. Zadanie samego algorytmu wyszukiwania upraszcza się wtedy do znalezienia najlepszych dopasowań pomiędzy słowami z zapytania, a charakterystykami dostępnymi w przeszukiwanej przestrzeni (mierzony jest kosinus kąta pomiędzy dwoma wektorami cech) [9]. Model ten posiada bardzo istotną wadę - zapytanie, mające kluczowe znaczenie dla uzyskanych wyników, jest formułowane przez użytkownika i dlatego nie zawsze wystarczająco dobrze reprezentuje jego potrzeby. Niemniej schemat ten znajduje się ciągle w użyciu.

Kolejnym fundamentalnym modelem jest model opracowany przez Dawisa Elissa, przedstawiony w [8]. W jednej z jego wersji wyróżniono osiem etapów:

- badania - odzyskiwania ogólnych informacji o zbiorze, na potrzeby dalszych etapów,
- łączenia - tworzenia sieci powiązań miedzy elementami,
- monitorowania - wyodrębniania w stworzonej sieci grup i węzłów kluczowych,
- przeglądania - właściwego wyszukiwania danych,
- rozróżniania - przypisywaniu grupom i węzłom fraz,
- filtrowania - ponownego przeszukiwania w poszukiwaniu grup o odpowiedniej frazie,
- weryfikacji - weryfikacji zwróconych wyników z zapytaniem,
- kończenia- sortowania po trafności.

Taki schemat pozwala uzyskiwać bardzo dobre rezultaty, jednak jego słabą stroną jest złożoność, a co za tym idzie, wydłużony czas działania. Znacznie szybszą metodą jest metoda kolejnych przybliżeń, która zakłada coraz głębsze przeszukiwanie zbioru wraz z każdą iteracją. Atutem tego podejścia jest możliwość przerwania procesu w dowolnym momencie i otrzymania przybliżonego wyniku wyszukiwania.

3.4.3. Nawigowanie

Ostatni etap wyszukiwania informacji to nawigowanie po zwróconym zbiorze danych. Proces ten obejmuje całość narzędzi i procedur używanych przez wyszukiwarkę do ułatwienia i śledzenia drogi, jaką przebywa użytkownik w celu uzyskania informacji. Podstawowym mechanizmem służącym użytkownikowi do nawigacji jest historia przeglądania, w której są logowane wszystkie odwiedzone witryny. Narzędzie jest proste, ale jednocześnie skuteczne. W przypadku zagłębienia się w zbiorze nieważnych danych, pozwala na powrót do ostatniego, interesującego węzła. Innym narzędziem, dostępnym w niektórych wyszukiwarkach, jest mechanizm wbudowanych referencji pomiędzy elementami grupy. Mechanizm działania jest następujący - dla danego wyniku są typowane odnośniki bliskoznaczne, tzn. o podobnym zestawie opisujących je słów kluczowych lub o podobnych stosunkach wystąpień słów kluczowych w zasobie. Ostatnią metodą są mechanizmy pozyskiwania informacji zwrotnej od użytkownika o przydatności danego zasobu. Technika ta ma szczególne znaczenie w systemach o ograniczonym zaufaniu, gdzie opisy zasobów są tworzone przez samych użytkowników. Ocena relatywnej przydatności zasobu pozwala na dyskryminację zasobów opisanych błędnie i zepchnięcie ich na końcowe pozycje prezentowanego wyniku wyszukiwania. W praktyce założenia są spełnione dopiero po uzyskaniu odpowiednio dużej ilości głosów/opinii. W przeciwnym wypadku oceny subiektywne, które również mogą być błędne, potrafią spowodować błędne rozpoznanie zasobów istotnych dla zapytania.

3.5. Ocena jakości wyszukiwania informacji

Ocena jakości wyszukania to proces, w którym stwierdza się, jak dobra jest jakość informacji otrzymanej w wyniku wyszukiwania, oraz jakość samego procesu wyszukiwania. Pod uwagę bierze się najczęściej relewantność informacji, liczbę wyników oraz czas wyszukania. Istnieje kilka sposobów dokonywania oceny. Jednym z nich jest ocena przez grupę wykwalifikowanych ekspertów. Zaletą tej metody jest wysoka wiarygodność oceny. Innym sposobem jest wykorzystanie wzorców. Szerzej procesy oceny jakości wyszukiwanej informacji omówiono w [10].

3.5.1. Problem identyfikacji metody oceny

Gdyby można było określić metodę, którą wykorzystuje silnik do oceny informacji, można byłoby na tej podstawie orientacyjnie ocenić jakość wyszukiwania. Nie jest to jednak łatwe, ponieważ algorytmy zaszyte w wyszukiwarkach są pilnie strzeżone przez swych twórców. Jest to podyktowane dużą konkurencją i wartością samych algorytmów. Sytuacja ta sprawia, że identyfikacja metody oceny informacji wykorzystana w silniku wyszukiwarki staje się trudna, jeśli nie niemożliwa. Można jednak potraktować taką wyszukiwarkę jako „czarną skrzynkę" i spróbować przybliżyć zasady jej funkcjonowania.

3.5.2. Testowanie wzorcowe (benchmark)

W poprzednim rozdziale nakreślono problemy z dostępem do wiedzy na temat algorytmów zajmujących się wyszukiwaniem informacji, co pociąga za sobą istotne problemy z budowa kryteriów jakości. W związku z tym w eksperymencie posłużono się najprostszą miarą jakości, jaką jest ilość wystąpień słowa kluczowego. Inną metodą oceny jakości grupy jest stosowanie bota typu WebCrawler. Bot ma na celu w dostępnej liście zasobów znaleźć hyperlinki i rozpocząć metodyczne ich odwiedzanie w celu odzyskania struktury powiązań pomiędzy zasobami, listy słów kluczowych oraz ilości hyperlinków w różnych zasobach prowadzących do tych samych stron. Dzięki temu WebCrawler zwraca bardzo kompleksową miarę jakości zasobu internetowego, umożliwiającą jego względną ocenę istotności. WebCrawer wymaga określenia polityk selekcji odnośników, rewizyt na stronach i rozpoznawania wzorca.

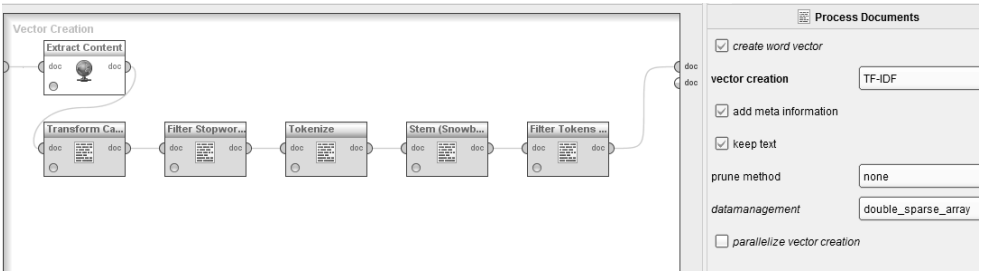
3.6. Eksperyment

W poprzednim podrozdziale został nakreślony problem identyfikacji metody wykorzystanej w danym silniku wyszukiwania. Zaprezentowany dalej eksperyment polega na uzyskaniu z wyszukiwarki zestawu wyników i znalezieniu korelacji pomiędzy pozycją zasobu na liście, a miarą wystąpień słów kluczowych z zapytania w zasobie. Do wykonania eksperymentu wybrano środowisko RapidMiner, ponieważ jest ono wolno dostępnym oprogramowaniem, co umożliwi powtórzenie eksperymentu przez dowolną osobę.

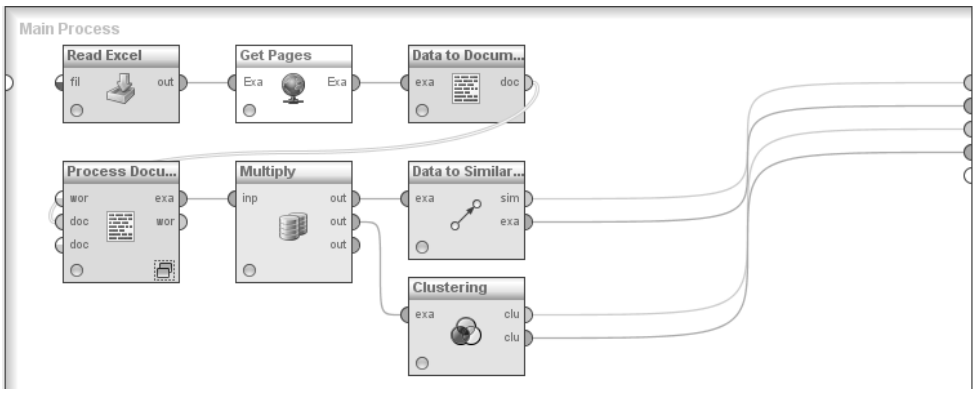
3.6.1. Opis eksperymentu

Eksperyment miał na celu określenie, czy da się zauważyć wpływ miary częstości występowania słowa kluczowego w treści wyszukanej informacji na uszeregowanie wyników wyszukiwania. Danymi wejściowymi do eksperymentu był zbiór dokumentów znalezionych przez silnik wyszukiwania dla słowa kluczowego. Następnie na podstawie treści dokumentów stworzono listę słów występujących w zbiorze dokumentów. Dla każdego z tych słów wyliczono miarę częstości występowania w każdym z dokumentów. Miary dla słów wyliczono za pomocą algorytmu `TF-IDF`. Następnie wyniki wyszukania uszeregowano według wyliczonej miary. Ostatecznie kolejność wyników otrzymanych z wyszukiwarki i z eksperymentu porównano. Na rys. 3.1 i 3.2 przedstawiono modele użyte na potrzeby eksperymentu, a na rys. 3.3 otrzymany wynik.

Kolumna `Id` (rys. 3.3) zawiera liczby mówiące o kolejności wyniku wg silnika wyszukiwania. Posortowano ją według kolejności, na jaką wskazywałaby miara `TF-IDF`. Jak można zauważyć, wykorzystana metoda dała dość dobry wynik. Kolejność zasobów jest tylko w niewielkim stopniu przemieszana, co może wskazywać na duży wpływ wybranej miary na pozycję zasobu. Aby potwierdzić tę tezę wykonano badanie na większym zbiorze danych. Zbiór danych pobrano ze strony projektu [11], który zajmuje się uczeniem algorytmów rankingujących. Zbiór składał się z ogromnego zestawu danych o rozmiarze ponad 6Gb. Dane



Rys. 3.1: Wersja podstawowa klasyfikatora.



Rys. 3.2: Klasyfikacja w oparciu o klasyfikator Bayesowski.

ExampleSet (14 examples, 17 special attributes, 2313 regular attributes)

Row No.	text	URL	Title	id	acceleromet
1	acceleromet	http://en.wikipedia.org/wiki/Accelerometer	Acceleromet	1	0.234
2	beginn guid	http://www.dimensionengineering.com/info/accelerometers	A beginner's	2	0.223
5	acceleromet	https://www.sparkfun.com/categories/80	Acceleromet	5	0.183
4	acceleromet	http://www.omega.com/prodinfo/accelerometers.html	Acceleromet	4	0.165
3	howstuffworl	http://www.howstuffworks.com/iphone-accelerometer.htm	HowStuffWo	3	0.112
10	tripl accelerc	https://www.sparkfun.com/products/10345	Triple Axis A	10	0.107
6	tripl accelerc	https://www.sparkfun.com/products/9269	Triple Axis A	6	0.101
7	acceleromet	http://www.gsmarena.com/glossary.php3?term=accelerome	Acceleromet	7	0.024
8	acceleromet	http://shop.lego.com/en-US/Accelerometer-Sensor-for-MIND	Acceleromet	8	0.021
9	what acceler	http://www.webopedia.com/TERM/A/accelerometer.html	What is acce	9	0.017
11	meter wikipe	http://en.wikipedia.org/wiki/The_Meters	The Meters -	11	0
12	meter conve	http://www.metric-conversions.org/length/meters-conversion	Meters conve	12	0
13	jame hardi e	http://www.accel.com.au/	James Hard	13	0
14	pleas choos	http://www.elero.com/	elero interna	14	0

Rys. 3.3: Wynik eksperymentu.

zawierały wiele różnych miar wyliczonych dla różnych zasobów z różnymi zapytaniami oraz jeden z czterech poziomów relewantności tego zasobu. Aby potwierdzić wcześniej postawione przypuszczenie spróbowano nauczyć klasyfikator Bayesowski rozróżniania wartości relewantności zasobu tylko na podstawie

miary TF-IDF. Jakość klasyfikacji sprawdzono poprzez walidację krzyżową. Wyniki przedstawiono w tab. 3.1 .

Na podstawie badań stwierdzono, że wykorzystana miara pozwala oddzielić zasoby całkowicie niezwiązane z zapytaniem od tych z pozostałych trzech poziomów ze skutecznością 56,67%. Nie jest to tak dobry wynik jaki można było się spodziewać na podstawie poprzedniego eksperymentu. Kolejne próby przeprowadzono na danych przekształconych tak, by odróżniać tylko dwie klasy: związane z tematem i te niezwiązane. W takim przypadku nastąpiła niewielka poprawa, co widać w tab. 3.2 i 3.3. Tak przekształcony problem pozwolił klasyfikatorowi na uzyskanie 68% i 64% precyzji w rozpoznawaniu klas. Nie zmienia to jednak faktu, że nie jest to wynik zadowalający.

Tab. 3.1: Wyniki badań zestawu danych.

class	true 0.0	true 1.0	true 2.0	true 3.0	true 3.0	class precision
pred 2.0	1	2	0	0	0	0.00%
pred 0.0	1673	809	402	41	17	56.87%
pred 1.0	22	18	14	0	0	33.33%
pred 3.0	0	0	0	0	0	0.00%
pred 4.0	0	0	0	0	0	0.00%
class recall	0.00%	98.64%	2.17%	0%	0%	

Tab. 3.2: Wyniki badań zestawu danych.

class	true 1.0	true 0.0	class precision
pred 1.0	491	222	68.86%
pred 0.0	813	1474	64.45%
class recall	37.65%	86.91%	

Tab. 3.3: Wyniki badań zestawu danych.

class	true 2.0	true 0.0	true 1.0	class precision
pred 2.0	194	165	178	36.13%
pred 0.0	185	1318	431	68.15%
pred 1.0	96	213	220	41.59%
class recall	40.84%	77.71%	26.54%	

3.6.2. Podsumowanie eksperymentu

Przedstawiony eksperyment jest nie do końca udaną próbą zbadania wpływu częstości występowania słowa kluczowego w zasobie na ocenę jego jakości. W eksperymencie pokazano, jak za pomocą algorytmu TF-IDF zbadać złożoność

metody oceniania wszytej w silnik wyszukiwania. Wyniki badań wskazały, że nie jest to wystarczająco dobry sposób na uszeregowanie zasobów według jakości zawieranej informacji. Można się nim natomiast posłużyć do oddzielenia zasobów niezwiązanych z tematem.

Literatura

[1] B. Bednarek-Michalska. *The evaluation of the quality of web-based library information services*. Biblioteka Uniwersytecka w Toruniu, 2001.

[2] B. Richmond. Ten c's for evaluating internet resources. *University of Wisconsin-Eau Claire*, 1999.

[3] B. Bednarek-Michalska. *Informacja w Internecie. Podręcznik studenta*. Biblioteka Główna Uniwersytetu Mikołaja Kopernika, 2007.

[4] E. Artowicz. *Reprezentacja wiedzy w systemie informacyjno-wyszukiwawczym : zagadnienia relewancji*. Wydaw. SBP, 1997.

[5] E. Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. Penguin Press New York, May 2011.

[6] D. Abilock. Choose the best search engine for your information needs. *The Nueva School*, 1999.

[7] A. Acharya. Generation of document snippets based on queries and search results. *US patent nr. US 8145617 B1*, 2012.

[8] T. Don. *Augmenting Information Seeking on the World Wide Web Using Collaborative Filtering Techniques*. `https://www.ischool.utexas.edu/~donturn/research/augmentis.html`.

[9] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[10] B. Grala-Kociak. *Jakość naukowa elektronicznych źródeł informacji*. Uniwersytet Medyczny w Łodzi, 2010.

[11] T. Qin, T.-Y. Liu, W. Ding, J. Xu, H. Li. Learn to rank. `http://research.microsoft.com/en-us/projects/mslr/`.

ROZDZIAŁ 4

BUDOWA SILNIKÓW DOSTARCZAJĄCYCH REKOMENDACJI

D. Moskal, M. Sawicki

W otaczającym nas świecie coraz częściej możemy natknąć się na systemy rekomendacji. Za każdym razem, kiedy używamy wyszukiwarki internetowej, korzystamy z serwisów społecznościowych, przeszukujemy oferty sklepów internetowych czy po prostu szukamy filmu do obejrzenia na wieczór, uruchamiamy algorytmy generujące oprócz odpowiedzi standardowych również zestawy odpowiedzi rekomendowanych. Zanim skorzystamy z takich rekomendacji warto przyjrzeć się bliżej zasadom ich tworzenia oraz funkcjonowania.

4.1. Wstęp

Celem tworzenia rekomendacji jest przedstawienie użytkownikowi pewnych propozycji (często w niejawny sposób), które w jakiś sposób wiążą się z obszarem jego zainteresowań. Silnik rekomendacji to system filtracji informacji, próbujący przewidzieć ocenę użytkownika dotyczącą obiektu, z którym ten wcześniej nie miał do czynienia. Polecane mogą być zarówno przedmioty, usługi jak i elementy socjalne, np. ludzie, grupy społeczne.

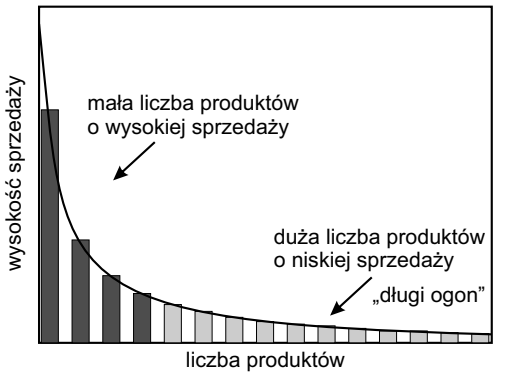
Wyróżnić można trzy rodzaje silników rekomendacji:

- bazujące na cechach ocenianych produktów (tzw. *content–based filtering*),
- bazujące na środowisku społecznym (tzw. *collaborative filtering*),
- łączące obie metody – hybrydowe.

4.1.1. „Długi ogon" – czyli dlaczego systemy rekomendacji są potrzebne

Uzasadnieniem, które przemawia za stosowaniem systemów rekomendacji jest zjawisko tzw. „długiego ogona" (ang. *long tail*). Pojęcie to, spopularyzowane przez Chrisa Andersona i opisane w książce [1], dotyczy pewnego zjawiska ekonomicznego, którego szczególnym przypadkiem jest sprzedaż internetowa.

Podczas normalnej sprzedaży liczba produktów oferowanych w danym sklepie jest ograniczona fizycznie rozmiarem przestrzeni magazynowej. W przypadku serwisów obsługujących sprzedaż internetową problem fizycznych ograniczeń nie istnieje. Wirtualna oferta może zawierać nieskończoną wręcz listę sprzedawanych produktów. Możliwe staje się więc sprzedawanie produktów niszowych. W pracy [2] udowodniono, że w internecie sprzedaż dużej liczby produktów niszowych generuje większe zyski niż sprzedaż bestsellerów. To właśnie jest „długi ogon". Na rys. 4.1 pokazano przykładowy wykres sprzedaży produktów w Internecie. Na osi odciętych pojawia się wysokość sprzedaży (odzwierciedlająca popularność produktów), zaś na osi rzędnych - liczba sprzedawanych produktów (obrazująca liczbę kategorii produktów, których sprzedaż osiąga taką samą wysokość).



Rys. 4.1: Sprzedaż produktów w Internecie obrazująca zjawisko „długiego ogona".

W książce [1] pokazano, jak bardzo silniki rekomendacji mogą wpływać na rynek rekomendując produkty z „długiego ogona". W 1988 wydano książkę poświęconą wspinaczce górskiej – „Touching the Void". Nie udało się jednak odnieść sukcesu w jej sprzedaży. Wiele lat później rozgłos zyskała inna pozycja o tej samej tematyce – „Into Thin Air". System rekomendacyjny serwisu Amazon zauważył, że kilku użytkowników, którzy kupili drugą pozycję, nabyli także starszą książkę. Rekomendowanie „Touching the Void" wszystkim zainteresowanym kupnem „Into Thin Air" spowodowały, że „Touching the Void" bardzo zyskała na popularności.

4.2. Przegląd algorytmów rekomendujących

4.2.1. Formalne postawienie problemu

Problem rekomendacji można sformułować w sposób formalny podobnie jak w pracy [3]. Niech C będzie zbiorem użytkowników, a S – zbiorem wszystkich elementów, które mogą być obiektem rekomendacji (np. książek, filmów czy restauracji). Oba zbiory mogą zawierać od setek do nawet milionów elementów.

Tab. 4.1: Przykładowe oceny produktów w systemie rekomendującym filmy.

Użytkownik	„Gwiezdne Wojny”	„Władca Pierścieni”	„Red”	„Avengers”	„Matrix”
Tomek	10	9			7
Adam	8		7	9	7
Kasia			5		10

Funkcję użyteczności elementu s dla użytkownika c oznaczmy przez $u : C \times S \rightarrow R$ (gdzie R to zbiór dodatnich liczb z wybranego zakresu). Zadaniem silnika rekomendacji jest więc znalezienie dla każdego użytkownika c takiego przedmiotu s', który maksymalizuje funkcję użyteczności.

Wartość użyteczności obiektu może być nadawana przez użytkownika – odzwierciedla wtedy bezpośrednio opinię użytkownika na temat przedmiotu, albo może być wyliczana przez system – odzwierciedla wtedy np. potencjalny zysk, jaki sklep osiągnie przy sprzedaży danego produktu. W tab. 4.1 pokazano przykładowy fragment bazy danych systemu rekomendacji filmów, w którym trzech użytkowników oceniało pięć filmów.

Każdy użytkownik może – ale nie musi – posiadać profil z zestawem zdefiniowany cech, takich jak: wiek, płeć, stan cywilny, dochody itp. Podobnie każdy element przestrzeni S może posiadać własną charakterystykę. I tak w systemie rekomendacji filmów może to być: rok produkcji, gatunek, reżyser, aktorzy pierwszoplanowi itp.

Podstawowym problem w systemach rekomendujących jest konieczność ekstrapolacji funkcji użyteczności u, gdyż praktycznie zawsze nie jest ona określona na całej przestrzeni $C \times S$. Dobrze widać to w tab. 4.1, gdzie użytkownicy nie ocenili wszystkich filmów. Projektując silnik rekomendacji należy określić sposób, w jaki brakujące oceny będą estymowane. W tym celu:

1. określa się heurystykę definiującą funkcję użyteczności, a następnie sprawdza się eksperymentalnie jej poprawność,
2. estymuje się nieznane wartości funkcji użyteczności poprzez optymalizację pewnego kryterium (np. średniokwadratowe).

W większości wypadków nie jest konieczne przewidywanie oceny użytkownika c dla każdego z elementów s. Wystarczy, że określone zostaną te, dla których przewidywana wartość będzie wysoka, tak, aby dało się wygenerować trafną rekomendację.

4.2.2. Silniki typu *content–based*

W systemach bazujących na cechach produktów ([4, 5]), rekomendacje udzielane są na podstawie oceny podobieństwa elementów zbioru S. Podobieństwo to jest określane z wykorzystaniem cech opisujących element. Chcąc estymować wartość funkcji użyteczności $u(c, s)$, jaką użytkownik c przypisałby dotąd nieocenianemu elementowi s, należy najpierw odnaleźć te elementy s_i, które są podobne do s. Potem estymacja dokonywana jest w oparciu o wartości $u(c, s_i)$.

Można wyróżnić dwie podstawowe elementy takiego systemu:

- element odpowiedzialny za sposób reprezentacji cech rekomendowanych elementów,
- element odpowiedzialny za przedstawienie profilu użytkownika.

Reprezentacja elementów

Pierwszym ze sposobów przypisania cech elementom systemu jest ich manualne scharakteryzowanie. W przypadku rekomendacji filmów należy więc wprowadzić do bazy wybrane cechy filmu, w dużej mierze uzyskiwane z jego opisu. To mogą być: gatunek, reżyser, aktorzy pierwszoplanowi, zdobyte nagrody, tematyka i inne.

Silniki typu *content–based* najczęściej jednak wykorzystywane są do rekomendacji tekstów: artykułów dostępnych on–line oraz całych stron internetowych. W tym przypadku konieczne staje się automatyczne określenie cech opisów wyrażonych w języku naturalnym. Najczęściej odbywa się ono poprzez przypisanie wag poszczególnym słowom i wybranie pewnej liczby słów najbardziej znaczących. Zwykle wymaga to oddzielenia rdzenia słów od końcówek, tak, by razem liczyć wyrazy jednej rodziny (w języku polskim należy też uniezależnić się od zmiany form wyrazu w zależności od deklinacji, np. komputer, komputerowi, komputeryzacja, komputerowy, komputerowo).

Określenie wagi bądź znaczenia może odbywać się na różne sposoby. Jednym z najbardziej znanych jest wykorzystanie miary TF–IDF (ang. *term frequency/inverse document frequency*). Określając wagę słowa k_i w dokumencie d_j najpierw należy wyliczyć znormalizowaną wartość częstości $tf_{i,j}$ ze wzoru:

$$tf_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}},$$

gdzie $tf_{i,j}$ to liczba wystąpień wyrażenia k_i w dokumencie d_j, a maksimum wyliczane jest po liczbie wystąpień wszystkich słów k_z w dokumencie.

Kolejnym krokiem jest wyznaczenie odwrotnej częstości idf_i słowa k_i

$$idf_i = \log \frac{N}{n_i},$$

gdzie N to liczba wszystkich dokumentów, zaś n_i to liczba dokumentów zawierających słowo k_i.

Ostatecznie wagę słowa $w_{i,j}$ wylicza się jako iloczyn wektorowy dwóch powyższych wartości

$$w_{i,j} = tf_{i,j} \times idf_i,$$

Opis całego dokumentu d_j stanowi więc wektor wartości $\vec{w_{dj}}$.

Profil użytkownika

Na profil użytkownika składają się dwa typy informacji: preferencje użytkownika oraz historia jego działań. W trakcie tworzenia profilu systemy rekomendacji często udostępniają interfejs umożliwiający użytkownikowi określenie jego preferencji. Zazwyczaj odbywa się to przez wyświetlenie serii przycisków wyboru

(ang. *check box*) oraz list wyboru. Sposób ten ma jednak sporo wad, wśród których warto wymienić niechęć użytkowników do uzupełniania tego typu danych oraz o małą precyzję generowanych informacji. Z drugiej strony wprowadzenie preferencji podczas tworzenia profilu pozwala na dokonywanie spersonalizowanych rekomendacji dla nowych użytkowników.

W przypadku użytkowników posiadających pewną historię działań w systemie, model ich preferencji jest tworzony z wykorzystaniem różnych metod klasyfikacji nadzorowanej. Jako dane treningowe dla klasyfikatorów wykorzystywane są zarówno bezpośrednie oceny użyteczności elementów podane przez użytkownika, jak i informacje pośrednie, takie jak zakup produktu lub zakup i zwrot produktu. Do najczęściej stosowanych metod tworzenia modelu użytkownika należą:

- drzewa decyzyjne,
- n najbliższych sąsiadów,
- klasyfikator liniowy,
- sieci neuronowe,
- wyszukiwanie relewantne, w szczególności algorytm Roccio'a,
- metody probabilistyczne, w tym naiwny klasyfikator Bayes'a.

W przypadku systemów rekomendacji stron lub artykułów, profil użytkownika c ma postać wektora $\vec{w}_c$, na który składają się użyteczności $w_{i,c}$ poszczególnych słów k_i. Ostatecznie, wartość użyteczności dokumentu d_j dla użytkownika wyznacza się wykorzystując wybraną heurystykę, np.:

$$u(d_j, c) = \cos(\vec{w}_c, \vec{w_{dj}}) = \frac{\sum_{i=1}^{K} w_{i,c} w_{i,j}}{\sqrt{\sum_{i=1}^{K} w_{i,c}^2 \sum_{i=1}^{K} w_{i,j}^2}}, \tag{4.1}$$

gdzie K to całkowita liczba słów kluczowych w systemie.

Podsumowanie

Systemy bazujące na cechach produktów są zwykle mocno ograniczone przez dostępność tych cech. O ile w przypadku tekstów automatyczne wyszukanie cech nie stanowi zazwyczaj problemu, o tyle w pozostałych przypadkach (rekomendacje filmów, muzyki czy przedmiotów) często konieczne jest ręczne uzupełnianie tych danych. Ponadto, jeżeli dwa elementy reprezentowane są przez ten sam wektor cech, system nie jest w stanie ich odróżnić – tak samo często będzie rekomendował np. źle i dobrze napisany artykuł.

Innym problemem jest duża specjalizacja takiego systemu – użytkownik otrzymuje rekomendacje elementów podobnych do tych, które zostały odnalezione w jego historii. Tym samym, jeżeli jest zainteresowany informatyką i czyta dużo artykułów z nią związanych, system nie będzie mu polecał tekstów na żaden inny temat. Z tego powodu często do silników rekomendacji wprowadza się element losowy, np. wykorzystując algorytmy genetyczne.

4.2.3. Silniki typu *collaborative*

W systemach bazujących na środowiskach społecznościowych [5, 6, 7], rekomendacje udzielane są na podstawie oceny, jaką inni użytkownicy dali danemu elementowi zbioru S. Chcąc estymować wartość funkcji użyteczności $u(c,s)$, jaką użytkownik c przypisałby dotąd nieocenianemu elementowi s, należy najpierw odnaleźć oceny $u(c_j,s)$ podane przez użytkowników c_j, którzy mają podobne zainteresowania do użytkownika c.

Wyróżnić można dwa podstawowe typy systemów [6]:

- systemy bazujące na pamięci (ang. *memory–based*) – oparte na heurystykach obejmujących wszystkie oceny wszystkich użytkowników systemu,
- systemy bazujące na modelu (ang. *model–based*) – tworzące model preferencji użytkowników.

Algorytmy typu *memory–based*

W tym podejściu wartość funkcji użyteczności $u(c,s)$ wyliczana jest bezpośrednio z ocen innych użytkowników. Agregacja opiera się na podstawie ocen użytkowników, którzy ocenili już element s. Zwykle branych jest pod uwagę N użytkowników, o preferencjach najbardziej podobnych do preferencji użytkownika c. Przykładami takich funkcji mogą być:

$$u(c,s) = \frac{1}{N}\sum_{i=1}^{N} u(c_i,s), \tag{4.2}$$

$$u(c,s) = \sum_{i=1}^{N} w(c_i,c)\,u(c_i,s), \tag{4.3}$$

$$u(c,s) = \bar{u}_c + k\sum_{i=1}^{N} w(c_i,c)\left(u(c_i,s) - \bar{u}_{c_i}\right), \tag{4.4}$$

gdzie
$\bar{u}_{c_i}$ – średnia ocen udzielonych przez użytkownika c_i,
$w(c_i,c)$ – miara podobieństwa pomiędzy użytkownikami c_i i c,
k – współczynnik skalujący (normalizujący).

W pierwszym przypadku (4.2) poszukiwaną wartością jest prosta średnia arytmetyczna. Zwykle jednak preferuje się średnią ważoną. Wagi $w(c_i,c)$ stanowią miarę podobieństwa pomiędzy użytkownikami i mogę być wyrażone jako odległość czy korelacja pomiędzy c_i i c. Należy zwrócić uwagę, że miary te są jedynie heurystykami wyliczanymi na podstawie ocen wprowadzonych przez użytkowników (zazwyczaj opierają się na ocenach elementów, które użytkownicy ocenili). Najczęściej stosowane miary to korelacja i kosinus kąta.

Korelacja
Do wyznaczenia podobieństwa pomiędzy użytkownikiem c_i a c wykorzystuje się współczynnik korelacji Pearsona:

$$w(c_i, c) = \frac{\sum_{j=1}^{m} \left(u(c_i, s_j) - \bar{u}_{c_i}\right)\left(u(c, s_j) - \bar{u}_c\right)}{\sqrt{\sum_{j=1}^{m} \left(u(c_i, s_j) - \bar{u}_{c_i}\right)^2 \sum_{j=1}^{m} \left(u(c, s_j) - \bar{u}_c\right)^2}}, \tag{4.5}$$

przy czym sumowanie odbywa się po elementach, które ocenili obaj użytkownicy.

Kosinus kąta
W tym podejściu, użytkowników traktuje się jako wektory w m–wymiarowej przestrzeni ($\vec{u}_c$ reprezentuje użytkownika c), gdzie m to liczba elementów ocenionych przez obu użytkowników. Podobieństwo określone jest przez kosinus kąta pomiędzy tymi wektorami

$$w(c_i, c) = \cos(\vec{u}_c, \vec{u}_{c_i}) = \frac{\vec{u}_c \cdot \vec{u}_{c_i}}{\|\vec{u}_c\|_2 \times \left\|\vec{u}_{c_i}\right\|_2} = \frac{\sum_{j=1}^{m} u(c_i, s_j) u(c, s_j)}{\sqrt{\sum_{k=1}^{m} u(c_i, s_j)^2} \sqrt{\sum_{k=1}^{m} u(c, s_j)^2}} \tag{4.6}$$

W tym przypadku wyrażenie w mianowniku pełni rolę normalizującą, aby waga nie była zależna od liczby m. Normalizacja ta może przybrać inną formę, jednak wtedy stracony zostanie sens kosinusa kąta.

Jednym z podstawowych problemów przy stosowaniu wzoru (4.3) jest wykorzystanie tej samej skali ocen do wszystkich użytkowników systemu. W rzeczywistości użytkownicy inaczej interpretują przyznawane przez siebie oceny, np. przy skali 1–10 użytkownik c_1 oceni każdy film, który mu się podoba, na 9, a inny przyzna 9 tylko wyjątkowym filmom. W celu wyeliminowania tej różnicy można uwzględnić średnią ocen, jaką dany użytkownik przyznał, i przewidywać oceny jako odchylenie od tej średniej – wzór (4.4).

Do podstawowych wad algorytmów typu *memory–based* należą:

- konieczność arbitralnego wyboru parametrów, np. liczby N najbliższych sąsiadów branych pod uwagę przy estymowaniu funkcji użyteczności,
- w przypadku serwisów społecznościowych trudność precyzyjnego określenie podobieństwa pomiędzy użytkownikami na podstawie np. łączących ich relacji.

Algorytmy typu *model–based*

Innym podejściem do systemów opartych na środowisku społecznym jest wykorzystanie modelu użytkownika uzyskanego w procesie uczenia maszynowego. Na jego podstawie jest określane prawdopodobieństwo P, że użytkownik c przyzna elementowi s ocenę i, $i \in \langle 0, n \rangle$

$$u(c, s) = E(c, s) = \sum_{i=0}^{n} i \times P\left(u(c, s) = i | u(c, s'), s' \in S_c\right),$$

gdzie S_c to zbiór wszystkich elementów ocenionych przez użytkownika c.

Najczęściej stosowane metody wyznaczania prawdopodobieństwa to model klastrowy (ang. *cluster models*) i sieci bayesowskie.

Model klastrowy
W modelu klastrowym użytkowników dzieli na pewne klasy Λ zawierające osoby

o podobnych preferencjach. Dodatkowo zakłada się, że w jednej klasie oceny poszczególnych elementów są niezależne od siebie. Przy takich założeniach do wyznaczenie prawdopodobieństwa można wykorzystać naiwny klasyfikator bayesowski

$$P(\Lambda_c = \Lambda, u(c, s_1,) \ldots, u(c, s_n)) = P(\Lambda_c = \Lambda) \prod_{i=1}^{n} P(u(c, s_i)|\Lambda_c = \Lambda).$$

Lewa strona to prawdopodobieństwo obserwacji użytkownika c o klasie Λ oraz zestawie ocen $u(c, s_1,) \ldots, u(c, s_n)$. Liczba klas, prawdopodobieństwo przynależności użytkownika do klasy, $P(\Lambda_c = \Lambda)$, oraz warunkowe prawdopodobieństwa ocen, $P(u(c, s_i)|\Lambda_c = \Lambda)$, uzyskiwane są z zestawu uczącego – bazy danych systemu rekomendacji.

Sieci bayesowskie
Innym podejściem jest zastosowanie sieci bayesowskich, w których elementy s stanowią węzły. Stan węzłów odpowiada ocenom, jakie może uzyskać dany element, z uwzględnieniem stanu „brak oceny". Za pomocą algorytmów uczących się, tworzona jest sieć powiązań pomiędzy elementami – każdy z nich otrzymuje zestaw rodziców, na podstawie których można najlepiej przewidzieć ocenę. Przykładowo, dla filmu „Avengers", rodzicami mogą być inne produkcje oparte o komiksy Marvela, jak „Iron Man", „Thor" czy „Hulk".

Wśród podstawowych wad algorytmów typu *model–based* można wymienić:

- konieczność dostrojenia bardzo wiele parametrów modelu,
- generalizacja, jaka pojawia się w modelu, może nie być odpowiednia dla wszystkich przypadków (np. wyszukiwanie może nie sprawdzać się dla „horrorów"),
- podobnie jak poprzednio, implementacja danych z serwisów społecznościowych wymaga dodatkowych zabiegów.

Podsumowanie

Systemy rekomendacji bazujące na środowisku społecznym są pozbawione niektórych wad charakterystycznych dla systemów bazujących na cechach produktów. Przede wszystkim, jako zależne od oceny innych użytkowników, mogą być zastosowane niezależnie od typów elementów, a więc także w stosunku do elementów, których z jakiś powodów nie da się opisać wcześniej ustalonymi cechami. Niemniej jednak, mają swoje ograniczenia.

Nowi użytkownicy
Nowi użytkownicy stanowią spory problem, bowiem zanim udzielone zostaną dobre rekomendacje, konieczne jest poznanie ich preferencji. Część serwisów wymaga więc, by użytkownik ocenił pewną liczbę elementów, zanim otrzyma rekomendacje. Inne sposoby na ominięcie tego problemu to wykorzystanie metod hybrydowych (patrz sekcja 4.2.4) oraz różnych technik opartych o popularność poszczególnych elementów, ich entropię czy personalizację profilu użytkownika.

Nowe elementy
W przypadku nowych elementów, aby system brał je pod uwagę podczas udziela-

nia rekomendacji, muszą one zostać ocenione przez dostateczną liczbę użytkowników. Tu ponownie rozwiązaniem jest zastosowanie metod hybrydowych.

Braki w danych
Macierze reprezentujące dane w systemach rekomendacji są zazwyczaj macierzami rzadkimi. Konieczne jest więc efektywne przewidywanie ocen w oparciu o niewielką liczbę danych. Jak już zostało wspomniane, jeżeli element został oceniony przez zbyt małą liczbę użytkowników, będzie rzadko polecany. Problem stanowią też użytkownicy o nietypowych preferencjach – w ich wypadku może się okazać, że w systemie jest zbyt mała liczba osób o podobnych zainteresowaniach, by udzielić skutecznej rekomendacji. Wraz z rozrastaniem się systemu to zagrożenie oczywiście maleje.

4.2.4. Silniki hybrydowe

Część systemów rekomendacji wykorzystuje silniki hybrydowe [8, 3], scalające oba podejścia opisane w poprzednich podrozdziałach. Systemy takie łączą w sobie zalety zarówno algorytmów *content–based* (CB) jak i *collaborative* (COL). Dzięki temu:

- rekomendacje opierają się na doświadczeniu innych użytkowników, a nie tylko na opisie produktu (COL),
- silnik uwzględnia elementy ocenione przez niewielką liczbę użytkowników (CB),
- można udzielać rekomendacji użytkownikom o nietypowych preferencjach wykorzystując opisy produktów (CB),
- możliwa staje się filtracja elementów (CB),
- wykorzystanie profilu elementu (patrz podrozdział 4.2.2) do określenia grup podobnych elementów umożliwia udzielanie rekomendacji w oparciu o użytkowników, którzy ocenili zbliżone, ale nie identyczne elementy,
- przez grupowanie elementów można przyspieszyć proces rekomendacji.

W nawiasach występujących w powyższym wyliczeniu podano typ silnika rekomendacji, który odpowiada za daną cechę. Jak widać, zakres osiąganych korzyści zależy od konkretnej metody połączenia silników rekomendacji różnych typów. Silniki hybrydowe można podzielić na cztery grupy (zgodnie z [3]). Omówione je krótko poniżej.

Niezależna implementacja

W najprostszym podejściu oba typy silników zaimplementowane są niezależnie i każdy z nich niezależnie tworzy listę rekomendacji. Ostateczna lista dla użytkownika może powstać poprzez agregację (np. w formie kombinacji liniowej) lub poprzez wybór jednej z list („lepszej" w sensie jakiegoś kryterium, np. wyznaczonej z większą pewnością lub z ocenami bliższymi średniej ocen danego użytkownika).

Rozszerzenie metody *content–based*

Podejście to rozszerza silnik typu *content–based* o profil użytkownika charakterystyczny dla silników typu *collaborative*. Podobieństwo użytkowników wyliczane jest na podstawie takiego profilu, a nie w oparciu o wspólnie ocenione elementy. Podejście to daje dwie znaczące korzyści:

- omija problem rzadkich macierzy opisujących preferencje użytkowników, pozwalając na znalezienie osób o podobnych preferencjach niezależnie od wielkości systemu (największe znaczenie ma to oczywiście w przypadku mniejszych serwisów rekomendacji),
- nowe elementy, ocenione przez niewielką liczbę użytkowników, mogą być rekomendowane na podstawie dopasowania do profilu użytkownika.

Rozszerzenie metody *collaborative*

W tym podejściu zazwyczaj wykorzystuje się techniki zmniejszające wymiar przeszukiwanych przestrzeni, wykorzystując społecznościowe zależności pomiędzy użytkownikami.

Silniki zunifikowane

Metody typu *content–based* i *collaborative* można połączyć, tworząc zunifikowany silnik rekomendacji. Zaproponowanych zostało wiele sposobów na dokonanie takiego połączenia, np.:

- wykorzystanie charakterystyk z obu metod do stworzenia pojedynczego klasyfikatora bazującego na regułach,
- połączenie silników z wykorzystaniem metod probabilistycznych,
- połączenie silników z wykorzystaniem wnioskowania logicznego.

4.2.5. Podsumowanie

Pomimo usilnych badań nad silnikami rekomendacji, nie zdołano zbudować rozwiązania spełniającego wszystkie stawiane wymagania, a stosowane algorytmy wymagają wielu zmian i ulepszeń. W większości przypadków omówione podstawowe typy silników rekomendacji nie są wystarczające i konieczne jest wprowadzenie zmian, czasem bardzo znacznych, zależnych od specyfiki wykorzystujących je serwisów. Innym wyzwaniem jest przetwarzanie rosnącej liczby danych, tak by użytkownik otrzymywał odpowiedź od systemu natychmiastowo, nawet jeśli do bazy danych wprowadził właśnie nowe informacje (do tej pory część serwisów aktualizuje rekomendacje co jakiś czas, np. raz dziennie). Zwiększenie efektywności rekomendacji zapewne na długo pozostanie szerokim polem do badań naukowych.

4.3. Zastosowanie

Silniki rekomendacji mają zastosowanie w bardzo wielu dziedzinach. Poniżej dokonano przeglądu kilku obszarów zastosowań oraz serwisów korzystających z takich silników.

4.3.1. Portale ratingowe

Portale ratingowe, związane najczęściej z szeroko pojętą kulturą, to jeden z obszarów obecnie nierozerwalnie złączony z silnikami rekomendacji.

Filmy

Polecaniem filmów zajmują się dwa rodzaje serwisów. Pierwsze są nastawione na dostarczenie użytkownikowi informacji i nie czerpią bezpośrednich zysków z rekomendacji (przychód zwykle uzyskują z reklam). Do tej kategorii należą serwisy `filmweb.pl` oraz `filmaster.pl`, cieszące się największą popularnością w Polsce. W obu przypadkach rekomendacje udzielane są na bazie odpowiednio zmodyfikowanych silników typu *collaborative*, zaś użytkownik, aby je otrzymać, musi ocenić pewną minimalną liczbę filmów. Drugą grupą serwisów rekomendujących filmy są wypożyczalnie, które oczywiście czerpią bezpośrednie zyski z dobrze udzielonych rekomendacji. Największym portalem tego typu, a jednocześnie największą światową wypożyczalnią filmów, jest Netflix (posiadający silnik hybrydowy, obecnie jeden z najbardziej zaawansowanych). O znaczeniu rekomendacji świadczy fakt, że pochodzi z nich aż 65% wypożyczeń!

Poza samymi filmami, rekomendacje dotyczą także wszelkich programów telewizyjnych. Ciekawym przykładem jest tu silnik *Sybil* rozwijany przez stację BBC. W przeciwieństwie do większości obecnie stosowanych algorytmów, rekomendacji dokonuje się w nim po stronie użytkownika, a nie serwera, co znacznie przyspiesza działanie.

Rekomendacje wychodzą także poza ramy serwisów internetowych – Samsung wypuścił na rynek telewizory Smart TV z funkcją S Recommendation. Na podstawie wyświetlanych programów telewizor tworzy profil użytkownika, a następnie, łącząc się z internetową bazą danych, poleca mu filmy i programy, który w wybranym czasie mogą zainteresować właściciela sprzętu.

Książki

Poza portalami niekomercyjnymi, jak np. `lubimyczytac.pl` z systemów rekomendacji korzystają księgarnie internetowe. W największej z nich, `Amazon.com`, 35% sprzedaży pochodzi właśnie z rekomendacji. Serwis wykorzystuje algorytm hybrydowy *Item-to-Item Collaborative Filtering* [9] i sugeruje użytkownikowi produkty, które zakupili inni nabywcy danej książki.

Muzyka

Również w świecie muzyki nie brakuje rekomendacji. Polecane są zarówno całe płyty, jak i konkretne utwory bądź artyści. Przykładem serwisów udostępniających muzykę są `last.fm` i Pandora Radio. `last.fm` poleca utwory w oparciu

o silnik typu *collaborative*. Korzystając z serwisu wspólnie ze znajomymi można porównać swoje gusty muzyczne za pomocą „Gust-o-metrów", pokazujących podobieństwo preferencji użytkowników. W przypadku radia Pandora wykorzystywany jest silnik typu *content–based*, w którym utwory charakteryzuje wektor około 400 atrybutów takich jak tempo i główne wokalu, główne harmoniczne, synkopa czy system tonalny. Mechanizm ten, poza dostosowywaniem muzyki do gustu słuchającego, pozwala na promowanie mniej znanych artystów.

4.3.2. Zakupy

Sklepy internetowe należą niewątpliwie do największych beneficjentów rekomendacji. Wykorzystują one bardzo różne algorytmy, od najprostszych, aż po skomplikowane silniki hybrydowe znane z największych serwisów (np. wspomniane już `Amazon.com`). W większości przypadków konieczne jest jednak, by udzielanie rekomendacji odbywało się niezależnie od profilu użytkownika. W takim przypadku, w zależności od serwisu, jego charakteru i wielkości, listy rekomendacji mogą być budowane m.in. w oparciu o:

- produkty wprowadzone manualnie (mogą to być np. towary obecnie przecenione bądź takie, które mają wysoką marżę),
- *bestsellery*,
- nowości,
- produkty, które kupowały inne osoby zainteresowane właśnie wyświetlanym towarem,
- towary cieszące się największą popularnością w danej kategorii,
- produkty często kupowane razem z właśnie wyświetlanym (czasem będą to towary o zupełnie innym charakterze, czasem mocno powiązane, jak np. akcesoria uzupełniające wybrany sprzęt),
- podobne produkty.

W serwisach zakupowych należy zwrócić uwagę, aby rekomendacje nie były zbyt oczywiste. W przypadku np. serwisów filmowych polecanie „Władcy Pierścieni" każdemu użytkownikowi, który lubi filmy fantasy, a nie widział jeszcze ekranizacji dzieła Tolkiena, ma sens. Jednak np. w delikatesach internetowych, polecanie wszystkim klientom bananów nie przyniesie dodatkowych zysków – produkt ten jest kupowany niezależnie od tego, czy pojawi się rekomendacja. Dlatego też inteligentne silniki rekomendacji omijają polecanie produktów zbyt oczywistych.

W przypadku sklepów rekomendacje wychodzą poza obszar Internetu. Wyposażone w nie mogą być np. wózki sklepowe. Wykorzystując smartfony lub karty lojalnościowe, klient loguje się do systemu, a wózek podpowiada mu np. jak najkrótszą drogą dostać się do wybranych działów, jednocześnie zbierając informacje o kupującym. Historię zakupów konkretnych osób jest zapisywana, aby przyciągać klientów spersonalizowanymi ofertami promocyjnymi, dostarczanymi w formie książeczki rabatowej lub wyświetlanymi na ekranie wózka.

4.3.3. Artykuły i strony internetowe

Najpopularniejsze wyszukiwarki internetowe posiadają dobrze zaimplementowane algorytmy rekomendacji. Przeważnie już pierwszy uzyskany w nich wynik wyszukiwania jest zgodny z oczekiwaniami użytkownika. Do generowania rekomendacji i do rankingowania wyników wykorzystuje się w nich historię przeglądanych stron, historię wpisywanych haseł oraz wiele innych dodatkowych informacji. Dobrze działające algorytmy przekładają się na popularność usług je oferujących, a to z kolei na zyski dla ich właścicieli (generowanych poprzez podsuwanie użytkownikom wyników sponsorowanych przez firmy zewnętrzne). Biznesowe podejście szczególnie widać w usługach dostarczających reklamy o treści odpowiadającej zawartości aktualnie przeglądanej strony. W przypadku wykorzystywania usługi Gmail, treści reklam dobierane są na podstawie poczty, tak aby zmaksymalizować szansę zainteresowania nimi użytkownika.

Rekomendacje pojawiają się na stronach serwisów informacyjnych, gdzie zwykle przyjmują prostą formę polecanych najnowszych artykułów oraz wiadomości wzbudzające szczególne emocje. Znacznie bardziej zaawansowanym silnikiem dysponuje Google News - wprowadzenie rekomendacji bazujących na informacjach zawartych w profilu GoogleAcconts pozwoliło zwiększyć ruch na tym portalu o 38%. Innym ciekawym przykładem jest znajdujący się w fazie wczesnych testów serwis Lumi, stworzony przez twórców `last.fm`. Zadaniem Lumi jest anonimowa i bezpieczna rejestracja odwiedzanych stron i polecanie tych popularnych innym użytkownikom. Warto tu jeszcze wspomnieć o innych serwisach, polecających

- miejsca, takie jak restauracje, kluby czy hotele,
- kursy uczelniane i dodatkowe zajęcia edukacyjne,
- towarzystwo osób, zwłaszcza na portalach randkowych,
- grupy zainteresowań, włączając w to znajomych na portalach społecznościowych jak Facebook, MySpace, LinkedIn,
- narzędzia finansowe, np. monitoring akcji giełdowych, jak w portalu `Stop-loss.pl`.

Ten krótki przegląd nie wyczerpuje oczywiście tematu związanego z opisem wszystkich możliwych zastosowań silników rekomendacji. Rekomendowane może być bowiem prawie wszystko, czego użytkownicy poszukują w Internecie.

4.4. Rekomendator

4.4.1. Serwis Filmaster

Filmaster.pl to serwis społecznościowy poświęcony tematyce filmowej. Jedną z jego najważniejszych funkcji jest rekomendacja filmów – celem twórców jest stworzenie możliwie najlepszego algorytmu rekomendującego, który można byłoby wykorzystać także w innych aplikacjach. W przeciwieństwie do większości serwisów, Filmaster.pl udostępnia swój algorytm na licencji AGPLv3. Można więc

go swobodnie wykorzystywać we własnych serwisach. Co więcej, autorzy zachęcają także do włączenia się w tworzenie kodu – silnik jest bowiem stale udoskonalany.

Algorytm wykorzystywany w serwisie jest typowym algorytmem typu *content–based*. Każdy film z bazy jest w nim opisywany przez 30 abstrakcyjnych cech (liczba cech została określona eksperymentalnie – przy mniejszej algorytm miał zbyt mało danych, większa zaś powodowała przeuczenie go). Te same cechy wchodzą w wektor opisujący użytkownika. Wzór na ostateczną ocenę filmu s dla użytkownika c podobny jest do równania (4.1):

$$u(c,s) = \sum_{i=1}^{30} w_{i,c} w_{i,s}, \tag{4.7}$$

gdzie $w_{i,c}$ – waga poszczególnych cech dla użytkownika c, $w_{i,s}$ – waga poszczególnych cech w filmie s. Wagi są tak dobierane, aby końcowa wartość nie musiała być normalizowana. Jeżeli zdarzy się, że wykroczy ona poza zakres (oceny przyznawane są w skali 0–10), przypisywana jest jej wartość skrajna.

Wagi dla poszczególnych cech wyznaczane są w procesie uczenia. Wykorzystując ocenione przez użytkownika filmy (serwis wymaga, by było ich nie mniej niż 15), przeprowadzanych jest 50 iteracji algorytmu optymalizującego. Celem jest minimalizacja błędu średniokwadratowego wyznaczonego na próbie uczącej. W kolejnych przebiegach wagi zmieniają się według prostego wzoru, i tak w k-tym kroku przybierają wartość

$$w_{i,c}^k = w_{i,c}^{k-1} + \mu \left(u(c,s) - u^{k-1}(c,s) \right) w_{i,c}, \tag{4.8}$$

$$w_{i,s}^k = w_{i,s}^{k-1} + \mu \left(u(c,s) - u^{k-1}(c,s) \right) w_{i,s}, \tag{4.9}$$

gdzie $\mu = 0{,}0001\,i$ to współczynnik uczenia dla i-tej cechy. Algorytm ten, pomimo swojej prostoty, cechuje się bardzo dużą trafnością rekomendacji.

4.4.2. Rekomendator

Użytkownicy portali rekomendujących filmy zwykle poszukują odpowiedzi na pytanie, jaki film mogliby obejrzeć. Zazwyczaj ich oczekiwania są bardziej sprecyzowane: tym filmem ma być komedia albo melodramat, film konkretnego reżysera bądź taki, w którym występuje jakiś ulubiony aktor. Często pojawia się również problem znalezienia propozycji na wspólny seans ze znajomymi.

W niniejszym podrozdziale opisano rozwiązanie, które powstało z myślą spełnienia tak szczegółowych wymagań, adresowane do użytkowników serwisu Filmaster.pl. Rozwiązanie to przyjęło postać strony internetowej „Rekomendator" i zostało udostępnione pod adresem `http://diablo.ict.pwr.wroc.pl/~msawick2/rekomendator/` (dostęp: czerwiec 2013). Powstanie strony było możliwe dzięki otwartemu dostępowi do zasobów serwisu Filmaster.pl poprzez funkcjonalne API (*Filmaster.pl API 1.1 documentation*, `http://filmaster.org/static/doc/html/index.html`) oraz dzięki testowej aplikacji webowej autorstwa Mariusza Kryńskiego. Na rys. 4.2 pokazano podstawowy wygląd strony.

Rys. 4.2: Widok na świeżo załadowaną stronę „Rekomendator”.

Użytkownicy

Bezpośrednio po załadowaniu strona jest uruchamiana w trybie odpowiadającym domyślnemu użytkownikowi „Ranking”. Użytkownik ten widnieje w bazie danych serwisu Filmaster.pl pod nazwą „blog”. Rekomendacje jemu przypisane przedstawiają listę najlepiej ocenianych filmów w serwisie.

Pole tekstowe znajdujące się na górze strony pozwala na dodawanie kolejnych użytkowników serwisu. Pola wyboru przy nazwach użytkowników pozwalają na przełączanie się pomiędzy wczytanymi osobami (rys. 4.3).

Przy udzielanych rekomendacjach wyświetlane są dodatkowo informacje kontekstowe, które mają wspomóc użytkownika w podjęciu właściwej decyzji, a niekoniecznie wchodzą do procesu rekomendacji jako jej parametr. Takimi informacjami są np. średnia ocena danego filmu, rok premiery czy nazwisko reżysera. Lista tagów widoczna z prawej strony panelu pozwala na filtrowanie wyników, aby można było określić oczekiwany typ polecanych filmów. Pozwala to na dodatkowe zawężenie zbioru prezentowanych rekomendacji do takich, które najbardziej odpowiadają potrzebom użytkownika.

Wspólne rekomendacje

Po dodaniu wielu użytkowników serwisu możliwe jest wyświetlenie rekomendacji wspólnych dla całej wybranej grupy. Problem scalenia rekomendacji dla wielu osób można rozwiązać na różne sposoby. „Rekomendator” nie wywołuje algorytmów rekomendujących, a jedynie korzysta z zasobów już znajdujących się w bazie serwisu Filmaster.pl. Dlatego też wspólne rekomendacje stanowią filmy należące do części wspólnej zbiorów filmów polecanych poszczególnym użytkownikom. Metoda ta zapewnia umieszczenie na rekomendowanej liście filmów,

które powinny podobać się wszystkim członkom grupy. Niestety, jej dużą wadą jest proponowanie zwykle bardzo niewielkiej liczby filmów.

Na poziomie algorytmu można byłoby zaproponować inne rozwiązanie: dla użytkowników biorących udział w tworzeniu wspólnej listy należy stworzyć jeden wektor wagowy (4.8). W takim podejściu wynikowa lista powinna być obszerniejsza i bardziej zróżnicowana, jednak jej stworzenie może przysporzyć wielu problemów. Już na początku pojawia się pytanie, jak stworzyć wektory wagowe. Rozwiązaniem może być wykorzystanie filmów ocenionych przez wszystkich użytkowników z grupy z ocenami będącymi średnią ocen udzielonych przez poszczególne osoby. Tu pojawiają się jednak kolejne problemy: w przypadku większej grupy zbiór takich filmów może być niedostatecznie duży – dlatego też uwzględnić można np. filmy ocenione przez więcej niż połowę grupy. Kolejną trudnością jest zapewnienie, by na liście nie pojawił się film, który może się niektórym użytkownikom wyjątkowo nie podobać – sytuacja taka jest szczególnie prawdopodobna, jeżeli tylko jedna osoba w grupie nie lubi jakiegoś gatunku filmu.

Przykład działania

Na rys. 4.2, 4.3 i 4.4 pokazano widok strony na różnych etapach jej użycia:

1. uruchomienie strony – widok podstawowy (rys. 4.2),
2. dodanie do listy dwóch użytkowników „wampir" i „mroczna_elfka" i wyświetlenie wspólnych rekomendacji,
3. odznaczenie domyślnego użytkownika „Ranking" (rys. 4.3),
4. filtrowanie uzyskanych rekomendacji pod kątem wybranych tagów – w tym wypadku chodzi o filmy animowane (rys. 4.3).

Ostateczne wyniki pokazano na rys. 4.4 – w tym wypadku rekomendowane są filmy „Epoka Lodowcowa", „Odlot" oraz „Iniemamocni".

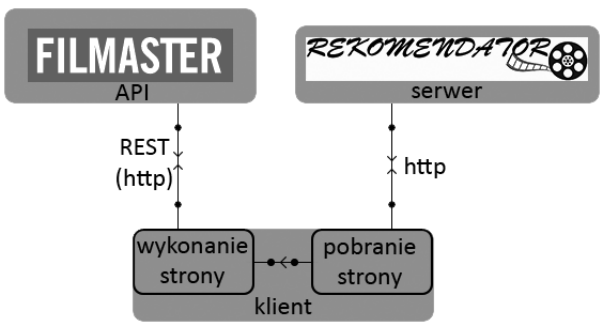
Implementacja

„Rekomendator", aby realizować postawione przed nim zadania, musi na bieżąco obsługiwać zmiany dokonane przez użytkownika oraz mieć umiejętność komunikowania się z bazą danych serwisu Filmaster.pl w celu pobrania wymaganych informacji. Serwis Filmaster.pl w swoich zasobach przechowuje m.in. szczegółowe informacje na temat filmów oraz rekomendacji użytkowników.

„Rekomendator" został napisana przy użyciu języków JavaScript, HTML oraz CSS. Na szczególną uwagę zasługuje wykorzystanie jQuery – biblioteki programistycznej dla języka JavaScript, dzięki której zrealizowano obsługę zdarzeń użytkownika oraz komunikację z API serwisu Filmaster.pl. Żądanie pobrania odpowiednich zasobów serwisu odbywa się poprzez wykorzystanie protokołu HTTP, zgodnie z wzorcem architektury REST. Poszczególne informacje identyfikowane są za pomocą adresów URL, a pobierane są w formacie JSON. Przepływ danych schematycznie przedstawiono na rys. 4.5.

Rys. 4.3: Widok na wszystkie wspólne rekomendacje dwóch użytkowników.

Rys. 4.4: Widok na przefiltrowane wspólne rekomendacje dwóch użytkowników.



Rys. 4.5: Przepływ informacji pomiędzy poszczególnymi elementami systemu.

4.5. Podsumowanie

Budowa silników rekomendacji jest dziedziną znaną i ostatnio szeroko rozwijaną. Niebagatelny wpływ na jej popularność ma rola silników rekomendacji w generowaniu zysków. Z tego właśnie powodu firmy komercyjne nie szczędzą starań, by wdrażać coraz to skuteczniejsze rozwiązania, a w miarę zwiększania skuteczności – znajdować nowe obszary potencjalnych ich zastosowań. Proponowane metody rozpinają się od prostych algorytmów rekomendacji (wymagających dostrojenia do konkretnego przypadku użycia) aż po implementacje bardzo skomplikowane procesów biznesowych, obejmujących uruchomienie i agregację wyników kilku algorytmów na raz. Przedstawione w niniejszym rozdziale algorytmy to zaledwie wstęp do tej tematyki.

Opisany „Rekomendator" jest prostą aplikacją, sugerującą użytkownikom tytuły filmów wartych obejrzenia. Na jej przykładzie pokazano, jak wiele możliwości daje wykorzystanie silników rekomendacji. Dzięki przejrzystej prezentacji, możliwości filtracji oraz dodatkowym informacjom o filmie, rekomendacje stają się wyjątkowo przyjazne dla użytkownika, a serwis dostarczający je – atrakcyjny. Aplikację tę warto byłoby rozszerzyć o możliwość generowania rekomendacji skierowanych do wielu użytkowników.

Literatura

[1] C. Anderson. *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion Books. Harper Collins Usa, USA, 2006.

[2] E. Brynjolfsson, Y. J. Hu, D. Simester. Goodbye pareto principle, hello long tail: The effect of search costs on the concentration of product sales. *Management Science*, 57:1373 – 1386, 2011.

[3] G. Adomavicius, A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734 – 749, 2005.

[4] M. J. Pazzani, D. Billsus. Content-based recommendation systems. *The adaptive web*, strony 325 – 341. Springer-Verlag, Berlin, 2007.

[5] A. Rajaraman, J. Ullman. *Mining of Massive Datasets*, rozdział 9. Cambridge University Press, 2011.

[6] J. S. Breese, D. Heckerman, C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, San Francisco, 1998.

[7] I. Konstas, V. Stathopoulos, J. M. Jose. On social networks and collaborative recommendation. *Proc. of the 32nd Int. ACM SIGIR Conf. on Research and development in information retrieval*, strony 195–202, New York, 2009.

[8] M. Balabanović, Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40:66 – 72, 1997.

[9] G. Linden, B. Smith, J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, Jan./Feb. 2003.

ROZDZIAŁ 5

MODELOWANIE I EKSPLORACJA GRAFOWYCH BAZ DANYCH

N. Czop, M. Gawron, M. Orynicz

W niniejszym rozdziale opisano zagadnienia związane z przechowywaniem danych w strukturach o postaci grafu oraz ich zarządzaniem. W pierwszej jego części przedstawiono podstawowe wiadomości oraz zarys ogólnej koncepcji grafowych baz danych. Można dowiedzieć się z niej nie tylko czym są grafowe bazy danych, ale również jak się je obsługuje, jakie dane można w nich przechowywać (oraz jak je reprezentować), jak wygląda ich język zapytań. W kolejnych częściach zmieszczono porównanie wybranych grafowych bazy danych pod względem oferowanych przez nie funkcji oraz opisano zalety i wady grafowych baz danych w porównaniu do baz danych innych typów (np. relacyjnych). Ostatnią część poświęcono zagadnieniom praktycznym, związanym z utworzeniem grafowej bazy danych przy użyciu dedykowanych do tego celu narzędzi. Opisom tam zamieszczonym towarzyszy kompletny samouczek, przeprowadzający czytelnika przez proces tworzenia grafowej bazy danych w systemie Neo4j.

5.1. Grafy a reprezentacja danych

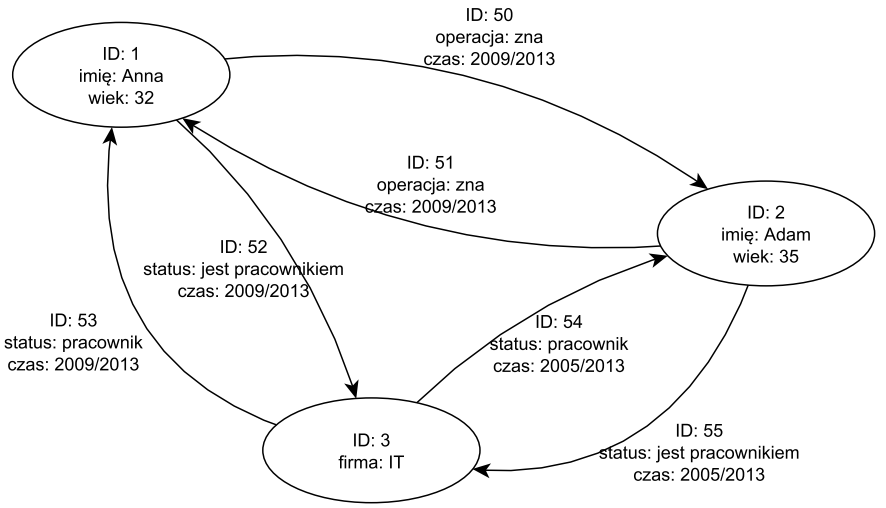
Bazy danych są narzędziami służącymi do reprezentowanie rzeczywistych podmiotów, przechowywania danych o nich i modelowania ich wzajemnych relacji [1]. Z tej racji bazy danych muszą zawierać zbiór struktur typów danych, zbiory operacji oraz reguły wnioskowania, a także zbiór ogólnych zasad integralności opisany matematycznie [2].

Jeśli chodzi o strukturę danych, grafowe bazy danych, zgodnie ze swą nazwą, wykorzystują grafy. Graf to jedna z podstawowych struktur matematycznych definiowana parą $G(V,E)$ lub trójką $G(V,E,\gamma)$, gdzie V jest zbiorem wierzchołków, E zbiorem krawędzi a γ funkcją ze zbioru krawędzi w uporządkowany zbiór par wierzchołków. Grafy w matematyce pojawiają się co najmniej od XVIII wieku, kiedy to Leonhard Euler rozwiązał tzw. zagadnienie mostów królewieckich. W bazach danych pojawiły się w latach 90' XX wieku, po czym stopniowo zaniknęły. Ich

reaktywacja przypada na moment rozpowszechnienia się Internetu wśród użytkowników niekomercyjnych i wzrostu ilości danych do tak dużych rozmiarów, że reprezentowanie ich w postaci relacyjnych baz danych przestało być wydajne i funkcjonalne.

5.1.1. Grafowy model danych

Grafowy model danych pojawił się niemalże równolegle z obiektowym modelem baz danych. Miał on pozwolić na ominięcie ograniczeń podstawowych modeli bazodanowych oraz ułatwić rozwiązywanie kwestii dotyczących dziedziczenia atrybutów. Grafowy model danych wykorzystuje trzy atrybuty grafów: węzły (wierzchołki), krawędzie i właściwości [3] (zobacz przykład na rys. 5.1).



Rys. 5.1: Przykładowy grafowy model danych.

Dane (czy schematy) są reprezentowane albo przez grafy, albo przez struktury uogólniające to pojęcie (np. hypergraf). Podstawowymi jednostkami są etykietowane grafy skierowane, składające się na digraf modelujący całą bazę danych. Manipulacja danymi odbywa się z wykorzystaniem takich właściwości grafów, jak: ścieżki, sąsiedztwo, podgrafy czy statystyka grafów (dzięki czemu operatory języka mogą bazować na porównywaniu wzorców oraz wyszukiwaniu powtarzających się podgrafów).

Ważnym aspektem w bazach danych są więzy integralności, które definiują i egzekwują zbiór spójnych stanów bazy danych i/lub zmian stanów. Wymagają one m.in. unikalnych nazw etykiet, a także wprowadzają pojęcie zależności funkcyjnych. Zależności te są zależnościami semantycznymi, reprezentowanymi na poziomie grafu poprzez krawędzie skierowane. Wyrażenie $A \rightarrow B$, gdzie A, B zbiory atrybutów, mówi, że A określa wartość B we wszystkich węzłach bazy danych.

5.1.2. Zapytania i języki zapytań

Język zapytań, niezależnie od typu bazy danych, jest zbiorem operatorów i reguł wnioskowania, które mogą być zastosowane do dowolnych instancji typów struktur danych modelu, zarówno celem akwizycji, jak i manipulacji danymi zawartymi w owych strukturach [3]. Języki te pozwalają wyrazić jedno z dwóch typów zapytań:

- zapytanie strukturalne – zapytanie o strukturę (postać) grafu, polegające przykładowo na zliczeniu węzłów na jakiejś głębokości grafu lub znalezieniu węzłów spełniających podane warunki (dwie krawędzie wchodzące, jedna wychodząca);
- zapytanie o dane – zapytania służące do działania z danymi zapisanymi w bazie. Przy ich pomocy można zapytać o liczbę węzłów zawierających dane o podanej wartości czy też wyszukać węzły, w których znajdują się dane o wybranym wzorcu tekstowym.

Istnieje spora grupa języków zapytań przeznaczonych do obsługi grafowych baz danych. Jednym z nich jest `Gremlin` (`https://github.com/tinkerpop/gremlin/wiki`), który wspiera Javę i Groovy oraz korzysta z projektu `Blueprints` (`https://github.com/tinkerpop/blueprints/wiki`). Projekt `Blueprints` jest zbiorem interfejsów, wdrożeń oraz zestawów testów dla grafowych baz danych.

`Gremlin` pomaga w manualnej pracy z grafami danych (manipulacji danymi) i utrzymaniu integralności danych. Umożliwia łatwe przechodzenie przez zawarte w bazie informacje oraz pozwala na aktualizację właściwości wierzchołków, dodawanie krawędzi, usuwanie wierzchołków. Co więcej, w `Gremlin`ie wiele typów kwerend może być wyrażone w bardziej zwięzły i zrozumiały sposób niż ma to miejsce w tradycyjnych językach programowania. To co w Javie zajęłoby wiele linii kodu, w `Gremlin`ie zajmie tylko kilka.

`Gremlin` został przystosowany do współpracy z algorytmami JUNG oraz językiem zapytań SPARQL. Dodatkowo może on być używany do eksploracji sieci semantycznych, a możliwość jego rozszerzania pozwala dopasować go do konkretnego przypadku (bazę można rozszerzyć np. o nowe metody).

`SPARQL` umożliwia formułowanie zapytań dotyczących źródeł danych przechowywanych w RDF, w wyniku których otrzymuje się zbiór grafów RDF. `SPARQL` opisano w specyfikacjach W3C (`http://www.w3.org/TR/rdf-sparql-query/`).

Rodzina języków `G`, `G+` oraz `GraphLog` umożliwia tworzenie grafów zapytań. Dla języka `G` przyjmują one postać zbioru etykietowanych multigrafów, o węzłach stałych lub zmiennych. Wynikiem zapytania jest suma wszystkich grafów zapytań, dopasowanych do podgrafów z instancji grafu (bazy).

5.2. Zastosowanie

Grafowe bazy danych powstały w celu czytelniejszej reprezentacji danych i ich wzajemnych zależności. Są one odpowiedzią na ograniczenia istniejących języ-

ków zapytań do baz danych. Można powiedzieć, że grafowe bazy danych uogólniają klasyczne modele bazodanowe. Dzięki nim użytkownik może niejako „zobaczyć" dane i lepiej zrozumieć ich wzajemne zależności (wykorzystując semantykę i strukturę grafu). Dodatkowo, bazy te pozwalają reprezentować zarówno dane, jak również umożliwiają manipulację nimi przy pomocy struktury jednego typu – grafu właśnie.

Pomimo posługiwania się stosunkowo złożonym modelem informacyjnym, grafowe bazy danych są bardzo elastyczne i pozwalają na łatwą obsługę nowych danych. Mają zastosowanie w graficznych i wizualnych aplikacjach dla systemów multimedialnych, obrazowych i geograficznych. Ponadto znajdują zastosowanie w sieciach informatycznych, technologicznych czy też biologicznych i społecznych (gdzie węzły grafu reprezentują ludzi i grupy, a przepływy między węzłami modelują relacje międzyludzkie, tj. przyjaźnie, współzawodnictwo, kontakty biznesowe).

5.3. Przegląd istniejących rozwiązań

Istnieje wiele gotowych implementacji grafowych bazy danych. Poniżej przedstawiono najpopularniejsze z nich.

5.3.1. Opis wybranych implementacji

AllegroGraph (`http://www.franz.com/agraph/allegrograph/`) – jest bazą danych wspierającą SPARQL, RDFS++ oraz wnioskowanie w Prologu. Wykorzystuje dyskowy magazyn danych oraz pozwala na uzyskanie dużej wydajności przy przechowywaniu bilionów trójek RDF. Aktualna wersja na dzień 23.03.2013 to 4.1. Od wersji 4.0 do `AllegroGraph` dodano automatyczne indeksowanie, transakcje i 100% współbieżność operacji odczytu.

DEX (`http://www.dama.upc.edu/technology-transfer/dex`) – jest wysoce wydajną biblioteką do zarządzania dużymi grafami i sieciami. Pozwala na przechowywanie skierowanych multigrafów etykietowanych. Więzy integralności obejmują relacje, atrybuty dziedzinowe, typy węzłów i krawędzi.

Neo4j (`http://www.neo4j.org/`) – jest projektem typu *open-surce*, zaimplementowanym w Javie, ocenianym jako najpopularniejsza grafowa baza danych. Zgodnie z opisem twórców posiada wbudowany, dyskowy, w pełni transakcyjny silnik przechowujący dane w postaci grafów (a nie tabel). Ponadto dla operacji związanych z danymi `Neo4j` pracuje tysiące razy szybciej niż relacyjne bazy danych. Jego kolejnymi atutami mają być: niezawodność, w pełni zaimplementowane transakcje ACID (*atomicity* – atomowość, *consistency* – spójność, *isolation* – izolacja, *durability* – trwałość), skalowalność (do kilku miliardów urządzeń), łatwo zrozumiały język zapytań, prosty interfejs i obiektowo zorientowane API.

Sones GraphDB (`https://github.com/sones`) – to projekt komercyjny, w całości zaimplementowany w języku C# i działający jedynie na platformie

.NET. Istnieje jednak jego *open-surce*'owy odpowiednik, działający na platformie `Mono` (alternatywna implementacja platformy .NET). W bazie danych `Sones GraphDB` nie jest wymagany indeks globalny relacji między węzłami. Realizuje ona również podejście pół-strukturalne do danych. Dane niestrukturyzowane mogą być przesyłane do danych strukturyzowanych i na odwrót. Kolejną zaletą tej bazy danych jest możliwość dynamicznej rozbudowy, czyli dodanie (też usunięcie) dodatkowych węzłów czy właściwości podczas działania bazy. Dedykowany język zapytań Sones GraphQL jest określony jako „przyjazny dla użytkownika SQL dla grafów".

Titan (`http://thinkaurelius.github.com/titan/`) – podobnie do `Neo4j` jest wysoce skalowalną grafową bazą danych typu *open-surce*, przeznaczoną do przechowywania i odpytywania grafów o miliardach wierzchołków i krawędzi. `Titan` może obsługiwać tysiące użytkowników w tym samym czasie. Dodatkowo wspiera transakcje ACID i zapewnia spójność danych, pozwala ponadto tworzyć tzw. gorące kopie zapasowe poprzez wykorzystanie *Apache Cassandra*, *Apache HBase* oraz *berkeleyDB Oracle*.

VertexDB (`http://www.dekorte.com/projects/opensource/vertexdb/`) – jest wysokiej wydajności serwerem grafowych baz danych z funkcją automatycznego zbierania śmieci (ang. *garbage collection*). Do zapytań używa protokołu HTTP, formatem odpowiedzi jest JSON. API jest wzorowane na systemie FUSE (ang. *Filesystem in USErspace*), wzbogaconym o kilka dodatkowych metod zapytań i kolejek.

5.3.2. Porównanie wybranych cech grafowych baz danych

Bazy danych można porównywać nie tylko ze względu na takie cechy jak transakcyjność czy współbieżność. Dla zwykłego użytkownika ważnym aspektem są obsługiwane języki programowania oraz języki zapytań. Nie bez znaczenia dla łatwości korzystania z bazy ma jej API. W tab. 5.1 zamieszczono porównanie wybranych cech i właściwości przedstawionych wcześniej grafowych baz danych ('-' oznacza brak informacji).

5.4. Grafowe bazy danych a inne typy baz

Poniżej przedstawiono krótkie porównanie popularnych typów baz danych z bazami grafowymi. Porównanie to ma na celu wskazać sytuacje, w jakich grafowe bazy danych są dobrym rozwiązaniem oraz kiedy należy skorzystać z innego rodzaju systemu składowania danych.

5.4.1. Bazy relacyjne

Charakterystyka

Model relacyjny i język SQL są podstawą implementacji najczęściej stosowanych obecnie baz danych. Przetwarzanie danych w tych bazach opiera się na

Tab. 5.1: Porównanie wybranych cech grafowych baz danych.

	Allegro-Graph	DEX	Neo4j	Sones GraphDB	Titan	VertexDB
Baza danych						
ACID	tak	-	tak	nie	tak	-
mapowanie obiektowe	-	nie	nie	tak	tak	nie
rozproszona	-	nie	częściowo (RMI)	tak	tak	nie
silnik	-	własny	własny	własny	-	Tokyo Cabinet
transakcyjność	tak	nie	tak	nie	tak	tak
współbieżność	tak	tak	-	-	-	
Zapytania						
język zapytań	SPARQL	brak	Gremlin	własny (GQL)	Gremlin	JSON
przechodzenie grafu	-	tak	tak	nie	-	nie
Języki programowania						
Clojure	nie	nie	tak	nie	-	-
C++	nie	tak	nie	nie	-	-
C#	tak	nie	nie	tak	-	-
Erlang	nie	nie	tak	nie	-	-
Java	tak	tak	tak	nie	-	-
Lisp	tak	nie	nie	nie	-	-
Perl	tak	nie	nie	nie	-	-
PHP	nie	nie	tak	nie	-	-
Python	tak	-	tak	nie	-	-
REST	tak	nie	tak	nie	-	-
Ruby	tak	nie	tak	nie	-	-
API, platforma, licencje, charakter						
API	REST	Java	Java embedded/ REST	.NET embedded/ REST/ WebServices	Blueprints	HTTP/ JSON
platforma systemowa	Linux		JVM	.NET	-	Linux/ Unix
licencja	-	-	AGPLv3	SaaS	-	Revised BSD
komercyjne	tak	tak	tak	tak	-	nie
open-source	nie	nie	tak	tak	-	tak

wykorzystaniu algebry relacji [4], zaś ich projektowanie wymaga dopracowania pełnego schematu, co jest procesem skomplikowanym [5].

W niektórych przypadkach trudno wymagać, aby na etapie projektowania znana już była dokładnie struktura przetwarzanych danych. Co prawda możliwe jest modyfikowanie schematu na etapie eksploatacji bazy danych, jednak jest to operacja kosztowna, wymagająca modyfikacji wszystkich ustalonych już zależności. W trakcie projektowania relacyjnego schematu dąży się przede wszystkim do wyeliminowania redundancji danych oraz ich złożoności. Dokonuje się więc normalizacji schematu [5]. Dzięki temu np. przy późniejszej aktualizacji zmiana jest wprowadzana tylko w jednym miejscu. Kolejnym czynnikiem, jaki należy brać pod uwagę, jest zestawienie danych w samych tabelach. Należy to zrobić tak, by podczas użytkowania ograniczyć ilość operacji łączenia (ang. *join*) wierszy tabel. Jest to kosztowna obliczeniowo operacja, sprowadzająca się do utworzenia kombinacji wszystkich wierszy z wybranych tabel i odfiltrowania tych, które nie spełniają zadanego kryterium.

Bazy relacyjne a grafowe

Grafowe bazy danych spisują się o wiele lepiej od baz relacyjnych, gdy dane są ze sobą mocno powiązane i trzeba dokonać głębokiej analizy tych powiązań [6]. W grafowej bazie danych możliwe jest nisko kosztowe przemieszczanie się po kolejnych stopniach powiązań, odpytywanie oparte o te powiązania i generalnie wszystko, co powiązań dotyczy. Każde zapytanie typu „wypisz przyjaciół moich przyjaciół moich przyjaciół", albo „znajdź jakie smaki lodów kupują klienci o danym kodzie pocztowym, którzy kupują również kawę rozpuszczalną" generuje dużo większe obciążenie w relacyjnych bazach danych niż w bazach grafowych. Z drugiej strony płytkie zapytania, takie jak odszukanie wszystkich pracowników danego działu firmy lub zsumowanie przychodów w bieżącym roku podatkowych prawdopodobnie szybciej zostałyby obsłużone w bazach relacyjnych niż grafowych.

5.4.2. Bazy obiektowe

Charakterystyka

Bazy obiektowe są projektowane z wykorzystaniem tego samego paradygmatu, co paradygmat stosowany podczas tworzenia aplikacji w popularnych obecnie obiektowych językach programowania. Bazy te pozwalają na tworzenie własnych typów danych (klas) oraz na modelowanie skomplikowanych obiektów w sposób naturalny [4]. Pozwalają one również na łatwe śledzenie powiązań pomiędzy poszczególnymi instancjami obiektów za pośrednictwem referencje w nich zawartych, o ile zostały one przewidziane podczas projektowania danej klasy. Dodatkowym atutem jest to, że podczas programowego odczytu i zapisu obiektów w bazie nie jest konieczna ich konstrukcja i dekonstrukcja - obiekt używany w aplikacji korzystającej z bazy danych jest taki sam, jak ten przechowywany w bazie. Z drugiej strony mechanizm ten powoduje silne powiązanie warstwy logiki biznesowej z warstwą danych tworzonych aplikacji, utrudniając póź-

niejsze ich zmiany. Ponadto w bazach obiektowych występuje dość duże uzależnienie od pierwotnie założonego rozkładu danych w postaci zaprojektowanych klas obiektów.

Bazy obiektowe a grafowe

Istnieje pewne podobieństwo pomiędzy obiektowymi a grafowymi bazami danych z racji przechowywanych w obydwu typach bezpośrednich odnośników do powiązanych bytów. Bazy obiektowe nie są jednak ograniczone co do informacji na temat typu powiązania (który musi być zapisany bezpośrednio w jednym bądź obu połączonych obiektach), a w bazach grafowych informacja ta jest przechowywana bezpośrednio w krawędzi. Oczywiście możliwe jest utworzenie w bazie obiektowej obiektu klasy krawędź, odpowiedzialnego za przechowywanie takich informacji. Jednak taka baza nie będzie posiadała wbudowanych algorytmów do przeszukiwania zamodelowanych zależności. Grafowe bazy danych zaś dysponują do tego odpowiednimi mechanizmami.

Obydwa rodzaje baz pozwalają na samodzielne definiowanie złożonych typów danych. Jednak typy te w bazach obiektowych są ściśle powiązane z aplikacją korzystającą z bazy, zaś w bazach grafowych mogą być całkowicie od niej niezależne. Dzięki temu możliwe jest obsługiwanie pojedynczej grafowej bazy za pomocą kilku niezależnych aplikacji. Dodatkowo, możliwa jest dowolna modyfikacja zawartości poszczególnych wierzchołków czy krawędzi odpowiednio do potrzeb, co jest wygodniejsze od konieczności modyfikacji całych klas bądź dziedziczenia w bazach obiektowych.

Podsumowując, bazy obiektowe pozwalają na uproszczenie modelowania skomplikowanych zjawisk i bytów oraz przenoszenie zbudowanych obiektów pomiędzy warstwą logiki biznesowej a warstwą danych w ramach tworzonych aplikacji. Bazy grafowe nie tyle służą do modelowaniu bytów, co do modelowania i eksploracji powiązań pomiędzy tymi bytami.

5.4.3. NoSQL

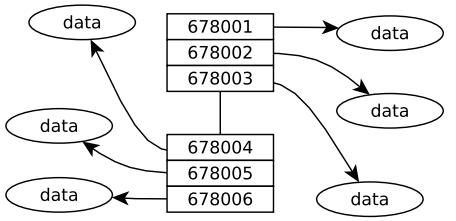
Rodzina nie tylko SQL-owych baz danych NoSQL (ang. *Not Only SQL*) jest alternatywą dla klasycznych baz relacyjnych. W bazach NoSQL stosuje się inne podejście do przechowywanych danych, polegające na rozluźnieniu ograniczeń narzuconych na kształt przechowywanych danych, czy wymagania dotyczące spójności. Same grafowe bazy danych zaliczają się do tej rodziny.

Systemy typu Klucz-Wartość

Charakterystyka

Systemy typu Klucz-Wartość są systemami przystosowanymi do przechowywania dużych ilości danych w tablicach asocjacyjnych [6]. W czystej postaci „nie rozumieją" przechowywanych danych. Ich jedynym zadaniem jest zapewnienie szybkiego dostępu do informacji zapisanych pod zadanym kluczem oraz zabezpieczenie przed ich utraceniem. Z racji braku powiązań pomiędzy poszczególnymi wpisami, tego typu bazy danych łatwo skalują się na wiele maszyn. Z drugiej strony próba wyekstrahowania nawet najmniejszego kawałka danego wpisu

wymaga pobrania całego obiektu zapisanego pod danym kluczem, zewnętrznego zinterpretowania, zmodyfikowania i ponownego zapisania. Przykład bazy danych tego typu przedstawiono na rys. 5.2.



Rys. 5.2: Przykład bazy danych typu Klucz-Wartość.

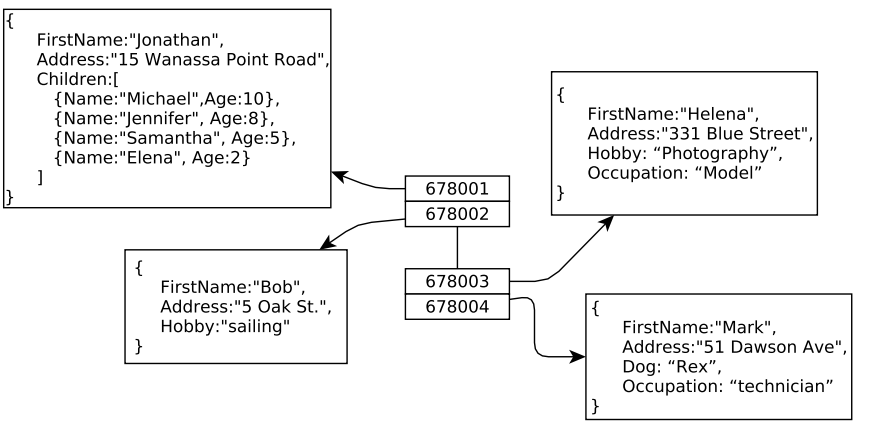
Bazy klucz-wartość a bazy grafowe

Ten rodzaj systemu przechowywania danych, podobnie jak bazy grafowe, nie wymaga określenia z góry struktury przechowywanych danych. Z drugiej strony nie przechowuje on także wprost połączeń pomiędzy przechowywanymi obiektami. O ile grafy skupiają się na przechowywaniu relacji pomiędzy zapisanymi w nich obiektami, bazy klucz-wartość skupiają się na przechowywaniu czegokolwiek, w dużych ilościach, z szybkim dostępem do dowolnego elementu, do którego znany jest klucz.

Bazy zorientowane na dokumenty

Charakterystyka

Dokumentowe bazy danych są nastawione na przechowywanie niestrukturyzowanych elementów zwanych dokumentami, zawierającymi zagnieżdżające się pola danych oraz listy takich pól. Wygląd dokumentowej bazy danych zilustrowano na rys. 5.3. Odwoływanie się do poszczególnych dokumentów może odbywać się przez klucz lub przez zapytanie dotyczące zawartości [6]. Można więc



Rys. 5.3: Wizualizacja wyglądu dokumentowej bazy danych.

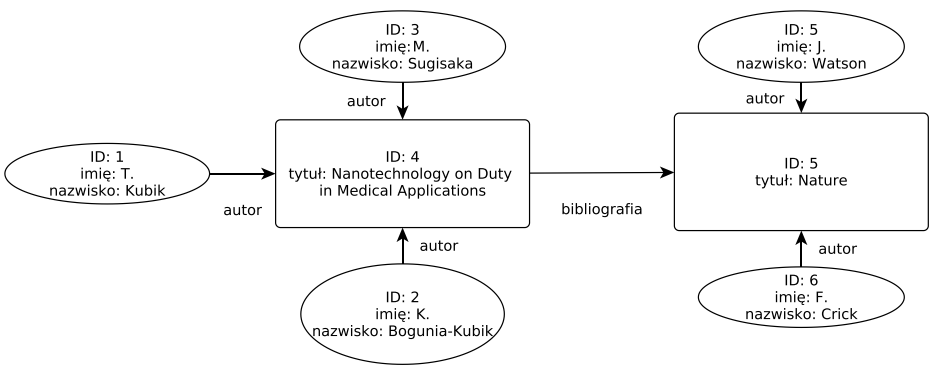
przyjąć, że jest to modyfikacja systemu klucz-wartość, w którym przez częściowe skodyfikowanie przechowywanych danych (dokument musi mieć zadany format) system jest w stanie zrozumieć przechowywane informacje i wykorzystywać przechowywane wartości. Dokumentowe bazy danych, tak jak systemy składowania klucz-wartość, nie przechowują relacji pomiędzy zapisanymi w nich dokumentami i są przystosowane do łatwego skalowania na wiele maszyn wraz ze wzrostem objętości przechowywanych danych.

Bazy dokumentowe a bazy grafowe
Co prawda rozumienie przechowywanych danych pozwala na przechowywanie np. kluczy do dokumentów, do których się odwołują, operacje śledzenia relacji w bazach dokumentowych wciąż są bardziej kosztowne niż w grafowych. Dodatkowo, bazy te nie posiadają natywnego systemu zapewniającego zachowanie spójności takich relacji, co nakłada dodatkowe obciążenie na aplikację kliencką. Z drugiej strony, w wypadku danych słabo ze sobą powiązanych, dla których liczy się przede wszystkim ilość czy szybkość dostępu, dokumentowa baza danych może być lepszym rozwiązaniem niż inne bazy danych.

5.5. Przykładowa implementacja grafowej bazy danych

Celem niniejszego podrozdziału jest zademonstrowanie procesu tworzenia grafowej bazy danych opartej na systemie `Neo4j`. Opisana aplikacja pozwoli naświetlić sposób tworzenia zapytań skierowanych do serwerów grafowych baz danych w języku `Gremlin`, których obsłużenie byłoby bardzo czasochłonne w relacyjnych bazach danych. Na jej przykładzie zostanie również pokazane, w jaki sposób dodać do bazy własne dane i jak je odpowiednio powiązać.

Przetwarzanym zasobem pierwotnie miała być baza danych przechowująca relacje między pracami naukowymi publikowanymi na Politechnice Wrocławskiej i wykorzystaną w nich bibliografią. Przykładowy graf zależności między jedną pracą i pojedynczym elementem z jej bibliografii pokazano na rys. 5.4. Baza ta miała umożliwiać wyszukiwanie prac, które są najpopularniejsze wśród naukowców z Politechniki Wrocławskiej. Ponadto miała pozwalać na odszukanie

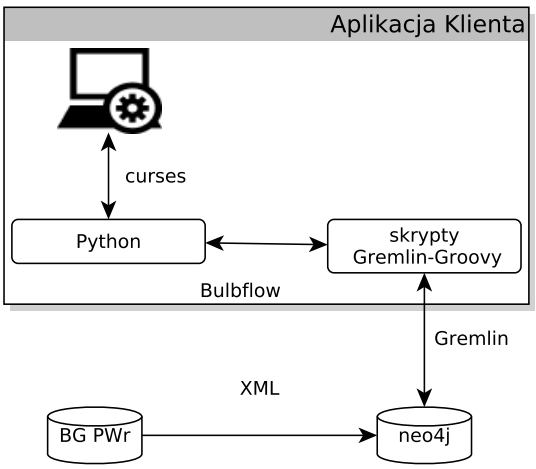


Rys. 5.4: Przykładowe relacje pomiędzy pracami naukowymi i autorami.

badaczy opierających swoje publikacje na tych samych pracach lub innych pracach tych samych autorów, czyli grup naukowców pracujących nad bardzo podobnymi zagadnieniami.

Założono, że dane zostaną pozyskane z Biblioteki Głównej Politechniki Wrocławskiej. Niestety, ze względu wrażliwość tych informacji (daty urodzenia, miejsca zamieszkania itd.) ostatecznie nie udało się uzyskać do nich dostępu. Dlatego zamiast pełnego zbioru danych przygotowano skromny zbiór testowy.

W ogólnym zarysie dane pozyskane z relacyjnej bazy danych miały być zaimportowane do bazy grafowej, opartej na serwerze `Neo4j`. Komunikacja pomiędzy bazami miała odbywać się jednostronnie, za pomocą plików w formacie `XML`, które przenosiłyby niezbędne dane. Zarys architektury tego systemu pokazano na rys. 5.5. Dalszy jego rozwój mógłby doprowadzić do migracji dotychczasowej bazy danych Biblioteki Głównej do bazy grafowej, która powinna spełniać dotychczas udostępnione funkcje równie dobrze jak pierwowzór. Ponadto wiele funkcji przedstawionych w następnym rozdziale z pewnością byłyby łatwiejsze do zaimplementowania i działałyby szybciej na nowo utworzonej grafowej bazie danych, niż na relacyjnym odpowiedniku.



Rys. 5.5: Schemat planowanych w projekcie interfejsów oraz powiązań pomiędzy modułami.

5.5.1. Instalacja bazy Neo4j

Baza danych `Neo4j` dostępna jest w trzech wersjach: *Community*, *Advenced* i *Enterprise*. Na potrzeby przykładu została wybrana najprostsza, bezpłatna wersja *Community*, której uruchomienie nie wymaga instalacji ani rejestracji na stronie producenta. Wymagane jest jedynie pobranie odpowiednich plików z oficjalnej strony dystrybutora (`www.neo4j.com`) i, w przypadku systemów z rodziny Linux, uruchomienia bazy z wiersza poleceń komendą `./neo4j start`. Polecenie to uruchomia serwer `Neo4j` z możliwością kontroli poprzez stronę http (standar-

dowo zlokalizowaną pod adresem `http://localhost:7474/webadmin/`). Strona ta umożliwia dostęp do narzędzi pozwalających na obserwowanie i edytowanie bazy danych. Po uruchomieniu programu stworzona jest pierwsza pusta baza danych (zlokalizowana domyślnie w katalogu `./var/db/neo4j/graph.db`).

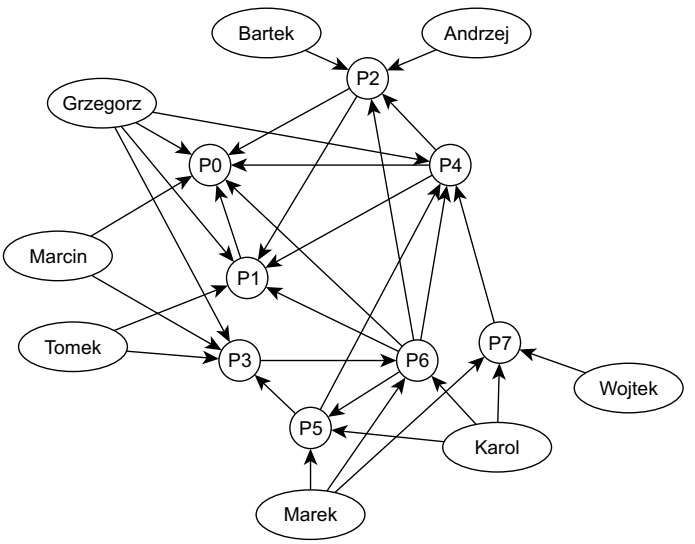
5.5.2. Implementacja

Aby umożliwić użytkownikom komunikację z bazą danych został stworzony program napisany w języku Python 2, wykorzystujący framework `Bulbflow` wraz z językiem zapytań `Gremlin` do komunikacji z bazą danych `Neo4j`. Stworzony program umożliwia wyszukanie:

- wszystkich prac danego autora,
- wszystkich autorów danej pracy,
- najczęściej cytowanych prac,
- najczęściej cytowanych autorów,
- autorów, którzy opierają swoje prace na tych samych pracach,
- autorów, którzy opierają swoje prace na pracach tych samych autorów.

Tworzenie bazy danych

Na rys. 5.6 pokazano prostą bazę danych złożoną z 8 prac i 8 autorów. Baza ta została stworzona za pomocą skryptu napisanego w języku Python 2. W tym celu utworzono 2 klasy odpowiedzialne za wierzchołki w grafie: `Paper` i `Person` modelujące, odpowiednio, publikacje i autorów. Krawędzie w grafie są modelowane przez klasy `Wrote` i `Cite`. Klasa `Wrote` jest przeznaczona do łączenia wierzchołków pochodzących z klasy `Person` z wierzchołkami `Paper`, a więc określenia autorów prac. Krawędź klasy `Wrote` skierowana jest w stronę wierzchołka klasy



Rys. 5.6: Przykładowe relacje pomiędzy pracami naukowymi i autorami.

`Paper`. Klasa `Cite` łączy ze sobą instancje klasy `Paper`, modelując zależności bibliograficzne pomiędzy pracami. Wierzchołek odpowiadający pracy, która cytuje drugą pracę, jest wyjściem krawędzi.

```
from bulbs.model import Node, Relationship
from bulbs.property import String

class Person(Node):
    element_type = "person"
    name = String(nullable=False)
    university = String(nullable=True)

class Paper(Node):
    element_type = "paper"
    name = String(nullable=False)

class Wrote(Relationship):
    label = "wrote"

class Cite(Relationship):
    label = "cite"
```

Skrypt odpowiedzialny za utworzenie przykładowej bazy danych tworzy najpierw wszystkie wierzchołki (zarówno odpowiedzialne za prace jak i za autorów), a następnie łączy je ze sobą, tworząc zamierzony graf reprezentujący niewielką kolekcje prac naukowych. Przyjęto taką właśnie metodę, gdyż nie wymaga ona sprawdzania, czy istnieje już dany autor w bazie danych przy dodawaniu nowych publikacji. Jednak nic nie stałoby na przeszkodzie, aby została zaimplementowana metoda dodająca pracę i autorów, sprawdzająca czy dane prace lub autorzy znajdują się już w bazie danych.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from libraryClasses import Person, Paper, Wrote, Cite
from bulbs.neo4jserver import Graph
import operator
import os

def createNewGraph():
  personList = ["Grzegorz", "Marcin", "Tomek", "Bartek",
                "Andrzej", "Wojtek", "Marek", "Karol"]
  papersList = ["P0", "P1", "P2", "P3", "P4",
                "P5", "P6", "P7"]
  peopleList = [[0, 1], [0, 2], [3, 4], [0, 1, 2],
                [0], [6, 7], [6, 7], [5, 6, 7]]
  citeList = [None, [0], [0, 1], [6], [0,1,2], [3,4],
              [0,1,2,4,5], [4]]
```

```
g = Graph()

g.clear()
g.add_proxy("person", Person)
g.add_proxy("paper", Paper)
g.add_proxy("wrote", Wrote)
g.add_proxy("cite", Cite)

personAdded = []
paperAdded = []

for x in personList:
  personAdded.append( g.person.create(name=x) )

for x in papersList:
  paperAdded.append( g.paper.create(name=x) )

for i~in range(len(papersList)):
  for x in peopleList[i]:
    g.wrote.create(personAdded[x], paperAdded[i])
  if citeList[i] != None:
    for x in citeList[i]:
      g.cite.create(paperAdded[i], paperAdded[x])
```

Trawersowanie

Przeszukiwanie grafowej bazy danych odbywa się za pomocą przechodzenia z jednego wierzchołka do drugiego poprzez łączące je krawędzie grafu. Zachowanie takie jest określane jako trawersowanie bazy danych. Obecnie nie istnieją łatwe w obsłudze ani dobrze spopularyzowane frameworki służące do tego celu, które wspierałyby język Python. Dlatego wybrano `Bulbflow`, który umożliwia bezpośrednie wykonywanie skryptów napisanych w języku `Gremlin` z poziomu programu. Pozwala to sprawnie i szybko poruszać się po grafie. Na potrzeby przykładu zostały napisane następujące skrypty Gremlin-Groovy, wykonujące założone zadania:

1. `allPapers` - funkcja zwraca listę wartości pola `name` wszystkich wierzchołków określających publikację, czyli tych których pole `element_type` ma wartość `paper`.

```
def allPapers() {
  return g.V.has('element_type', 'paper').name.toList()
}
```

2. `allPeople` - identycznie jak w `allPapers` z tą różnicą, że teraz szukani są autorzy publikacji.

```
def allPeople() {
  return g.V.has('element_type', 'person').name.toList()
}
```

3. `personPapersByName` - funkcja wyszukuje najpierw wierzchołki należące do autorów o zadanym nazwisku, po czym przemieszcza się po wszystkich krawędziach o wartości pola `label = wrote` do prac powiązanych z autorami. Na końcu zwraca tytułu (pole `name`) publikacji.

```
def personPapersByName( name ) {
  return g.V.has('element_type', 'person')
          .has('name', name).out('wrote').name
}
```

4. `authorsOfPaperByName` - podobnie jak w przypadku `personPapersByName` najpierw wyszukuje prace o odpowiednim tytule, po czym przemieszcza się po krawędzi wchodzącej do tych wierzchołków o wartości pola `label = wrote` do wierzchołków odpowiedzialnych za ich autorów. Funkcja zwraca wartości pól `name`.

```
def authorsOfPaperByName( name ) {
  return g.V.has('element_type', 'paper')
          .has('name', name).in('wrote').name
}
```

5. `mostCitedPapers` - funkcja dla wszystkich wierzchołków zawierających publikacje wyszukuje ich referencje (`out('cite')`), po czym zlicza ich nazwy `name`, a wynik zapisuje w mapie `m`.

```
def mostCitedPapers() {
  m = [:]
  g.V.out('cite').name.groupCount(m).toList()
  m = m.sort{a,b -> a.value <=> b.value}
  return m
}
```

6. `mostCitedAuthors` - działa podobnie jak `mostCitedPapers`, z tym wyjątkiem, że tym razem wyszukiwani i zliczani są autorzy prac.

```
def mostCitedAuthors() {
  m = [:]
  a~= g.V.in('wrote').name.groupCount(m).toList()
  m = m.sort{a,b -> a.value <=> b.value}
  return m
}
```

7. `sameReferancesAsByName` - funkcja wyszukuje autorów prac, którzy pisząc swoje prace wykorzystują te same pozycje bibliograficzne co zadany autor. W tym celu najpierw jest wyszukiwany wierzchołek odpowiedzialny za autora z prawidłową nazwą, po czym zapytanie przesuwa się do referencji jego prac (zapisując do zmiennej `x` prace, które on sam napisał). Następnie znajdywane są prace, które wykorzystują te same referencje, wykluczając z nich prace zadanego autora (`in('cite').except(x)`). Potem funkcja przesuwa się do wierzchołków odpowiedzialnych za autorów tych prac, zlicza ich wystąpienia i zapisuje wynik w mapie `m`. Metoda `dedup` usuwa duplikaty z wyniku.

```
def sameReferancesAsByName( name ) {
  m = [:]
  x = []
  g.V.has('name', name)
        .out('wrote').store(x)
        .out('cite')
        .in('cite').except(x).dedup()
        .in('wrote').name.groupCount(m).toList()
  m = m.sort{a,b -> a.value <=> b.value}
  return m
}
```

8. `sameAuthorReferancesAsByName` - działa podobnie do funkcji `sameReferancesAsByName`. Jej analizę pozostawiono jako proste ćwiczenie dla czytelnika.

```
def sameAuthorReferancesAsByName( name ) {
  m = [:]
  x = []
  y = []
  g.V.has('name', name).store(x)
        .out('wrote').store(y)
        .out('cite')
        .in('wrote').except(x)
        .out('wrote')
        .in('cite').except(y).dedup()
        .in('wrote').name.groupCount(m).toList()
  m = m.sort{a,b -> a.value <=> b.value}
  return m
}
```

Wymienione funkcje znajdują się w pliku `gremlin.groovy`. Same skrypty wykonywane są za pomocą interfejsu udostępnionego przez `Bulbflow` przez odpowiednie funkcje, które dalej operują na danych np.:

```
def sameReferancesAsByName( Name ):
  g = Graph()
  g.scripts.update('gremlin.groovy')
  script = g.scripts.get('sameReferancesAsByName')
  items = g.gremlin.execute(script, dict(name=Name))
  m = items.content
  if( len(m) ):
    sorted_m = sorted(m.iteritems(),
                      key=operator.itemgetter(1))
    return sorted_m
  else:
    return None
```

Do każdej wcześniej omówionej funkcji napisanej w groovy została stworzona funkcja w Pythonie, która odbiera, przetwarza i udostępnia stworzonemu UI odpowiednie dane. Wszystkie te funkcje umieszczone są w pliku `library.py`.

Interfejs użytkownika

Na potrzeby demonstracji napisano w Pythonie minimalistyczny konsolowy interfejs użytkownika. Wykorzystuje on bibliotekę `curses` do stworzenia menu, które pozwala wybrać polecenie dla bazy. Klasa menu została pobrana z `http://www.promisc.org/blog/?p=33`. Funkcje, klasy i metody odpowiedzialne za menu zamieszczono w `library.py`, `menu.py`. Odpowiedzi na zapytania są wyświetlane w konsoli. Wywołanie programu znajduje się w pliku `mainLibrary.py`.

5.5.3. Praca z programem

Uruchomienie programu demonstracyjnego odbywa się za pomocą polecenia `./mainLibrary.py` wypisanego w konsoli (po wcześniejszym nadaniu praw wykonalności dla pliku). Program wita użytkownika przejrzystym menu:

```
Wybierz opcje...
  0. Wypisz wszystkich autorow
  1. Wypisz wszystkie prace
  2. Wypisz autorow pracy
  3. Wypisz prace autora
  4. Wypisz ilosc cytowan poszczegolnych autorow
  5. Wypisz ilosc cytowan poszczegolnych prac
  6. Wypisz autorow korzystajacych z tej samej bibliografii
     co zadany autor
  7. Wypisz autorow korzystajacych z prac tych samych autorow
     co autor
  8. Zakoncz prace programu
```

Opcja nr 0 pozwala wypisać wszystkich autorów zapisanych w bazie. Po jej wybraniu oczom użytkownika ukaże się odpowiedź zależna od zawartości bazy. W tab. 5.2 zebrano odpowiedzi wygenerowane dla kolejnych opcji z menu programu dla przykładowej bazy danych. Każda opcja odpowiada za wywołanie kolejnej funkcji opisanej w podrozdziale 5.5.2.

5.6. Podsumowanie

Zagadnienia modelowanie i reprezentacji danych przedstawione w niniejszym rozdziale należy traktować jako wstęp do ogólniejszego tematu budowy i wykorzystania grafowych baz danych. Ograniczono się w nim do opisu cech grafowych baz danych, ich porównania z innymi rozwiązaniami oraz do oceny tkwiącego w nich potencjału. Rozważania te zilustrowano implementacjami prostych programów, operujących za pośrednictwem dedykowanego języka zapytań na przykładowej grafowej bazie danych.

Tab. 5.2: Wyniki otrzymane po wybraniu kolejnych pozycji menu dla przykładowej bazy danych.

0. Wypisz wszystkich autorow	1. Wypisz wszystkie prace
nazwa --- Grzegorz Marcin Tomek Bartek Andrzej Wojtek Marek Karol	nazwa --- P0 P1 P2 P3 P4 P5 P6 P7
2. Wypisz autorow pracy	3. Wypisz prace autora
Podaj nazwe publikacji P1 nazwa --- Tomek Grzegorz	Podaj nazwe autora Grzegorz nazwa --- P4 P3 P1 P0
4. Wypisz ilosc cytowan poszczegolnych autorow	5. Wypisz ilosc cytowan poszczegolnych prac
nazwa il. cytowan --- Andrzej 1 Wojtek 1 Bartek 1 Tomek 2 Marcin 2 Marek 3 Karol 3 Grzegorz 4	nazwa il. cytowan --- P3 1 P6 1 P5 1 P2 2 P1 3 P4 3 P0 4
6. Wypisz autorow korzystajacych z tej samej bibliografii co zadany autor	7. Wypisz autorow korzystajacych z prac tych samych autorow co autor
Podaj nazwe autora Grzegorz nazwa il. wspol. ref. --- Andrzej 1 Karol 1 Tomek 1 Marek 1 Grzegorz 1 Bartek 1	Podaj nazwe autora Grzegorz nazwa il. ws. autorow ref. --- Andrzej 1 Tomek 1 Grzegorz 1 Bartek 1 Karol 2 Marek 2

Przykładowa grafowa bazy danych została zbudowana w oparciu o system `Neo4j`. W rozdziale dokonano prezentacji procesu jej wykorzystania, od momentu instalacji `Neo4j`, przez napisanie programu i skryptu w języku Python, aż po uruchamianie zapytań i pracę na danych.

Jak pokazano, korzystanie z grafowych baz danych nie jest trudne i w zależności od typu składowanych danych mogą one okazać się bardzo wygodnym narzędziem. Zastosowanie grafowego modelu przechowywania danych pozwala na efektywne drążenie nawet bardzo odległych zależności stosunkowo niskim kosztem. Jeżeli więc głównym zadaniem baz danych nie ma być generowanie płytkich zestawień dużych liczby rekordów ani też przechowywanie gigantycznych zasobów słabo powiązanych ze sobą informacji, zastosowanie bazy grafowej może okazać się rozwiązaniem optymalnym. Przykładem takiego zastosowania mogą być serwisy Facebook oraz Twitter, które z dużym sukcesem wykorzystują ten rodzaj bazy danych.

Literatura

[1] A. Silberschatz, H. F. Korth, S. Sudarshan. Data models. *ACM Computing Surveyes, 28(1):105-108*, 1996.

[2] W. C. McGee. On user criteria for data model evaluation. *ACM Transactions on Database Systems (TODS), 1(4):370-387*, 1976.

[3] R. Angles, C. Gutierrez. *Survey of Graph Database Models*. Technical report number TR/DCC-2005-10, Computer Science Department, Universidad de Chile, 2005.

[4] F. Stajano. A gentle introduction to relational and object oriented databases. *Olivetti & Oracle Research Laboratory, Cambridge, UK*, 1998.

[5] E. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[6] I. Robinson, J. Webber, E. Eifrem. *Graph Databases*. O'Reily Media, 2013.

ROZDZIAŁ 6

ROBOTYCZNE ZASTOSOWANIE DOKUMENTOWEJ BAZY DANYCH

R. Kmieć, J. Zych

Niniejszy rozdział poświęcono dokumentowym bazom danych oraz towarzyszących im zagadnieniom. Zaprezentowano w nim istniejące oprogramowanie oraz omówiono przykłady robotycznych zastosowań. W rozdziale tym przedstawiono również szczegóły prostej aplikacji, zaimplementowanej z wykorzystaniem dokumentowej bazy danych.

6.1. Wprowadzenie

Nierelacyjne bazy danych określa się często miane baz NoSQL (ang. *Not only SQL databases*). Charakteryzują się one wyższym współczynnikiem dostępności oraz optymalizacją pod kątem przetwarzania i przechowywania ogromnych ilości informacji.

Dokumentowe bazy danych są bazami danych typu NoSQL. Przechowują one informacje w „dokumentach", które można porównać do rekordów w relacyjnych bazach danych. Każdy dokument posiada swój unikalny identyfikator. Choć wszystkie dokumenty mają podobną budowę, to jednak mogą przechowywać całkowicie różne i niezwiązane w żaden sposób ze sobą dane. Dane są przechowywane w tak zwanych *kolekcjach*. Podczas odpytywania dokumentowych bazy danych kolekcje są przeszukiwane najczęściej z wykorzystaniem technik MapReduce.

Pod względem popularności dokumentowe bazy danych plasują się zaraz za bazami `SQL` [1].

6.2. Przegląd najpopularniejszych aplikacji do tworzenia dokumentowych baz danych

Aktualnie na rynku dostępnych jest kilka programów obsługujących dokumentowe bazy danych. Poniżej przedstawiono przykłady najpopularniejszych.

6.2.1. CouchDB

CouchDB jest dokumentową bazą danych *Open source*, która w zamierzeniach miała w jak największym stopniu integrować się z aplikacjami sieciowymi. Używa ona standardu `JSON` do przechowywania danych, językiem zapytań jest JavaScript, który wykorzystuje technikę MapReduce w celu przyśpieszenia wyszukiwania odpowiedzi. Dostęp do bazy danych realizowany jest poprzez protokół `HTTP`. Sama baza napisana jest w języku Erlang.

W CouchDB zaimplementowano model MVCC (ang. *Multi-Version Concurrency Contro*). Dzięki niemu nie jest potrzebna blokada rekordów bazy podczas ich modyfikacji. Konflikty są rozwiązywane na poziomie aplikacji. Podczas ich rozwiązywania różne wersje dokumentów zostają najpierw połączone, aby później odrzucić informacje nieaktualne.

CouchDB opublikowano po raz pierwszy w 2005 roku. W 2008 roku projekt został przejęty przez fundację `Apache` i jest dalej przez nią rozwijany. Wykorzystywany jest przez takie firmy, takie jak: BBC, Credit Suisse, Meebo czy Ubuntu (dla usług synchronizacji). Główne cechy CouchDB to:

- schematy ACID z wykorzystaniem MVCC,
- indeksowanie ze wsparciem MapReduce,
- rozproszona architektura z replikacją,
- REST API.

6.2.2. OrientDB

`OrientDB` nie jest bazą *stricte* dokumentową - jest połączeniem bazy grafowej i dokumentowej. Dane są przechowywane w „dokumentach", ale relacje między nimi są podobne do powiązań w grafowych bazach danych. Baza ta jest w całości napisana w Javie, co zapewniło jej wieloplatformowość. `OrientDB` korzysta z nowatorskiego algorytmu indeksowania, nazwanego MVRB-Tree. Dzięki temu możliwe jest zarówno szybkie dodawanie nowych rekordów jak i szybkie przeszukiwanie. Baza jest na licencji *Apache 2*, tzn. korzystanie z niej jest darmowe. Główne cechy `OrientDB` to:

- wspiera ACID,
- jest kompatybilna ze standardem *TinkerPop Blueprints* dla grafowych baz danych,
- posiada wsparcie dla języka SQL (poza operacją `join`),
- udostępnia interfejs sieciowy,
- cechuję ją wieloplatformowość - działa na Linux'ie, Windowsie i każdej innej platformie wspierającej Javę,
- jest lekka - serwer zajmuje jedynie 1Mb i nie wymaga żadnych zależności w systemie.

6.2.3. MongoDB

MongoDB jest kolejną omawianą dokumentową bazą danych i jednocześnie najpopularniejszą z nich. Jej nazwa jest powiązana ze słowem *huMONGOus*, co w wolnym tłumaczeniu oznacza „*wielgachny*"). Historia rozwoju MongoDB jest stosunkowo krótka. W 2007 roku firma 10gen stworzyła platformę, a w 2009 udostępniono kod źródłowy MongoDB na licencji *AGPL* (jako osobnego produktu). Ostatnie stabilne wydanie nosi numer wersji 2.4.3 (stan w maju 2013). Z MongoDB korzystają takie firmy, jak: eBay, MetLife, Telefónica i inne.

MongoDB przechowuje dane w dokumentach o strukturze podobnej do JSON, wraz z dynamicznymi schematami. Taki rodzaj struktur często nazywa się BSON. MongoDB jest napisane w języku C++. Główne cechy MongoDB to:

- kolejki Ad hoc,
- indeksowanie,
- replikacja,
- balansowanie obciążenia,
- przechowywanie plików,
- agregacja z wykorzystaniem MapReduce,
- zapytania po stronie serwera w JavaScript,
- kolekcje o stałym wymiarze.

6.2.4. RavenDB

Kolejną popularną dokumentową bazą danych jest RavenDB stworzona przez hibernating rhinos. Korzystają z niej między innymi takie firmy, jak: msnbc.com, Beatman Ltd, Cornvalius czy JetBrains.

Baza RavenDB napisana jest w języku .NET, a jej najbardziej wyróżniającą cechą jest możliwość wbudowania klienta w dowolną aplikacje .NET-ową. Do przechowywania danych wykorzystuje standard JSON, a zapytania można tworzyć w technologiach: .NET, REST, JavaScript lub Silverlight. Baza RavenDB dostępna jest na licencji *AGPL*. Główne cechy RavenDB to:

- model ACID,
- tworzenie zapytań w wielu językach,
- indeksowanie z wsparciem MapReduce,
- kontrola wersji,
- możliwość wbudowania w dowolna aplikacje .NET.

6.3. Praca z dokumentową bazą danych

Choć dokumentowe i relacyjne bazy danych różnią się w wielu elementach, to jednak można znaleźć pewne analogie między nimi. Ich krótką listę przedstawiono w tab. 6.1 [2]. W tab. 6.2 zaś przedstawiono porównanie podstawowych poleceń dokumentowych i relacyjnych baz danych.

Główną różnica polega na zmianie kolejności czynności związanych z tworzeniem tabel/obiektów i zasilaniem bazy danych. Dla MongoDB najpierw obiekt jest

Tab. 6.1: Analogie między bazami typu SQL a dokumentowymi.

Bazy typu SQL	Dokumentowe bazy
Rekord	Dokument
Tablica	Kolekcja
Klucz	Identyfikator
Relacja 1:N	Zagnieżdżenie
Relacja M:N	Tablica referencji do obiektów
Indeks	Indeks

Tab. 6.2: Zestawienie komend `MySQL` i `MongoDB`

Polecenie	`MySQL`	`MongoDB`
Uruchomienie serwera i klienta	`mysqld , mysql`	`mongod , mongo`
Wyświetlenie aktualnej bazy	`select database();`	`db`
Wyświetlenie dostępnych baz	`show databases;`	`show dbs`
Wybór bazy	`use baza;`	`use baza`
Utworzenie tabeli/dokumentu	`create table tab (c varchar(20), i int);`	`j={c:"tekst", i=3}`
Dodanie rekordu do tabeli lub bazy	`insert into table tab values ("tekst",3);`	`baza.katalog.insert(j)`
Wyświetlenie tabel/kolekcji	`show tables;`	`show collections`
Wyświetlenie zawartości tabeli/kolekcji	`select * from tab;`	`baza.katalog.find()`
Połączenie z poziomu `PHP`	`mysql_connect($host, $username,$password);`	`$m = new MongoClient();`
Wybór bazy z poziomu `PHP`	`mysql_select_db('baza');`	`$db = m->baza;`
Wybór kolekcji z poziomu `PHP`	N/D	`$coll = db->kolekcja;`

tworzony, a dopiero potem jest dodawany do bazy. Natomiast w przypadku bazy relacyjnej najpierw jest tworzona tabela o zadanych polach (schemat), a dopiero potem dodawane są do niej obiekty.

W przypadku relacyjnych baz danych należy więc na sztywno zdefiniować strukturę danych, które będą zawarte w tabeli. Aby dodać obiekt innego typu, należy utworzyć zupełnie nową tabelę. Rozwiązanie to sprawdza się bardzo dobrze, gdy elementy tabeli są ze sobą spójne. Jeżeli występują zmienne typy danych, rozwiązania stosowane w relacyjnych bazach przestają się sprawdzać na szeroką skalę. W takim przypadku, podejście dokumentowych baz danych sprawdza się

znacznie lepiej. W bazach dokumentowych można tworzyć kolekcje pewnych danych, a następnie dodawać do niej dowolnie złożone obiekty.

Odwoływanie się do poszczególnych obiektów w dokumentowej bazie MongoDB bardziej przypomina używanie pól i metod w klasach języka C++, co będzie można zaobserwować w przykładowych wdrożeniach.

6.4. Przykład zastosowania w robotyce

Według autorów artykułu [3] bazy danych używane w robotyce powinny charakteryzować się:

- zdolnością zbierania wszystkich danych przesyłanych od robota do bazy w czasie rzeczywistym,
- szybką odpowiedzią na zapytania do bazy,
- kompatybilnością z typowymi interfejsami robotycznymi,
- brakiem lub minimalna konfiguracją,
- łatwym przystosowaniem bazy do ciągle rozwijanych struktur danych,
- jednoczesnym dostępem dla wielu robotów i platform zewnętrznych,
- niezależnością między platformą robotyczną a kontekstem programowym.

Według autorów niniejszego rozdziału bazą, która spełnia te wszystkie warunki, jest MongoDB. Baza ta, wraz z oprogramowaniem ROS i Fawkes, została zaimplementowana w robocie HERB 2.0 (zobacz rys. 6.4). Platforma ta posiada kamerę RGBD, kamerę monochromatyczną, obrotowy dalmierz laserowy oraz dwa ramiona o 7-miu stopniach swobody. HERB 2.0 w zamierzeniach swoich twórców ma być domowym asystentem robotycznym człowieka.

Rys. 6.1: Robot HERB 2.0 stworzony w laboratorium Personal Robotics Lab na uniwersytecie CMU (ang. *Carnegie Mellon University*) [3].

W czasie pracy robota dokumentowa baza danych jest zasilana danymi zawierającymi obraz, wraz z jego kolorem i głębią, oraz stowarzyszonymi z nimi transformacjami odniesionymi do globalnego układu współrzędnych. Przykładowy dokument z danymi wygląda następująco:

```
{
  _id: "4f55e28ffa24ebb2e469d331",
  __recorded: "2011-11-11T17:17:17Z .2342",
  frame: "/rx28/tilt",
  child_frame: "/kinect/image",
  translation: { x: 0.1, y: 0.2, z: 0.3 },
  rotation: { x: 0.0, y: 0.0, z: 0.0, w: 1.0 }
}
```

Dane te są wykorzystywane do wyszukiwania błędów w pracy robota oraz oceny wydajności jego akcji.

6.5. Wdrożenie dokumentowej bazy danych na platformę mobilną

Aby pokazać jakie możliwości daje zastosowanie dokumentowej bazy danych w robotyce przeprowadzono prosty eksperyment. Polegał on na wygenerowaniu pewnej liczby dokumentów reprezentujących symulowane odczyty z czujników robota. Po wstawieniu do bazy danych dokumenty te zostały przetworzono odpowiednio sformułowanym zapytaniem. Pozwoliło to uzyskać informacje o historii działań robota. Szczegóły tego eksperymentu oraz sposób jego implementacji na platformie `MongoDB` opisano poniżej. W opisie tym przyjęto następującą konwencję:

- linie rozpoczynające się od $ oznaczają polecenia wydawane w konsoli systemowej.
- linie rozpoczynające się od > oznaczają polecenia wpisywane do klienta `MongoDB`.

6.5.1. Uruchomienie bazy danych i połączenie z nią

Uruchomienie dokumentowej bazy danych nastąpiło na jednym z serwerów *Zakładu Podstaw Cybernetyki i Robotyki* (znajdującym się pod adresem *panamint.ict.pwr.wroc.pl* i działającym pod kontrolą systemu operacyjnego `Debian`). Cała procedura rozpoczęła się od pobrania plików binarnych ze strony domowej projektu MongoDB (`www.mongodb.org`) w odpowiedniej dla sprzętu wersji (w tym przypadku były to pliki binarne skompilowane dla architektury `i686`). Po tym nastąpiło uruchomienie dokumentowej bazy danych (zapewniło to wykonanie poniższych instrukcji).

```
$ wget http://fastdl.mongodb.org/linux/
mongodb-linux-i686-2.4.3.tgz
$ tar xzvf http://fastdl.mongodb.org/linux/
```

```
mongodb-linux-i686-2.4.3.tgz
$ cd mongodb-linux- i686-2.4.3/bin
$ ./mongod --dbpath ..
```

Po uruchomieniu serwera bazy danych został uruchomiony klient (program do komunikacji z bazą danych, do pobrania ze strony domowej projektu MongoDB lub z repozytoriów dystrybucji):

```
$ mongo panamint.ict.pwr.wroc.pl
```

Uwaga: Baza danych uruchomiona bezpośrednio po pobraniu zezwala na dowolne połączenia, co łamie pewne zasady bezpieczeństwa. Aby ją zabezpieczyć należy dokonać odpowiedniej konfiguracji.

6.5.2. Generowanie danych

Po połączeniu z bazą można było rozpocząć pracę nad symulacją odczytów dwóch czujników. Wykorzystano do tego dodatkową funkcję, generującą liczby losowe o rozkładzie normalnym. Zgodnie z specyfikacją bazy `Mongo`, w bazie tej można używać poleceń i składni `JSON`. Stąd funkcję tę można było zaimplementować jako:

```
nrand = function() {
  var x1, x2, rad;
  do {
    x1 = 2 * Math.random() - 1;
    x2 = 2 * Math.random() - 1;
    rad = x1 * x1 + x2 * x2;
  } while(rad > 1 || rad ==  0);
  var c = Math.sqrt(-2*Math.log(rad) / rad);
  return x1 * c;
};
```

Należy jednak pamiętać, że wszystko, co znajduje się w bazie, musi być dokumentem. Dlatego tworząc własną metodę posłużono się następującym poleceniem:

```
  > db.system.js.save(
  {_id:'nrand', value:function()
    {
      var x1, x2, rad;
        do {
          x1 = 2 * Math.random() - 1;
          x2 = 2 * Math.random() - 1;
          rad = x1 * x1 + x2 * x2;
        } while(rad > 1 || rad ==  0);
      var c = Math.sqrt(-2 * Math.log(rad) / rad);
      return x1 * c;
    }
  });
```

Polecenie to dodaje do listy metod w kolekcji `db.system.js` funkcję `nrand()`. Aby ją uruchomić należy wywołać komendę:

```
> db.eval('nrand()');
```

Jak widać, odpowiednio przekształcone metody `JSON` można przenosić do `MongoDB`.

Skoro zdefiniowano już funkcję, która miała posłużyć do generowania szumu, można było ją wykorzystać do zasymulowania zaszumionego sygnału. W tym celu posłużono się pętlą `for`.

```
> for (var i = 1; i <= 2000; i++){
  db.sensors.insert({
    type: 'SharpR',
    value: 134 + db.eval('nrand()') * 10,
    TimeStamp: new Date().getTime()
  });
  db.sensors.insert({
    type:'SharpL',
    value: 120 + db.eval('nrand()') * 19,
    TimeStamp: new Date().getTime()
  });
}
```

Wykonanie powyższego kodu skutkuje wygenerowaniem 4000 symulowanych odczytów z dalmierzy. Prawy czujnik ma wartość oczekiwaną równą 134 oraz stosunkowo mały szum. Lewy czujnik ma wartość oczekiwaną 120 oraz większy szum od poprzedniego. Dodatkowo każdy z odczytów oznaczono czasem jego wystąpienia. W tym przypadku stanowił on liczbę milisekund, jaka minęła od 1 stycznia 1970 roku na serwerze.

6.5.3. Odpytywanie bazy

Obliczenie prostych wartości na podstawie danych można dokonać za pomocą nieskomplikowanych metod, np. za pomocą metody `db.sensors.aggregate()` wyliczającej średnią z obu czujników:

```
> db.sensors.aggregate( {
  $group : {
    _id : {
      type: '$type',
      time:'$TimeStamp'
    },
    val : {
      $sum: '$value'
    },
  }
}, {
  $group: {
    _id : '$_id.type',
```

```
    total: { $avg : '$val'}
  }
})
```

Wartości te można także obliczyć pomijając jedno grupowanie w następujący sposób:

```
> db.sensors.aggregate({$group:
{ _id: '$type', val: {$avg: '$value' },}})
```

W obu przypadkach wynik jest ten sam i przedstawia się jak niżej:

```
{
  "result" : [
    {
      "_id" : "SharpR",
      "total" : 133.83784978905038
    },
    {
      "_id" : "SharpL",
      "total" : 119.57165509074615
    }
  ],
  "ok" : 1
}
```

Jak widać otrzymana wartość średnia okazała się bardzo zbliżona do wartości rzeczywistej. Zmieniając słowo *„avg”* na *„max”* lub *„min”*, można uzyskać odpowiednio największą i najmniejszą wartość odczytaną z czujników:

```
> db.sensors.aggregate({$group: { _id: '$type',
val: {$min: '$value'},}})
{
  "result" : [
    {
      "_id" : "SharpL",
      "val" : 45.880575696370215
    },
    {
      "_id" : "SharpR",
      "val" : 102.29447234636108
    }
  ],
  "ok" : 1
}

> db.sensors.aggregate({$group: { _id: '$type',
 val: {$max: '$value'},}})
{
  "result" : [
```

```
    {
      "_id" : "SharpL",
      "val" : 196.6290358346244
    },
    {
      "_id" : "SharpR",
      "val" : 163.61557321608615
    }
  ],
  "ok" : 1
}
```

6.6. Podsumowanie

Podsumowując rozważania nad dokumentowymi bazami danych można powiedzieć, że nadają się one do rozwiązywania problemów, w których dane nie mają jakiejś z góry narzuconej struktury, gdy ich format oraz kolejność występowania w bazie nie mają znaczenia. Ponadto sprawdzają się one doskonale podczas przetwarzania ogromnych ilości danych, idących w setki tysięcy czy miliony rekordów. MongoDB jest dodatkowo bardzo wydajne przy przechowywaniu dużych plików binarnych, np. zdjęć czy filmów.

Środowiska robotyczne zazwyczaj charakteryzują się silną strukturyzacją, o jasno określonych postaciach wyjścia i wejścia do/z kolejnych obiektów. Sprawia to, że praca w nich wydaje się łatwiejsza i sprawniejsza przy wykorzystaniu relacyjnych baz danych. Prędkość odpowiedzi na zapytania bazy dokumentowej nie wydaje się już tak atrakcyjna gdy zestawimy ja z czasem zapisu danych do bazy, który jest przynajmniej o rząd większy. W robotyce bardzo istotny jest czas reakcji, dlatego każdy wpis musi być otagowany czasem wystąpienia zdarzenia z dokładnością do milisekund.

Dokumentowe bazy danych są atrakcyjne w zastosowaniach, w których istnieje potrzeba przechowywania i przetwarzania ogromnych ilości danych o strukturze początkowo nieznanej lub zmieniającej się z czasem. Niestety, nie spełnia ona wszystkich wymagań jakie potrzebują zastosowania robotyczne (długi czas zapisu do bazy), a niektóre z jej zalet nie są aż tak bardzo przydatne (jak elastyczność struktury danych).

Literatura

[1] Db-engines ranking - popularity ranking of database management systems. http://db-engines.com/en/ranking.

[2] A. Boicea. MongoDB vs Oracle – Database Comparison. *Emerging Intelligent Data and Web Technologies (EIDWT)*, 2012.

[3] T. Niemueller. A Generic Robot Database and its Applications in Fault Analysis and Performance Evaluation. 2012.

ROZDZIAŁ 7

SYSTEMY DECYZYJNE

M. Kret, P. Ptasznik

W niniejszym rozdziale przedstawiono podstawy teorii systemów decyzyjnych oraz przedstawiono przykład ich praktycznego zastosowania. Na wstępie zdefiniowano kluczowe pojęcia, a następnie przedstawiono różnego rodzaju systemy ekspertowe charakteryzujące się różną strategię zarządzania zgromadzoną wiedzą. Szczególną uwagę zwrócono na sposób reprezentacji danych, metody ich przetwarzania i interpretacji oraz problemy doboru najlepszej strategii wnioskowania.

Praktyczne zastosowanie opisanych metod zilustrowano przykładowym projektem systemu wspomagającego działanie autonomicznego układu sterowania samochodem. Jego funkcjonowanie polega na generowaniu parametrów dla algorytmu sterującego samochodem przy ciągle uzupełnianej wiedzy o otoczeniu, a dokładniej: zalecanej prędkości, możliwości skrętu lub jazdy na wprost, reguł pierwszeństwa, zalecanego rodzaju sterowania oraz sugestii co do użycia świateł.

7.1. Rozwój systemów decyzyjnych

System decyzyjny to program, maszyna lub ich kombinacja, która pozwala na rozwiązanie jakiegoś problemu decyzyjnego na bazie dostępnej wiedzy z zastosowaniem logicznych mechanizmów dedukcyjnych, indukcyjnych lub innych form rozumowania [1]. Systemy te odgrywają ważną rolę w rozwoju szeroko rozumianej sztucznej inteligencji.

Teoria systemów decyzyjnych to stosunkowo młoda dziedzina badań. Rozwija się nieco od ponad pół wieku, a listę najważniejszych wydarzeń z nią związanych można uszeregować jak w poniższym zestawieniu:

- **1847** - George Boole przedstawił matematyczny zapis logiki nazywany dzisiaj algebrą Boole'a;
- **1965-1975** - Edward Feigenbaum i Robert K. Lindsay zbudowali pierwszy ekspertowy system decyzyjny DENDRAL (jego zadaniem było podanie składu molekularnego związku chemicznego na podstawie danych ze spektrometru);

- **1966** - powstał program Eliza, przez który niektórzy badani uwierzyli, że maszyna przeszła test Turinga;
- **1970** - zademonstrowano pierwsze praktyczne zastosowania logiki rozmytej do sterowania procesem;
- **1971-1972** - Alain Colmerauer i Phillipe Roussel stworzyli język Prolog;
- **Późne lata 70** - powstał pierwszy komercyjny system ekspertowy XCON;
- **1980** - w systemie do obsługi japońskiego metra zaimplementowano logikę rozmytą;
- **1997** - komputer IBM Deep Blue pokonał w szachach arcymistrza Garija Kasparowa.

Pierwszy ekspertowy system decyzyjny Dendral zbudowano z wykorzystaniem języka Interlisp - dialektu języka Lisp. Dał on podstawy do stworzenia wielu innych systemów w późniejszych latach (w ten sposób powstał m.in. język OPS5, służący do opisu reguł produkcyjnych). W systemie Dendral w celu określenia składu molekularnego badanej cząstki wykorzystano algorytm heurystyczny, który przeszukiwał bazę wiedzy badając zapisane w niej możliwości. Sprawdzał się on doskonale jako asystent eksperta z dziedziny chemii.

Inny znany system XCON (komercyjny, wcześniej nazywany R1) został z kolei stworzony w języku OPS5 i bazował na około 2500 regułach produkcyjnych. Służył on do konfigurowania (dobierania odpowiednioch komponentów) komputerów VAX zgodnie z potrzebami klientów. Dzięki niemu zrealizowano około 80000 zamówień, osiągając dokładność rzędu 95%–98% i oszczędzając miliony dolarów na niepotrzebnych częściach i czasie pracy.

Oparty na bazie wiedzy system ekspertowy KBES (ang. *knowledge based expert system*) to wynik badań nad sztuczną inteligencją, dostarczony w formie oprogramowania. Nadawał się on do rozwiązywania problemów polegających na klasyfikacji lub diagnostyce [2]. Architekturę KBES, metody reprezentacji danych, strategie rozwiązywania problemów i obszary zastosowań tego typu aplikacji omówiono w kolejnych podrozdziałach.

7.2. Systemy ekspertowe

7.2.1. Wiedza w systemach ekspertowych

Część programu odpowiedzialna za reprezentację wiedzy (z określonej dziedziny) jest nazywana *bazą wiedzy*. Przeważnie jest ona odseparowana od modułu odpowiedzialnego za wnioskowanie. Pozwala to na elastyczne przeprogramowanie jednej części programu bez naruszania drugiej. Można np. dodać informacje do bazy wiedzy bez ingerencji w silnik wnioskujący lub zmienić algorytm podejmowania decyzji bez wprowadzania zmian w bazie wiedzy. Aby ułatwić przetwarzanie informacji często próbuje się dokonywać uproszczeń w wykorzystywanych modelach. Z podobnych pobudek omija się sprowadzanie różnych rodzajów wiedzy do jednego formalizmu.

Akwizycja wiedzy

Akwizycję wiedzy można zdefiniować jako „transfer i transformację wiedzy eksperckiej, potencjalnie rozwiązującej dany problem, z pewnego źródła do programu". Pojęcie to występuje w obszarze sztucznej inteligencji określanym mianem *uczenia maszynowego*. Od sposobu pozyskiwania wiedzy zależy często funkcjonowanie algorytmów uczących się, powiązanych z systemami ekspertowymi. Nowa wiedza może być:

- wprowadzona ręcznie przez programistę,
- uzyskana poprzez manipulację aktualnie posiadanymi danymi,
- uzyskana ze zbioru aktualnych informacji metodami indukcyjnymi,
- uzyskana poprzez empiryczne odkrywanie nowych reguł.

Dla przykładu: informacja do programu Emycin wprowadzana jest tylko przez człowieka, a Teiresias dodatkowo sprawdza, czy nowe reguły nie wprowadzają redundancji i sprzeczności. Meta-Dendral generuje i modyfikuje swoją wiedzę współpracując z człowiekiem przy ocenianiu i testowaniu nowych informacji, podczas gdy programy takie jak AM i Eurisko próbują wyszukiwać nowe koncepcje i powiązania.

Reprezentacja wiedzy

Problem reprezentacji wiedzy w systemach decyzyjnych próbuje się rozwiązać naśladując pracę ludzkich umysłów. Do najczęściej stosowanych metod reprezentacji wiedzy należą:

- podejście regułowe (korzystające z reguł produkcyjnych: jeśli wystąpiło TO, zrób TAMTO);
- podejście oparte na grafach uogólnionych;
- logika predykatów.

Wymienione metody stosuje się zamiennie lub wszystkie na raz, w zależności od natury problemu wymagającego rozwiązania. Korzystają z nich systemy wnioskowania, które czasem są nazywane systemami wnioskującymi w oparciu o wzorce (ang. *pattern directed inference system*). W każdym takim systemie za niezwykle ważne uważa się:

- zespół modułów aktywujących się podczas dopasowaniu danych do charakterystycznego dla nich wzorca;
- dynamiczne struktury danych, które mogą być odczytywane i modyfikowane w trakcie działania programu;
- interpreter zarządzający selekcją i aktywacją odpowiednich modułów.

Sposób, w jaki te elementy współgrają ze sobą zostanie wyjaśniony przy okazji przedstawienia różnych form reprezentacji informacji.

Zastosowanie wiedzy

Znajomość dostępnej wiedzy oraz sposobów jej zastosowania nazywa się często *meta-wiedzą* tzn. wiedzą o wiedzy. Jest to kluczowy aspekt podczas pro-

jektowania algorytmów decyzyjnych, ponieważ złożoność procesu dostępu do bazy wiedzy wpływa znacząco na szybkość podejmowania decyzji. W dalszych częściach rozdziału zostaną przedstawione różne sposoby rozwiązania tego problemu.

7.2.2. Backtracking

Ciekawą własnością systemów ekspertowych, która wyraźnie odróżnia je od klasycznych algorytmów wyszukiwania, jest możliwość zaobserwowania wygenerowanej przez nie ścieżki prowadzącej do rozwiązania. Inaczej mówiąc, system ekspertowy powinien udostępniać informacje na temat obszarów bazy wiedzy, które uaktywniały się podczas generowania rozwiązania. Pozwala to programistom na wgląd w sposób działania systemów, a co za tym idzie, na ich ulepszanie i usuwanie zauważonych błędów lub nienaturalnych zachowań.

7.3. Systemy produkcyjne

Istnieje wiele sposobów projektowania systemów decyzyjnych. W niniejszym podrozdziale główną uwagę skierowano na funkcjonowanie systemów produkcyjnych oraz różne formalizmy reprezentacji wiedzy i techniki pozwalające osiągnąć zadowalające wyniki przy wspieraniu człowieka (lub autonomicznego układu sterowania) w podejmowaniu decyzji. Omówiono je dość pobieżnie, jednak bardziej szczegółowe informacje można znaleźć w [3].

7.3.1. Reguły produkcyjne

Zanim reguły produkcyjne zostały zaadoptowane dla systemów ekspertowych, były one wykorzystywane w teorii automatów, gramatyce formalnej i przy projektowaniu języków programowania. W kontekście systemów ekspertowych są one czasami nazywane regułami akcji i reakcji lub sytuacji i reakcji. Każdy system produkcyjny powinien składać się ze zbioru reguł (nazywanymi często *pamięcią produkcyjną*), interpretera i *pamięci operacyjnej*, w której przetrzymywane są pośrednie wyniki obliczeń i pośrednie cele. Pamięć operacyjna pełni też poniekąd rolę bazy wiedzy.

7.3.2. Składnia

Reguły składają się z warunków i akcji (konkluzji):

$$\begin{array}{ll} if & P_1 \& P_2 \ldots \& P_n \\ then & Q_1 \& Q_2 \ldots \& Q_m \end{array}$$

Warunki są zwykle trójkami w formie (obiekt-atrybut-wartość), np. (Paweł-wiek-24), co oznacza: wiek Pawła wynosi 24. W oparciu o ten warunek można skonstruować regułę:

if	(Paweł wiek 24)
then	(Paweł kończ studia)

oznaczającą, że jeżeli Paweł ma 24 lata, to ma skończyć studia. Zakłada się oczywiście możliwość wprowadzania bardziej ogólnych reguł korzystając ze zmiennych (oznaczanych tutaj gwiazdką):

```
if    (*człowiek wiek *x) & (*człowiek praca NULL) &
      (*x większe-od 24) & (*x mniejsze-od 67)          (7.1)
then  (*człowiek bierz zasiłek)
```

Podczas interpretacji takich reguł poszczególnym zmiennym będą przypisywane te same wartości w każdej z reguł (np. we wszystkich trzech miejscach wystąpienia zmiennej *x* zostanie wpisana ta sama wartość, podobnie stanie się ze zmienną *człowiek*).

7.3.3. Pamięć operacyjna

Zadaniem pamięci operacyjnej jest przechowywanie trójek (obiekt-atrybut-wartość). Dane te są podstawą dla interpretera, który uaktywnia odpowiednie reguły na zasadzie dopasowania do wzorców. To znaczy, że jeżeli w danym warunku nie występują żadne zmienne, to jest on spełniony jedynie wtedy, gdy w pamięci operacyjnej znajdzie się identyczne wyrażenie. Jeżeli natomiast warunek zawiera zmienne (pełni rolę *wzorca*), to jest on spełniony tylko wtedy, gdy w pamięci operacyjnej istnieje wyrażenie, które „pasuje" do wzorca w taki sam sposób, jak reszta *pod-warunków* w rozpatrywanej regule.

Dla przykładu można sprawdzić, czy reguła (7.1) aktywuje się, jeżeli w pamięci operacyjnej (ang. *working memory* – WM) znajdą się następujące dane:

```
WM  =  ((Paweł wiek 30) (Paweł praca NULL))
```

Pierwsza trójka warunku (*człowiek wiek *x) pasuje do trójki w pamięci operacyjnej (Paweł wiek 30) przy podstawieniu *człowiek = Paweł, *x = 30. Próby dopasowania kolejnych trójek warunku do trójek z bazy wiedzy będą teraz ograniczone tym podstawieniem, a więc zamiast (*człowiek praca NULL) będzie (Paweł praca NULL) itd. W tym konkretnym przypadku widać, że wszystkie trójki warunku znajdują odpowiednie dopasowania w bazie wiedzy, a więc konkluzja (Paweł bierz zasiłek) zostanie zastosowana.

7.3.4. Rozwiązywanie konfliktów

Zadaniem interpretera jest:

- znalezienie dopasowania między elementami bazy wiedzy, a warunkami stosowania reguł;
- podjęcie decyzji, która reguła ma być aktywowana, gdy spełnione są warunki dla więcej niż jednej z nich;
- zastosowanie reguły i ewentualnie modyfikacja pamięci.

Sytuacja, w której można zastosować więcej niż jedną regułę jest częsta i wymaga odpowiedniego mechanizmu rozwiązywania konfliktów. Oczywiście można za-

programować system w ten sposób, aby w danej sytuacji możliwe było zastosowanie tylko jednej reguły (taki system nazywamy deterministycznym), jednak w przypadku dużych systemów staje się to bardzo trudne. Wybrana strategia rozwiązywania konfliktów zależy zawsze ściśle od założonego celu, niemniej najczęściej stosuje się kombinacje trzech niżej wymienionych mechanizmów.

Strategia odporna (ang. *refractoriness*) - reguła nie powinna być stosowana więcej niż raz dla tego samego zestawu danych. Oczywistą metodą implementacji tej strategii jest kasowanie instancji, które były już stosowane. Słabszą wersją tej metody jest kasowanie instancji, które były stosowane w poprzedniej iteracji. Strategia ta stosowana jest zwłaszcza w sytuacji, gdy chce się uniknąć zapętlenia programu.

Strategia świeżych danych (ang. *recency*) - elementy pamięci operacyjnej posiadają zwykle stemple czasowe, aby można było rozpoznać, kiedy zostały dodane. Instancje, które biorą udział w aktualnym dopasowaniu sortowane są według czasu ich dodania do pamięci operacyjnej i w razie konfliktu odpalana jest reguła, dla której stosowane są dane świeższe. Strategia taka jest bardzo intuicyjna, gdyż nowe dane niosą zwykle ze sobą nowe szanse na rozwiązanie problemu.

Strategia reguł specyficznych (ang. *specificity*) - instancje otrzymane z reguł o bardziej złożonych warunkach stosowane są chętniej od instancji otrzymanych z dopasowania do reguł bardziej ogólnych. Mechanizm ten pozwala np. na rozwiązanie takiego konfliktu:

[reg. 1]	if	(*X jest Ptakiem)
	$then$	(*X umie Latać)
[reg. 2]	if	(*X jest Ptakiem) & (*X jest Emu)
	$then$	(*X nie umie Latać)

Zastosowana zostanie reguła 2, ponieważ jej warunek wymaga wzięcia pod uwagę większej liczby danych.

7.3.5. Rozumowanie w przód i rozumowanie w tył

Zagadnienie odpowiedniego wysterowania systemu produkcyjnego niesie ze sobą wiele nietrywialnych problemów. W podrozdziale 7.3.4 przedstawiono sposoby lokalnego wpływu na zachowanie programu. Istnieją też rozwiązania globalne. Techniki globalne zwykle nie zależą od natury rozwiązywanego problemu i są zakodowane na stałe w interpreterze, podczas gdy wybór strategii lokalnych zależy od poszczególnej aplikacji i programista przeważnie może na nie mocno wpływać.

Przy sterowaniu systemem produkcyjnym na poziomie globalnym reguły produkcyjne mogą być rozwijane w przód lub w tył. Oznacza to, że na podstawie posiadanej wiedzy można szukać jakiejś konkluzji poprzez dopasowywanie warunków do elementów pamięci operacyjnej (rozumowanie w przód). Można także

postawić pewną hipotezę i sprawdzić, czy jest prawdziwa. W tym wypadku decyzje są dopasowywane do prawych stron reguł produkcyjnych, aby na końcu dało się stwierdzić, czy istnieją dane pozwalające na wygenerowanie danego ciągu akcji (rozumowanie w tył). Przykładem systemu rozumującego w przód jest R1 - program który konfiguruje komputery VAX. Przykładem systemu rozumującego w tył jest MYCIN - program który diagnozuje infekcje krwi.

7.3.6. Meta-reguły

Ciało każdej reguły produkcyjnej powinno być skonstruowane w ten sposób, aby nie wywoływała ona bezpośrednio żadnej innej reguły inaczej niż poprzez wprowadzenie odpowiednich zmian do pamięci operacyjnej. Bywają jednak sytuacje, w których programista chciałby odpowiednio ukierunkować proces rozwiązywania danego problemu bez konieczności ingerencji w bazę wiedzy. Narzędziem, które mu na to pozwala, są meta-reguły. Od zwykłych reguł różnią się tym, że zamiast rozwiązywać dany problem decyzyjny faworyzują one pewne reguły produkcyjne, zmuszając system do rozumowania w określony sposób. Przykładem meta-reguł są strategie rozwiązywania konfliktów opisane w podrozdziale 7.3.4.

7.4. Reprezentacja wiedzy

W przedstawionym systemie wykorzystano reguły produkcyjne, które z kolei bazowały na prymitywnych typach danych. Nie zawsze jest to rozwiązanie optymalne. Można zauważyć, że wnioskowanie można przyspieszyć poprzez zdefiniowanie odpowiedniej struktury danych. Termin *obiekty ustrukturyzowane* odnosi się do dowolnej reprezentacji wiedzy, której bloki składowe są analogiczne do łuków i wierzchołków grafu. Obiekty ustrukturyzowane pełnią bardzo ważną rolę w grupowaniu informacji. Poniżej omówiono sposób ich wykorzystania w systemach decyzyjnych.

7.4.1. Teoria grafów

Teoria grafów znalazła szerokie pole zastosowań w opisie abstrakcyjnych typów danych stosowanych w implementacjach algorytmów sztucznej inteligencji. Podstawą tej teorii jest założenie o istnieniu prostych obiektów nazywanych wierzchołkami i łukami. Łuki łączą wierzchołki w pary i mogą przyjmować pewne wartości, często utożsamiane z kosztem przejścia z jednego wierzchołka na drugi. Jeżeli zbiór wierzchołków oznaczymy N, to każdy podzbiór NxN jest grafem. Jeżeli kolejność w parach NxN ma znaczenie, to graf jest skierowany. Jeżeli graf nie zawiera pętli, ani krawędzi wielokrotnych, to nazywany jest grafem prostym. Jeżeli graf dodatkowo nie zawiera żadnych cykli, to jest lasem. Jeżeli G jest prostym grafem o n wierzchołkach i $n-1$ krawędziach bez cykli, to G jest drzewem. Takie drzewo G nazywa się grafem spójnym, jeżeli dla każdego wierzchołka istnieje droga do każdego innego wierzchołka.

W teorii grafów sieć jest skierowanym grafem prostym, zawierającym pewne wartości numeryczne skojarzone z krawędziami, które utożsamia się z kosztem przejścia lub odległością. W kontekście sztucznej inteligencji graf może być czymkolwiek, to znaczy może reprezentować dowolnie przyjęte relacje między wierzchołkami. Daje to możliwość modelowania złożonych abstrakcyjnych typów danych. Współcześnie istnieje wiele różnego rodzaju algorytmów działających na grafach, dzięki czemu wiedza przechowywana w takiej formie może być efektywnie przetwarzana.

7.4.2. Sieci asocjacyjne

Wiedzę zaczęto reprezentować za pomocą sieci w końcu lat sześćdziesiątych ubiegłego wieku przy okazji prac nad semantyką języków naturalnych. Postulowano, że kluczem do zrozumienia danego języka może być pewien zbiór prostych reguł. Zaowocowało to powstaniem różnych koncepcji co do sposobu przechowywania znaczenia zdań w pamięci komputera, umożliwiającego ich przetwarzanie i wykorzystanie na wzór tego, jak to czyni człowiek. Celem było zapisanie danych w takiej formie, aby nie zajmowały one zbyt wiele miejsca w pamięci oraz były zrozumiałe zarówno dla maszyny, jak i dla człowieka (jego osiągnięcie do dzisiaj okazuje się sporym problemem).

Szczególnym przypadkiem grafowego modelu danych jest sieć semantyczna (ang. *semantic net*), choć może nazwa *sieć asocjacyjna* lepiej oddaje jego istotę. W sieci asocjacyjnej każdy wierzchołek określany jest jako *koncept*, czyli pewien abstrakcyjny typ, a każda krawędź reprezentuje związek między odpowiednimi konceptami. Jest to twór podobny do słownika, w którym każdy termin zdefiniowany jest za pomocą innych, aż dochodzi się do terminów uznawanych za niewymagające dalszych definicji. Innymi słowy, budowana jest pewna hierarchia, w której koncepty leżące niżej definiowane są za pomocą konceptów leżących bliżej szczytu.

Dla przykładu *maszynę* można zdefiniować jako złożenie pewnych komponentów, umożliwiające przekształcenie siły w pewną pracę. Wymaga to dodania do grafu takich konceptów jak *komponent*, *złożenie* itd. oraz zamodelowania łączących je związków. Ponadto do tworzonego grafu można dodać takie koncepty, jak *samochód* i *lodówka*, uzupełniając graf związkami dzięki którym będzie wiadomo, że samochód jest typem maszyny, tak samo jak lodówka. Ciekawą własnością, nazywaną czasami gospodarką poznawczą (ang. *cognitive economy*), jest fakt, że skoro samochód zdefiniowany jest jako typ maszyny, a maszyna zdefiniowana jest jako złożenie pewnych komponentów, to samochód również jest takim złożeniem [4]. Zastosowanie takiego mechanizmu wyeliminowało konieczność uzupełniania grafu krawędziami reprezentującymi powtarzające się związki dla każde konceptu w hierarchii. Pozwoliło to na znaczące obniżenie zapotrzebowania na pamięć, zwłaszcza w dużych bazach danych, kosztem zwiększenia złożoności obliczeniowej impementowanych algorytmów. Konwencja ta dzisiaj jest znana jako *dziedziczenie* i jest szeroko stosowana w różnych dziedzinach (np. w sztucznej inteligencji, programowaniu i modelowaniu).

7.4.3. Przeszukiwanie intersekcyjne

Podstawową procedurą pozwalającą stwierdzić, czy dany koncept jest w odpowiedniej relacji z innym konceptem, jest przeszukiwanie intersekcyjne (ang. *intersection search*). Polega ono na jednoczesnym aktywowaniu takich samych procedur dla obu porównywanych ze sobą wierzchołków grafu i propagacji obliczeń we wszystkich kierunkach. Jeżeli w pewnym momencie obie procedury „spotkają się", oznacza to że dana relacja istotnie ma miejsce. Dużą wadą tego podejścia jest znaczne zwiększanie się czasu obliczeń wraz ze wzrostem rozmiaru i stopnia skomplikowania grafu. Ponadto dla przypadków negatywnych konieczne jest przeglądnięcie całego grafu, co w istocie jest niczym innym, jak przeglądem zupełnym danego problemu.

7.4.4. Ramki

Z reprezentacją wiedzy za pomocą grafów wiąże się kilka problemów. Jednym z nich jest prawdopodobieństwo pojawienia się różnych interpretacji dla danej relacji między konceptami. Na przykład definiując *lodówka* → *maszyna* trudno ocenić, czy projektant (bazy wiedzy) chciał zaznaczyć, że lodówka jest pewnym rodzajem maszyny, czy jakąś konkretnie zdefiniowaną maszyną, czy być może konkretną instancją lodówki. W literaturze tego typu nieścisłości określa się mianem logicznej nieprawidłowości (ang. *logical inadequacy*). Innym, problemem jest brak odpowiednich mechanizmów heurystycznych w algorytmach przeszukiwania. Człowiek, który starając się odpowiedzieć na pytanie dotyczące typu danego obiektu nie przegląda całej dostępnej mu wiedzy o świecie, a raczej bierze pod uwagę jedynie fakty potencjalnie związane z daną kategorią. Podobne zachowanie powinno charakteryzować heurystyczne algorytmy przeszukiwania grafów.

Ramki są pewnym sposobem reprezentacji wiedzy w postaci grafu. Pozwalają na unikanie nieścisłości oraz problemów związanych z przeszukiwaniem. Pojedyncza ramka odpowiada wierzchołkowi, w którym przechowywane są informacje o nazwie danego abstrakcyjnego typu danych oraz jego właściwościach. Intuicja podpowiada, że ludzki umysł nie stara się definiować konceptów zbyt szczegółowo i aby dany obiekt mógł przynależeć do danej klasy, nie musi wcale spełniać ściśle wszystkich założeń. Prowadzi to do definicji klas uogólnionych, nazywanych prototypami lub klasami prototypowymi. Obrazując to na przykładzie klasa ptaków, posiadająca właściwość *umie latać* jest klasą uogólnioną. Mogą do niej należeć ptaki, które nie latają. W ten sposób zarówno jastrząb, jak i emu mogą być reprezentantami klasy *ptaki* (z tym, że jastrząb wydawać się będzie lepszym przykładem ptaka).

Systemy oparte na ramkach starają się określić właściwości pewnych obiektów korzystając z prototypowych klas, co sprawdza się w większości przypadków. Czasami konieczne jest wprowadzenie pewnych modyfikacji w obiekcie dziedziczącym po danej klasie.

7.4.5. Przykłady zastosowania ramek

Istotę programowania w oparciu o ramki zilustrować można następującym przykładem. Niech dana będzie struktura danych (dla kompatybilności z tego typu językami programowania zastosowano angielskie nazwy zmiennych):

```
[PERSON is a kind of THING with
  AGE
  HEIGHT
  WEIGHT]
[PROPERTY is a kind of THING with
  UNITS
  RANGE]
[AGE is a PROPERTY with
  UNITS: YEARS
  RANGE: 0-120]
[HEIGHT is a PROPERTY with
  UNITS: METERS
  RANGE: 0-240]
[WEIGHT is a PROPERTY with
  UNITS: kg
  RANGE: 0-200]
[SCHOOl GRADUATE is a kind of PERSON with
  AGE: 19]
[PETER is a PERSON with
  AGE: 36
  HEIGHT: 180
  WEIGHT: 80]
[JOHN is a SCHOOL GRADUATE with
  HEIGHT: 170
  WEIGHT: 75]
[CAROLINE is a SCHOOL GRADUATE with
  AGE: 18
  HIGHT: 160
  WEIGHT: 55]
```

Klasą podstawową dla wszystkich konceptów jest tutaj klasa `Thing`, która powinna być interpretowana jako najprostszy byt. Klasa `Person` jest jakimś rodzajem klasy `Thing` (programista C++ mógłby powiedzieć, że jest to klasa pochodna klasy `Thing`), tak samo jak `Property`. Następnie klasy `Age`, `Height`, `Weight` definiowane są jako poszczególne instancje klasy `Property`, ze ściśle zdefiniowanymi polami `Units` oraz `Range`. Widać wyraźnie, że użytkownik ma tutaj do czynienia z dwoma typami relacji: *is a kind of* (*ako*), oraz *is a* (*isa*). Szukając analogii w teorii zbiorów można zauważyć, że relacja *ako* odpowiada podzbiorowi klasy nadrzędnej, a relacja *isa* odpowiada relacji przynależności do zbioru. Idąc dalej, relacja *ako* odpowiada deklaracji klasy dziedziczącej po klasie podstawowej w języku C++, natomiast relację *isa* można porównać do zdeklarowania obiektu danej klasy, to znaczy do powołania jej instancji.

Ciekawym jest, że instancje klas pochodnych wcale nie muszą posiadać takich samych atrybutów, co ich klasy prototypowe. W przykładzie powyżej istnieje klasa `School Graduate`, która modeluje abstrakcyjnego ucznia kończącego szkołę średnią przypisując mu parametr `Age`: 19. Zabieg ten ma swoje uzasadnienie, ponieważ większość uczniów kończy szkołę średnią w wieku 19 lat, choć zdarzają się uczniowie starsi lub młodsi. W tym wypadku instancja klasy `School Graduate` o nazwie `Caroline` jest przykładem osoby kończącej szkołę w wieku lat 18, więc parametr `Age` zostaje nadpisany. Dozwolenie takich operacji otwiera szereg możliwości dla programistów posługujących się ramkami i pozwala na bardzo dokładne i intuicyjne modelowanie rzeczywistości.

7.4.6. Problemy z systemami zorientowanymi obiektowo

Okazuje się, że mnogość możliwości jaką daje mechanizm dziedziczenia z jednej strony pozwala na bardzo dokładne i elastyczne modelowanie rzeczywistości, z drugiej strony jednak bardzo łatwo może prowadzić do nieścisłości i problemów z interpretacją. Wielokrotne dziedziczenie wymusza na programiście skonstruowanie pewnych prymitywnych typów, na których mogłyby bazować inne, bardziej skonkretyzowane koncepty. Te podstawowe klasy obiektów mogą jednak być dobrane na różne sposoby. Przykładowo dla jednej grupy ludzi `Pojazd` jest wystarczająco ogólną klasą, po której dziedziczyć powinien obiekt `Samochód`, podczas gdy inni uważają, że `Samochód` powinien być raczej rodzajem klasy `Środek transportu`. Ponadto samochody mogą być spalinowe/elektryczne, osobowe/ciężarowe/sportowe i niekoniecznie muszą pełnić rolę transportową w najbardziej podstawowym tego słowa znaczeniu.

Idąc dalej, należy zauważyć, że typ bazowy obiektu, na którym pracujemy, powinien czasami zależeć od czynności na nim przeprowadzanych. Jeżeli z samochodu korzysta kierowca, interesują go przede wszystkim takie parametry, jak: położenie, prędkość, przyspieszenie oraz ewentualny interfejs służący do sterowania. Jeżeli natomiast samochód będzie „wykorzystywany" przez zgniatarkę śmieci, istotnymi jego parametrami będą: masa, twardość, skład chemiczny. Do podobnej sytuacji dochodzi, gdy trzeba danemu pojazdowi wydać rozkaz „Jedź!". Znaczenie tego rozkazu jest takie samo dla samochodu, motocykla, czołgu czy motorówki (z grubsza), jednak jego wykonanie zależy ściśle od typu konkretnej maszyny.

Wspomniane właściwości nazywane są polimorfizmem i z pewnymi ograniczeniami zastosowane zostały już w wielu obiektowych językach programowania (C++, Java). Niestety, ze względów technicznych jeszcze nigdzie nie zaimplementowano pełnej gamy właściwości charakteryzujących obiekty ustrukturyzowane. Języki deklaratywne dedykowane dla ekspertowych systemów decyzyjnych, takie jak KRL czy FLAVORS, pozwalają na korzystanie z wielu (lecz nie ze wszystkich) wymienionych możliwości na raz. Warto w tym miejscu zwrócić uwagę na język systemu produkcji CLIPS, który dzięki zaimplementowanemu interpreterowi COOL (ang. *CLIPS Object Oriented Language*) wspiera wszystkie elementy paradygmatu programowania obiektowego.

7.5. Formalizmy logiczne systemów decyzyjnych

Spójny i jednoznaczny opis systemów decyzyjnych jest warunkiem koniecznym zarówno dla prowadzenia badań teoretycznych jak i praktycznych realizacji. Poniżej nakreślono ramy formalizmów logicznych wykorzystywanych w systemach decyzyjnych.

Słowo logika można odnieść do wielu gałęzi współczesnej nauki. Według klasycznej definicji jest to nauka o sposobach jasnego i ścisłego formułowania myśli, o regułach poprawnego rozumowania i uzasadniania twierdzeń. W matematyce, stosowanej przez inżynierów, ważny jest pewien sformalizowany zapis logiki zwany algebrą Boole'a. Wśród wielu logik stosowanych w informatyce chyba najbardziej znaną jest logika predykatów.

7.5.1. Rachunek zdań

Prezentacja zasad logiki predykatów zaczyna się zazwyczaj od omówienia składni i pewnych reguł obowiązujących w rachunku zdań. Rachunek zdań jest działem logiki matematycznej badającym związki między zmiennymi zdaniowymi (zdaniami) lub funkcjami zdaniowymi. W klasycznym rachunku zdań korzysta się z alfabetu, który z kolei składa się z trzech rodzajów znaków:

zmienne zdaniowe (zdania) : p, q, r, s itd., które mogą przyjmować wartość „prawda" albo „fałsz" (wartości 1 lub 0);

funktory zdaniotwórcze : negacja ($\neg$), koniunkcja ($\wedge$), alternatywa ($\vee$), równoważność ($\equiv$), implikacja materialna ($\rightarrow$);

znaki pomocnicze : różne nawiasy: $()[]\{\}$.

Aby wyrażenie zapisane za pomocą przedstawionego alfabetu miało sens, to znaczy było wyrażeniem sensownym (ang. *well-formed formula*), musi być zbudowane zgodnie z następującymi regułami:

1. każda pojedyncza zmienna zdaniowa jest formułą;
2. każde połączenie dwóch zmiennych zdaniowych p i q za pomocą funktora również jest formułą.

Jeżeli wyrażenie nie spełnia tych reguł, to nie jest wyrażeniem sensownym. Każda zmienna zdaniowa ma jakieś arbitralnie przyjęte znaczenie, jednak ta interpretacja nie wpływa w żaden sposób na prowadzenie rachunków. Dla przykładu można przyjąć, że P oznacza *Paweł jest człowiekiem*, a Q oznacza *Paweł jest śmiertelny*. Wartością wyrażenia $P \wedge Q$ może być prawda, jednak wynikać to będzie z faktu przypisania zdaniom P i Q wartości 1, a nie z faktu, że ludzie są śmiertelnikami. Wyrażania zawsze prawdziwe są nazywane *twierdzeniami* lub *prawami* (na przykład $(P \wedge Q) \rightarrow P$).

Rachunek zdań posiada trzy kluczowe cechy [4]. Są to:

1. zupełność (ang. *completness*) – jeżeli P jest twierdzeniem, to można je otrzymać korzystając jedynie z reguł dedukcyjnych (nie będą one tutaj opisywane);

2. solidność (ang. *soundness*) – nie istnieje takie P, że $P \wedge \neg P$ są twierdzeniami (innymi słowy, korzystając z reguł dedukcyjnych nie można otrzymać sprzeczności);
3. rozstrzygalność (ang. *decidability*) – dla każdego P istnieje procedura rozstrzygająca, czy P jest twierdzeniem.

Z zachowaniem powyższych cech wiążą się jednak pewne koszty. Mianowicie stosując rachunek zdań nie da się poprawnie operować na uogólnionych zdaniach typu *wszyscy ludzie są śmiertelni*. Uściślając, z koniunkcji $P \wedge Q$ zdań P *wszyscy ludzie są śmiertelni* i Q *Paweł jest człowiekiem* nie wynika zdanie *Paweł jest śmiertelny*. Aby możliwe było tego typu wnioskowanie, należy rozbić zdanie P na predykaty i argumenty w następujący sposób:

$$\forall x F(x) = \text{prawda}$$

co oznacza dokładnie: *dla każdego x, x spełnia własność F*.

W ten sposób od rachunku zdań dochodzi się do logiki predykatów. Jej nazwa bierze się właśnie z faktu rozbicia zdania na predykat i argument. Chcąc przedstawić fakt, że dany obiekt x posiada własność F, nie korzysta się już z jednej stałej zdaniowej P, tylko stawia predykat F przed zmienną (Fx). Jest to analogiczne do przypadku użycia operatora w algebrze liniowej. Podobnie wygląda zapis relacji R pomiędzy obiektami x_1, x_2, x_3..., z tą różnicą, że podaje się całą listę argumentów ($Rx_1x_2x_3$), której długość jest dowolna (a w abstrakcyjnych przypadkach - nieskończona). Pozostaje zapytać, czy w takim formalizmie zupełność, solidność i rozstrzygalność pozostają zachowane? Otóż okazuje się, że logiki predykatów nie cechuje rozstrzygalność. W przypadku określania relacji dla poszczególnych instancji (np. Fx, gdzie x = Paweł, F = Śmiertelny) zachowana jest zupełność i solidność (logika taka nazywana jest logiką pierwszego rodzaju), jednak gdy stosuje się kwantyfikatory $\forall x, \exists x$, zupełność nie jest już zachowana (logika taka nazywana jest logiką drugiego rodzaju).

7.5.2. Trudności wynikające ze stosowania logiki predykatów

Wśród popularnych języków wspierających logikę predykatów, które można zastosować do opisu rzeczywistości można wyróżnić KIF i CycL. Oba są językami deklaratywnymi i oba wspierają logikę predykatów pierwszego rodzaju. Jednak tylko CycL posiada dodatkowo rozszerzenie dla logiki predykatów wyższych rzędów i logiki modalnej. Oryginalnie w CycL stosowany był formalizm ramkowy, został jednak porzucony w późniejszych wersjach języka.

Logika predykatów jest chętnie wykorzystywana do udowadniania twierdzeń. Mechanizm dowodzenia polega na wyciąganiu kolejnych wniosków na podstawie dostarczonych faktów, tak długo, aż uzyskane zostanie zdanie udowadniające tezę. W przypadku dużych baz wiedzy proces taki może trwać bardzo długo, co jest oczywiście niedopuszczalne w przypadku systemów decyzyjnych (zwłaszcza takich, które mają działać w czasie rzeczywistym na fizycznym obiekcie). Dlatego też dąży się do jego usprawnienia, co w szczególnym wypadku może polegać na optymalym doborze akcji w zależności od zaistniałej sytuacji.

Podczas realizacji przedstawionej strategii można natknąć się na pewien paradoks. Okazuje się, że im więcej wiadomo na temat danego problemu, to znaczy im większy jest zakres przetwarzanej wiedzy, tym mniejsza jest szansa na znalezienie właściwego rozwiązania. Problem ten jest podobny do konfliktów występujących w systemach produkcyjnych, gdyż również w nim istnieje potrzeba zaimplementowania pewnych meta-reguł mówiących o tym, w którym obszarze dostępnej wiedzy należy szukać rozwiązania. Trudność w opracowaniu optymalnego mechanizmu w tym obszarze jest główną przeszkodą w budowaniu skomplikowanych systemów decyzyjnych bazujących na słabo ustrukturyzowanych i niesklasyfikowanych informacjach. Mimo tego takie systemy powstały i wciąż są tworzone. Metody ich budowy i rozwiązywania przedstawionych problemów szeroko opisano w [5].

7.6. Przykładowa realizacja systemu eksperckiego

W celu ilustracji przedstawionych problemów i zagadnień zbudowano prosty system decyzyjny wspomagający uczestników ruchu w bezpiecznej jeździe. Poniżej zaprezentowano ogólną ideę działania systemu oraz rozważono zagadnienia związane z doborem narzędzi pracy z perspektywy omówionej wcześniej teorii.

System początkowo był implementowany w systemie Linux w języku OPS5 (ang. *official production system*) - następcy języka XCON/R1 stworzonego dla komputerów VEX. Instalacja i konfiguracja środowiska pracy polegała, na instalacji interpretera dialektu Common Lisp – Clisp, skonfigurowaniu zarządcy pakietów, w tym wypadku Quicklispa oraz zaimplementowaniu metod języka OPS5 dla środowiska pracy. Mała elastyczność języka, różnice względem składni Lispa, a przede wszystkim zarzucenie języka OPS5 przez programistów (co wiąże się również z ubogą bazą dostępnej wiedzy) spowodowały decyzję o zmianie języka pracy na CLIPS. Doświadczenie to pozwoliło jednak zbudować podstawy do dalszych prac.

7.6.1. Języki OPS5 i CLIPS

Rodzina języków OPS5 powstała pod koniec lat 70-tych ubiegłego wieku. Stworzono ją, podobnie jak język Dendral, w oparciu o składnię języka Lisp. Charakteryzuje ją notacja przedrostkowej (prefiksowa) oraz stosowanie nawiasów - konsekwencji używania listy jako podstawowej struktury danych.

W językach o notacji wrostkowej (infiksowej) – przykładowo w C, C++, Javie – proste dodawanie zapisze się w sposób dość intuicyjny, mianowicie:

```
2 + 3;
```

Spodziewanym wynikiem będzie oczywiście liczba pięć. Ta sama operacja arytmetyczna w językach prefiksowych wyglądać będzie następująco (tu: zgodnie Lispem):

```
(+ 2 3)
```

CLIPS (ang. *C Language Integrated Production System*) jest wolnym językiem budowy systemów eksperckich. Został opracowany w latach 80-tych XX wieku przez naukowców z NASA. Inspirowany językiem OPS5, oryginalny interpreter CLIPSa został napisany w ANSI C [6]. Nowszy interpreter, napisany w Javie, nazywa się JESS (częściowo komercyjny). Daje to potencjalne możliwości integracji programów w CLIPSie w środowiskach ANSI C oraz Javy. O żywotności języka świadczy również fakt, iż doczekał się on dialektu dedykowanego logice rozmytej FuzzyCLIPS.

Interpreter CLIPSa jest dostępny zarówno dla systemów UNIX/GNU Linux (w konsoli), jaki i dla Mac OS oraz MS Windows (aplikacja okienkowa).

7.6.2. Baza wiedzy w CLIPSie

Baza wiedzy w CLIPSie ma postać zbioru faktów. Fakty te są dołączane do bazy instrukcją `assert`. Można je wylistować poleceniem `(facts)` jak w poniższym przykładzie.

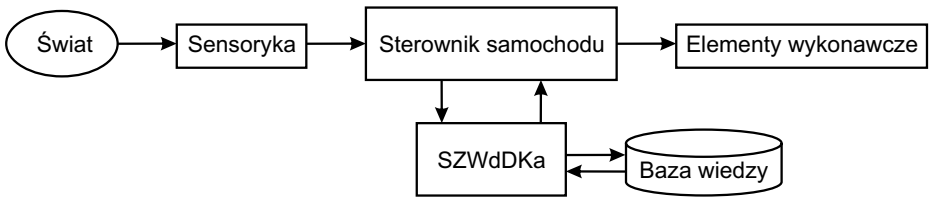
```
CLIPS> (assert (czlowiek Pawel))
<Fact-1>
CLIPS> (assert (wiek Pawel 24))
<Fact-2>
CLIPS> (facts)
f-0     (initial-fact)
f-1     (czlowiek Pawel)
f-2     (wiek Pawel 24)
For a total of 3 facts
```

Proces wnioskowania odbywa się poprzez rozumowanie w przód. Reguły (ang. *rules*) pozwalają na tworzenie nowych zdań logicznych na podstawie istniejących faktów. Zilustrowano to w poniższym przykładzie.

```
CLIPS> (defrule czy-zasilek
(czlowiek ?imie)
(wiek ?imie 24)
=>
(assert (zasilek ?imie)))
CLIPS> (rules)
czy-zasilek
For a total of 1 defrule.
CLIPS> (run)
CLIPS> (facts)
f-0     (initial-fact)
f-1     (czlowiek Pawel)
f-2     (wiek Pawel 24)
f-3     (zasilek Pawel)
For a total of 4 facts
```

7.6.3. SZWeDKa – System Zdalnego Wspomagania Decyzji Kierowcy

System SZWEDKA został zaimplementowany w środowisku CLIPS 6.24. Jest więc rozwiązaniem przenośnym pomiędzy różnymi systemami operacyjnymi. Ideę jego działania przedstawiono na rys. 7.1. W systemie tym dane o świecie zewnętrznym zaobserwowane przez system sensoryczny są przekazywane do sterownika samochodu. Tam następuje akwizycja danych i ich interpretacja. Sterownik, w zależności od typu danych i stopnia ich złożoności, wysyłana dane sterujące na elementy wykonawcze bądź przesyła informacje do Systemu Zdalnego Wspomagania Decyzji Kierowcy. Ten, na podstawie istniejącej bazy wiedzy, dokonuje procesu rozumowani w przód i podejmuje decyzje co do zachowania się pojazdu. Poprzez sterownik samochodu są one kierowane na elementy wykonawcze, takie jak: wyświetlacz na desce rozdzielczej, sterownik świateł, tempomat, układ hamulcowy itd.



Rys. 7.1: Diagram przepływu danych w systemie.

System posiada bazę wiedzy w postaci faktów opisujących część znaków drogowych oraz typowe zdarzenia na drodze (piesi, hamujący inny uczestnik ruchu). Pojawienie się konkretnych instancji wymienionych zdarzeń jest sygnalizowane poprzez system sensoryczny (źródłem sygnałów o zdarzeniach drogowych może być użytkownik). Dodaje on do bazy wiedzy fakty wywołujące konkretne reguły wnioskowania. W ich wyniku następuje podejmowanie decyzji.

Baza wiedzy – fakty

Baza wiedzy w systemie SZWeDKa została odseparowana od modułu wnioskowania. Pozwala to na zachowanie logicznego porządku programu oraz na bezpieczne modyfikacje jednej części bez ingerencji w drugą.

Baza wiedzy reprezentowana jest jako zbiór faktów. Dostęp do aktualnej wiedzy systemu jest możliwy w każdym momencie, m.in. poprzez wywołanie komendy `(lista-faktow)`, zobacz podrozdział 7.6.3).

System rozpoczyna swoją pracę z pewną wiedzą początkową. W szczególności – z wiedzą na temat pojazdu wspomaganego przez system SZWeDKa (pojazd *vovlo*). Deklaracja wartości początkowych wiedzy systemu wygląda następująco:

```
(deffacts szwedka
  (volvo (akcja postoj) ; domyslnie vovlo sie nie porusza
         (predkosc 0)
         (pas 0) ; domyslnie vovlo stoi na poboczu
         (swiatla 0)) ; swiatla zgaszone
```

```
  (pora dzien) ; pora dnia
  (kraj polska)
)
```

W powyższym kodzie wyróżnić można następujące fakty: `volvo`, `pora`, `kraj`. Pierwszy bazuje na zdefiniowanym wcześniej szablonie i opisuje stan pojazdu. Następne odnoszą się do pory dnia oraz kraju, w którym porusza się pojazd. Tak zdefiniowany szablon jest zgodny z szablonem opisanym w podrozdziale 7.3.3 – wspomniane fakty można przedstawić w równoważnej postaci odseparowanych trójek:

```
(assert (volvo akcja postoj))
(assert (volvo predkosc 0))
(assert (volvo pas 0))
(assert (volvo swiatla 0))
```

W trakcie działania systemu, system sensoryczny dodaje nowe fakty do bazy wiedzy, system zaś na podstawie zdefiniowanych reguł przeprowadza proces wnioskowania w przód. Jego wynikiem są decycje odnośnie zmiany stanu pojazdu.

Interfejs – funkcje

Użytkownik korzystający z systemu SZWeDKa posługuje się zaimplementowanymi funkcjami sterującymi. Pozwalają one na dodawanie nowych zdarzeń bądź usuwanie zdarzeń rejestrowanych oraz ich późniejszą obsługę. Jest to dogodne rozwiązanie w przypadku zagnieżdżania systemu produkcyjnego w programie napisanym np. w języku C [7]. Pozwala to bowiem uzyskać klarowny protokół komunikacji. Przy zastosowanym symulowaniu odczytów sensorycznych ma to tę zaletę, że zwalnia użytkownika ze znajomości składni języka CLIPS. Wreszcie, jest to wygodny sposób na zwiększenie odporności programu przed *trudnymi* użytkownikami.

Lista komend udostępnionych użytkownikowi jest następująca:

`(pomoc)` – wyświetla listę dostępnych komend;

`(inicjalizacja-systemu)` – czyści aktualną i generująca nową listę faktów startowych: pojazd volvo wraz z atrybutami, pora dnia, kraj, pogoda;

`(start)` – startuje proces wnioskowania: sprawdzane są warunki reguł dla aktualnej bazy faktów oraz wykonywane są ewentualne akcje;

`(lista-faktow)` – wypisuje aktualne fakty z bazy wiedzy;

`(ruszaj #WARTOŚĆ)` – rusza pojazd z miejsca i ustawia jego prędkość na wartość przekazaną w parametrze `#WARTOŚĆ`. Ponadto atrybutowi `akcja` pojazdu volvo przypisuje wartość `ruch`;

`(hamuj)` - zatrzymuje pojazd ustalając prędkość na wartość 0 oraz atrybut `akcja` na wartość `postoj`;

`(zmien-pas #WARTOŚĆ)` – zmienia pas ruchu na ten o numerze danym wartością `#WARTOŚĆ`. Numeracja pasów przyjmuje wartości całkowite i rośnie od osi jezdni (1, 2, 3 ...) w kierunku ruchu oraz maleje w kierunku przeciwnym do ruchu (-1, -2, ...). Liczba 0 oznacza pobocze po prawej stronie;

`(wlacz-sie-do-ruchu)` – zjeżdża z pobocza na skrajny prawy pas jezdni;

`(zmien-pore-dnia)` – zmienia porę dnia na przeciwną (dzień/noc);

`(zmien-kraj #WARTOŚĆ)` – zmienia kraj, w którym porusza się system (ma to znaczenie np. przy sterowaniu światłami);
`(dodaj-pojazd #PRD #PS #PLZ #ODL)` – dodaje do bazy wiedzy nowy pojazd (obiekt typu auto) o parametrach: prędkość `#PRD`, pas `#PS`, położenie `#PLZ`, odległość `#ODL`;
`(usun-pojazd #NUMER)` – usuwa z bazy wiedzy pojazd o numerze `#NUMER`;
`(dodaj-znak1 #ID)` – dodaje do bazy wiedzy nowy znak o identyfikatorze `#ID`;
`(dodaj-znak2 #ID #VAR)` – dodaje do bazy wiedzy nowy znak o identyfikatorze `#ID` oraz atrybucie `#VAR`;
`(usun-znak #ID)` – usuwa z bazy wiedzy znaki o identyfikatorze `#ID`;
`(pieszy)` – dodaje do bazy faktów pieszego na pasach;
`(pojazd-krzyzowka)` – dodaje do bazy faktów pojazd na skrzyżowaniu na drodze poprzecznej;
`(pojazd-rownorzedny)` – dodaje do bazy faktów pojazd na skrzyżowaniu równorzędnym po stronie określonej parametrem `#STRONA` (lewo/prawo).

Tak zdefiniowane funkcje pełnią rolę wygodnego interfejsu użytkownika. Niemniej poszczególne reguły programu mogą wykorzystywać funkcje, jako część wynikową (prawa strona, ang. *right-hand side, RHS*) w swojej konstrukcji. Aby to było możliwe należy pamiętać, by elementy, do których odwołują się reguły bądź funkcje były wcześniej zdefiniowane w kodzie. Szczegóły konstrukcji kodu omówione zostały w dokumentacji HTML. Ogólne wytyczne konstrukcji reguł i funkcji zostały zamieszczone w następnej sekcji.

Należy podać również listę możliwych do zadeklarowania znaków. Ich definicje atrybuty wywołania zostały wypisane w tab. 7.1. Przykładowo, chcąc wprowadzić do bazy wiedzy wykryty przez system sensoryczny znak ograniczenia prędkości do 50 km/h, użytkownik winien wprowadzić formułę w następującej formie:

```
CLIPS > (dodaj-znak2 predkosc 50)
```

Tab. 7.1: Zaimplementowane definicje znaków pionowych.

Nazwa znaku wg kodeksu drogowego	Komenda w programie	
Znaki wywoływane funkcją `dodaj-znak1`		
B-20: stop	`stop`	
B-25: zakaz wyprzedzania	`wyprzedzanie`	
B-27: koniec zakazu wyprzedzania	`koniec-wyprzedzania`	
B-34: koniec ograniczenia prędkości	`koniec-predkosci`	
A-7: ustąp pierwszeństwa	`ustap`	
Znaki wywoływane funkcją `dodaj-znak2`		
B-33: ograniczenie prędkości	`predkosc`	`#VAR` = wartość liczbowa
B-21/22: zakaz skręcania	`skret`	`#VAR = lewo | prawo`

Wnioskowanie

Każdorazowe wywołanie funkcji `(start)` powoduje, że CLIPS dokonuje procesu wnioskowania w oparciu o bazę wiedzy i zdefiniowane reguły. W tej sekcji nie zostały omówione wszystkie zdefiniowane reguły. Ich szczegółowy opis przedstawia dokumentacja w pliku HTML oraz komentarze w kodzie programu. Zostaną za to wskazane ogólne prawidła konstrukcji reguł w odniesieniu do wcześniejszej teorii.

W rozdziale 7.3.4 opisane zostały metody rozwiązywania konfliktów podczas procesu wnioskowania. Celem uniknięcia problemów zapętlania się programu, kod systemu SZWeDKa wdraża strategię odporną rozwiązywania konfliktów. Fakty komunikacyjne, instancje wydarzeń, ale i samego pojazdu volvo, w bazie wiedzy są kasowane tak, aby nie dochodziło do zapętleń programu. Funkcje interfejsu nie wywołują wprost żadnych procedur modyfikujących istniejącą bazę wiedzy, ani nie usuwają jej elementów. Funkcje te dodają nowe fakty (nazwane przez autorów *komunikacyjnymi*), które są następnie przetwarzane przez zdefiniowane reguły. Przykładowo, kod funkcji `ruszaj` wygląda następująco:

```
(deffunction ruszaj
  (?nowa)
  (assert (nowosc predkosc ?nowa))
  (assert (nowosc akcja ruch))
  (assert (nowosc komunikat predkosc "Volvo ruszylo! Predkosc: "))
)
```

Zapis `(?nowa)` określa pobiera argument funkcji do zmiennej. Wartość argumentu, tutaj prędkości pojazdu, będzie wykorzystana dalej. Kolejne trzy linie kodu dodają do bazy wiedzy trzy nowe fakty. Dotyczą one, odpowiednio, nowej prędkości, nowej akcji oraz komunikatu jaki zostanie wyświetlony przy okazji zmian parametrów pojazdu `volvo`. Fakty te zostaną przetworzone przez następującą regułę:

```
(defrule ruch-volvo
  ?v  <- (volvo (predkosc ?)(pas ?ps)(akcja ?)(swiatla ?swt))
  ?vp <- (nowosc predkosc ?np)
  ?va <- (nowosc akcja ?na)
  ?vk <- (nowosc komunikat predkosc ?nk)
=>
  (retract ?v)
  (assert (volvo (predkosc ?np)(pas ?ps)(akcja ?na)(swiatla ?swt)))
  (printout t ?vk ?np crlf)
  (retract ?vp)
  (retract ?va)
  (retract ?vk)
)
```

Sprawdza ona w pierwszej kolejności istnienie pojazdu volvo o podanych parametrach oraz pobiera wskaźnik na jego miejsce w bazie wiedzy (zapis `?v <-`) [8]. Następnie znajduje fakt zmiany prędkości, zmiany akcji opisującej stan `volvo` oraz komunikat odnośnie ruchu. Do wszystkich trzech faktów deklarowane są wskaźniki tak, aby można było później usunąć niepotrzebną wiedzę. W tym miejscu warto zwrócić uwagę, że wszystkie wyrażenia po lewej stronie znaku `=>` są niejawnie złączone znakiem koniunkcji. Niewystąpienie któregoś z wymaganych faktów, zatrzyma wykonywanie procedury.

Pierwsza linijka po prawej stronie reguły opisuje usunięcie (`retract`) dotychczasowej instancji `volvo`. Następnie, zostaje ona zastąpiona nową, z aktualnymi wartościami liczbowymi, po czym na standardowe wyjście zostaje wyświetlony komunikat o aktualnej prędkości pojazdu. Na końcu usuwane są fakty o nowej prędkości, akcji i komunikacie.

Powyższa reguła wpływa na przetwarzanie pewnych faktów. Nic nie stoi na przeszkodzie, by były one generowane przez funkcje inne niż `ruszaj`. Funkcja `hamuj` również *współpracuje* z regułą `ruch-volvo`:

```
(deffunction hamuj
 ()
 (assert (nowosc predkosc 0))
 (assert (nowosc akcja postoj))
 (assert (nowosc komunikat predkosc "Volvo zahamowalo! Predkosc: "))
)
```

Jej wykonanie, jak można się domyślić, zatrzyma pojazd.

Rozwój

SZWeDKa jest dobrym przykładem działania systemu produkcyjnego. Implementacja go na rzeczywistym obiekcie wymaga oczywiście jeszcze dużych nakładów pracy. Warto w tym miejscu zwrócić uwagę na dydaktyczny charakter systemu. Pokazano, że celem obsługi prostych zdarzeń zaobserwowanych na drodze należy konsekwentnie konstruować funkcje wg jasnych założeń i strategii. Wyraźnie wyróżniony moduł interfejsu użytkownika pozwala na integrację systemu z kodem w języku C. Użycie interpretera JESS daje możliwość łączenia SZWeDKi z programami pisanymi z języku Java.

7.7. Podsumowanie

W podrozdziałach 7.3–7.5 przedstawiono trzy różne metody reprezentacji i przetwarzania wiedzy dla potrzeb systemu wnioskującego. Nie jest łatwo stwierdzić, który z nich nadaje się najlepiej do systemów decyzyjnych działających w kontekście pojazdów autonomicznych. W każdym z pewne aspekty rozwiązane są lepiej, podczas gdy inne stwarzają problemy. Analizę przeprowadzono posługując się w dużym stopniu pozycją [4], gdzie oprócz informacji przedstawionych w tym rozdziale można znaleźć wiele praktycznych przykładów działających systemów.

Systemy produkcyjne dostarczają relatywnie prosty i szybko działający interfejs do zbudowania bazy wiedzy i określenia reguł, na podstawie których podjęta zostanie decyzja. Istnieje możliwość ograniczonego wpływania na kierunek wnioskowania systemu, dzięki czemu można uniknąć niejednoznaczności i sytuacji, w której system traci zbyt wiele czasu na wygenerowanie odpowiedzi.

Można zauważyć, że wśród trzech przedstawionych podejść, to właśnie obiekty ustrukturyzowane pozwalają na najwierniejsze odzwierciedlenie rzeczywistości. Mechanizm dziedziczenia praktycznie z miejsca pozwala uzyskać informacje na temat typu obiektu, w związku z czym możliwe jest bardzo proste zastosowanie odpowiedniego obszaru wiedzy do odpowiedniego typu danych, który jest aktualnie w użyciu. Duża swoboda dostarczona przez ten formalizm może jednak spowodować powstawanie niejednoznaczności i w efekcie zwracać błędne i całkowicie nieoczekiwane rezultaty. Projektowanie skomplikowanego systemu bazującego na obiektach ustrukturyzowanych wymaga od programisty szczególnej precyzji i konsekwencji, co nie zawsze jest w stu procentach spełnione.

Logika predykatów jest potężnym narzędziem wnioskującym i potencjalnie pozwala na podejmowanie optymalnych decyzji parametryzowanych aktualnym kontekstem. Projektując systemy z dużą bazą wiedzy, trzeba jednak pamiętać o dosyć wolnych metodach dopasowujących zapytania do pamięci operacyjnej. Może się zdarzyć, że system oparty na logice, pomimo posiadania wszelkich informacji potrzebnych do podjęcia określonej decyzji zawiedzie, ponieważ interpreter zacznie przeszukiwać bazę w złym miejscu i całość operacji zajmie zbyt wiele czasu.

Ekspertowy system decyzyjny działający na pokładzie autonomicznego samochodu powinien opierać się na prostej wiedzy dostarczonej przez kierowców w formie: jeżeli TO, zrób TAMTO. Oprócz tego wymaga się, aby program taki był w stanie dostosowywać się do warunków panujących na drodze, a więc pewne elementy *machine learning* są tutaj niezbędne. Wydaje się za to, że nie istnieje potrzeba każdorazowego przeglądania całej dostępnej pamięci, tak samo jak nie potrzebne jest wyciąganie nadmiarowych wniosków.

Podsumowaniem informacji przedstawionych w poprzednich rozdziałach, jest system SZWeDKa bazujący na regułach produkcyjnych. Choć zastosowanie elementów znanych z paradygmatu programowania obiektowego nie zostało uwidocznione, zauważa się potrzebę ich stosowania, w szczególności, gdy istnieją gotowe narzędzia programistyczne (np. język COOL). W przedstawionym systemie uzyskano wystarczającą szybkość działania, prostotę implementacji reguł oraz możliwość klasyfikacji obiektów takich jak znaki drogowe. Warto wspomnieć, iż systemy takie jak SZWeDKa funkcjonują już w pojazdach dostępnych dla przeciętnego użytkownika. Najczęściej monitorują one znaki drogowe, przypominając kierowcy np. o aktualnym ograniczeniu prędkości. Zaawansowane systemy decyzyjne można spotkać w pojazdach autonomicznych tworzonych na całym świecie. System SZWeDKa może być zalążkiem takiego samodzielnego systemu.

Literatura

[1] R. Schalkoff. *Intelligent Systems: Principles, Paradigms and Pragmatics: Principles, Paradigms and Pragmatics.* Jones & Bartlett Learning., 2011.
[2] C. Krishnamoorthy, S. Rajeev. *Artigicial Intelligence and Expert Systems for Engineers.* CRC-Press, 1996.
[3] D. Klahr, P. Langley, R. T. Neches. *Production System Models of Learning and Development (Computational Models of Cognition and Perception).* Cambridge, Mass.: The MIT Press, 1987.
[4] P. Jackson. *Introduction to expert systems.* International Computer Science Series. Wokingham : Addison-Wesley Publ. Co., 1986.
[5] D. Merritt. *Building Expert Systems in Prolog.* Amzi! inc., 1989.
[6] *CLIPS Reference Manual. Volume I. Basic Programming Guide*, 2006.
[7] *CLIPS Reference Manual. Volume II. Advance Programming Guide*, 2006.
[8] J. Giarratano. *CLIPS User's Guide*, 2007.

ROZDZIAŁ 8

GENERATOR UNIKALNYCH IDENTYFIKATORÓW

I. Góral, K. Szydłowski

W niniejszym rozdziale przedstawiono zagadnienia związane z zastosowaniem oraz sposobami generowania uniwersalnych identyfikatorów UID (ang. *Unique Identifiers*). Materiał ten zredagowano z myślą o czytelnikach zaznajomionych z tematem, jak również o czytelnikach, którzy usłyszeli o nim po raz pierwszy. Powstało więc opracowanie, w którym podano definicję unikalnego identyfikatora oraz scharakteryzowano obszary jego zastosowania. Ponadto dokonano w nim przeglądu istniejących unikalnych identyfikatorów wraz metodami ich generowania oraz opisano próbę zdefiniowania i implementacji uniwersalnego narzędzia do tworzenia i dekodowania unikalnych identyfikatorów.

8.1. Wprowadzenie

Budowa systemów do efektywnego archiwizowania oraz analizowania danych jest jednym z kluczowych zadań wpływających na rozwój różnych gałęzi gospodarki. Podstawą funkcjonowania takich systemów jest jednoznaczne rozróżnianie zawartych w nich informacji, co najczęściej realizowane jest poprzez zastosowanie unikalnych identyfikatorów.

Zgodnie z definicją [1], identyfikator jest nazwą służącą do oznaczenia obiektów (zmiennych, plików, instytucji itd.). Nadanie mu cechy unikalności sprawia, że może reprezentować przypisane mu pojęcia w sposób jednoznaczny – konkretny identyfikator może być przypisany wyłącznie do jednego obiektu, jakkolwiek pojedynczy obiekt może posiadać kilka identyfikatorów (reprezentujących go np. w różnych bazach wiedzy). Zakres unikalności identyfikatora wynika bezpośrednio ze środowiska danych, w którym jest on wykorzystywany. Można wyróżnić następujące rodzaje zakresów unikalności:

lokalny – identyfikator występuje w prywatnej bazie wiedzy o niewielkim zasięgu (przestrzeń nazw komputera osobistego, katalog firmy, system bankowy itd.);

krajowy – identyfikator wykorzystywany głównie w bazach wiedzy o zasięgu pojedynczego kraju (stosowany głównie w administracji publicznej: wykaz akt Ministerstwa Spraw Wewnętrznych, Powszechny Elektroniczny System Ewidencji Ludności (PESEL), Księga Ewidencji Robót Geodezyjnych (KERG) [2] itd.);

strukturowy – identyfikator stosowany w bazach wiedzy obejmujących całą organizację lub zrzeszenie o różnym zasięgu (rozpinających się np. na Unię Europejską lub rozległe portale internetowe, jak np. `wikipedia.org`);

globalny – identyfikator zachowuje swoją unikalność w światowej bazie wiedzy (np. Międzynarodowy Znormalizowany Numer Książki (ISBN, ang. *International Standard Book Number*), Seryjny Numer Jednostki Wysyłkowej (SSCC, ang. *Serial Shipping Container Code*)).

Zaproponowany podział zakresu unikalności identyfikatorów nie jest formalnie przyjęty, pomaga jednak zaplanować obszar stosowania nowo tworzonego identyfikatora a także związane z nim parametry (w tym liczbę wykorzystywanych znaków). Innym rodzajem klasyfikacji UID jest podział za względu na kodowaną informację [3]. W podziale tym wyróżnia się dwie grupy identyfikatorów:

niosące dane – ich postać przechowuje informacje o obiektach, którym zostały przypisane,

losowe – ich postać nie pozwala wywnioskować żadnych informacji o obiektach, którym zostały przypisane.

Identyfikatory przechowujące dane powstają w oparciu o wcześniej zdefiniowany klucz, który pozwala jednoznacznie zakodować i odkodować wybraną informację. Przykładem takiego identyfikatora jest wspomniany już numer PESEL, którego pierwsze sześć znaków odpowiada dacie urodzenia, a dziesiąty płci danej osoby. Identyfikatory drugiego rodzaju powstają najczęściej przy wykorzystaniu generatorów liczb pseudolosowych. W ich skład wchodzą losowe ciągi znaków niezwiązane znaczeniowo z identyfikowanym obiektem. Przykładem takiego identyfikatora jest UUID (ang. *Universally Unique Identifier*) stosowany jako standard identyfikacji wolnego oprogramowania.

UID znajdują zastosowanie we wszystkich dziedzinach związanych z administracją i przemysłem. Wprowadzenie unikalnych identyfikatorów pozwala na efektywne zarządzanie produktami, usługami oraz na jednoznaczne reprezentowanie podmiotów fizycznych (tj. organizacji, firm, osób). Cechy te wykorzystywane są zwłaszcza w administracji publicznej, która nieustannie rozwija się i digitalizuje, chcąc sprostać coraz to nowym wymaganiom prawnym. Warto zauważyć, że UID stanowią podstawę wszelkich operacji katalogowania informacji (tworzenia obszernych baz wiedzy) oraz stosowania szeroko pojętych technologii informatycznych (jako dowód można rozważyć identyfikator URI, opisany w podrozdziałach 8.2.2 i 8.2.4).

8.2. Przegląd istniejących UID

Poniżej wymieniono wybrane unikalne identyfikatory mające zastosowanie w różnych obszarach dziedziny społecznej i gospodarczej. Stanowią one przykład powszechności wykorzystania technologii UID.

8.2.1. Administracja

UID służy głównie do ewidencjonowania produktów, usług, dokumentów itd. zarówno w obszarach zarządzanych przez instytucje administracji publicznej, jak również różnego rodzaju organizacje oraz firmy prywatne. Poniżej wymieniono przykłady powszechnie stosowanych unikalnych identyfikatorów.

Numer akt MSW – od 2012 roku obowiązuje system Jednolitego Rzeczowego Wykazu Akt wprowadzony przez Ministerstwo Spraw Wewnętrznych. Jego funkcjonowanie polega na oznaczaniu teczek dokumentów dotyczących poszczególnych spraw według określonego schematu. Ma to pozwolić na efektywne archiwizowanie i wyszukiwanie informacji. Dokładny opis systemu zawiera podrozdział 8.3.2.

Identyfikatory w rejestrach administracji publicznej – rejestry stanowią podstawowe narzędzie do gromadzenia informacji o obywatelach, podmiotach fizycznych, instytucjach i organizacjach działających na terenie państwa, w celach statystycznych, gospodarczych, obronnych czy o charakterze kryminologicznym. Każdy rejestr charakteryzuje się indywidualnym sposobem przyznawania identyfikatorów – niektóre są nadawane automatycznie wszystkim obywatelom (jak numer PESEL), inne wymagają procedury wystąpienia o ich nadanie (np. Numer Identyfikacji Podatkowej, NIP). Przykładami istniejących rejestrów państwowych są:

- KRAZ, Krajowy Rejestr Agencji Zatrudnienia;
- KRS, Krajowy Rejestr Sądowy;
- REGON, Krajowy Rejestr Urzędowy Podmiotów Gospodarki Narodowej;
- CEPiK, Centralna Ewidencja Pojazdów i Kierowców;
- PRNG, Państwowy Rejestr Nazw Geograficznych;
- PRG, Państwowy Rejestr Granic i Powierzchni Jednostek Podziałów Terytorialnych Kraju;
- ROG, Rejestr Obszarów Górniczych;
- CBD, Centralna Baza Danych Ministerstwa Infrastruktury;
- KEP, Krajowa Ewidencja Podatników;
- Jednolity Rejestr Państwowy;
- Państwowy Rejestr Muzeów.

ISBN – jest to unikalny, trzynastocyfrowy Międzynarodowy Znormalizowany Numer Książki (ang. *International Standard Book Numer*) nadawany w Polsce przez Krajowe Biuro ISBN. Służy do jednoznacznej identyfikacji wydawcy oraz

wydawanych przez niego wydawnictw zwartych[1]. Jest wykorzystywany nie tylko przez wydawców, ale również przez sprzedawców, biblioteki i hurtownie książek podczas identyfikacji, klasyfikacji i kontroli stanów magazynowych. Wydawnictwa, które składają się z wielu tomów (np. encyklopedie) dla każdego tomu otrzymują osobny numer oraz jeden zbiorczy numer, który pozwala na zidentyfikowanie cyklu jako całości. Raz nadany numer ISBN nie może zostać użyty ponownie, ani nie może zostać zastąpiony. Numer ISBN książki jest zachowany dla każdego kolejnego jej dodruku. Nowy numer powinien jednak zostać nadany wznowieniu (reedycji) książki[2].

Trzynastocyfrowy numer ISBN[3] składa się z pięciu członów [5]:

1. prefiks lub Europejski Kod Towarowy EAN (ang. *European Article Number*), obecnie o postaci `978` lub `979`,
2. identyfikator grupy rejestracyjnej (np. `92` oznacza organizacje międzynarodowe),
3. identyfikator wydawcy,
4. identyfikator tytułu publikacji (właściwy konkretnemu wydaniu publikacji wydawcy),
5. znak kontrolny.

Ostatecznie, identyfikator przyjmuje przykładowo postać:
`ISBN 978-92-77-001477-3`.

ISSN – jest to unikalny, ośmiocyfrowy Międzynarodowy Znormalizowany Numer Wydawnictwa Ciągłego (ang. *International Standard Serial Number*) nadawany w Polsce przez Narodowy Ośrodek ISSN. Przyznawany jest dla wydawnictw ciągłych, czyli takich, których data zakończenia nie jest z góry planowana [5], np. periodyk, czasopismo, gazeta. Bazuje na koncepcji ISBN. Niektóre publikacje mogą posiadać przydzielony zarówno numer ISBN, jak i ISSN, ponieważ oba wykazy są od siebie niezależne. Numer ISSN jest nadawany pojedynczo, dla całego cyklu. Jeśli ISSN został nadany cyklowi książkowemu, każdemu tomowi osobno można równocześnie nadać numer ISBN.

Ośmiocyfrowy numer ISSN składa się z dwóch, czterocyfrowych członów i przyjmuje przykładowo postać:
`ISSN 1742-786X`.

8.2.2. Przemysł

URI – w przemyśle internetowym jest standardem, który pozwala na łatwą identyfikację zasobów w sieci. URI (ang. *Uniform Resouce Identifier*) składa się zazwyczaj z ciągu znaków sformatowanych zgodnie z odpowiednim standardem. Określenie danego zasobu może odbywać się za pomocą nazwy URN

[1] Wydawnictwami zwartymi są: książki i broszury, mapy, książki mówione, cyfrowe kopie książek drukowanych, oprogramowanie edukacyjne lub instruktażowe itp. [4].

[2] Wznowienie: „publikacja różniąca się od wydań poprzednich zmianami wprowadzonymi w treści (wydanie przejrzane) lub w kompozycji (nowe wydanie) i wymagająca nowego numeru ISBN" [4].

[3] Do końca 2006 r. numer ISBN był złożony z 10 cyfr [4].

(ang. *Uniform Resource Name*) lub jego lokalizacji URL (ang. *Uniform Resource Locator*). URN i URL są uznawane za szczególne przypadki URI – jeśli nazwa lub adres identyfikują dany zasób w niepowtarzalny, wyjątkowy, czyli unikalny sposób, URL lub URN samodzielnie mogą zostać potraktowane jako URI. URI może identyfikować dany obiekt jako część hierarchicznej struktury lub jako element niehierarchiczny. Ogólny wzór identyfikatora stosowanego w Internecie wygląda następująco [6]:

`<schemat>:<część hierarchiczna lub niehierarchiczna danego schematu>`

Schemat określa protokół (np. `http` dla sieci `www`) jaki należy zastosować przy interpretacji identyfikatora jako żądania udostępnienia wskazanego zasobu. To, że element należy do części hierarchicznej schematu można rozpoznać po wystąpieniu symbolu / (ukośnika, ang. *slash*) przed identyfikowanym elementem, np.:

`http://pwr.wroc.pl/index.dhtml`

Elementy niehierarchiczne wskazywane są bezpośrednio po dwukropku, np.:

`mailto:student@pwr.wroc.pl`

URL – specyficzny rodzaj URI, który identyfikuje zasób poprzez wskazanie jego lokalizacji [7, 8, 9]. URL w sieciach internetowych jest zazwyczaj tożsamy z URI, ponieważ lokalizacja zasobów wystarcza do ich jednoznacznej identyfikacji. Najbardziej znane URL wykorzystuje schemat `http` i przyjmuje postać:

`http://adres_serwera/sciezka_dostepu_do_zasobu`

Pozostałe schematy URL to:

- `ftp` – protokół transferu plików (ang. *File Transfer Protocol*),
- `gopher` – protokół `Gopher`,
- `mailto` – elektroniczny adres mailowy,
- `news` – wiadomości `USENET`,
- `nntp` – wiadomości `USENET` wykorzystujące dostęp `NNTP`,
- `telnet` – odwołanie do sesji interaktywnych,
- `wais` – serwery informacyjne (ang. *Wide Area Information Servers*),
- `file` – nazwy zasobów/plików na danym hoście,
- `prospero` – protokoły należące do systemu *Prospero Directory Service*.

Identyfikator transakcji – jest stosowany szeroko w systemach bankowych, aukcyjnych i sklepach internetowych. Identyfikator tego typu jednoznacznie wskazuje na dany zasób. Przykładowo, systemy bankowe nadają unikalny numer każdej transakcji pieniężnej wykonywanej przez klienta banku. Każda aukcja na `allegro.pl` posiada również swój unikalny numer. Dzięki temu możliwa jest archiwizacja danych i dostęp do nich. Identyfikatory te mają zazwyczaj postać ciągu cyfr o określonej długości. Unikalność tych numerów jest ograniczona do systemu, który je generuje (danego banku, czy serwisu internetowego).

8.2.3. Społeczność

Login – w serwisach społecznościowych, zakupowych, e-learningowych (a więc wszędzie tam, gdzie możliwa lub konieczna jest rejestracja konta) potrzebne są identyfikatory, które pozwolą na zalogowanie do tych kont. Login musi być identyfikatorem unikalnym, ale jedno konto może posiadać więcej niż jeden login. Przykładowo, niektóre serwisy pozwalają na logowanie się na konto zarówno przy użyciu nicka (nazwy użytkownika), jak i adresu mailowego przypisanego do konta. Login może być tożsamy nickowi, który widzą inni użytkownicy serwisu lub odrębny od niego. Loginy są identyfikatorami, których odbiorcami są nie tylko systemy informatyczne, ale również ich użytkownicy. Każdy użytkownik powinien być w stanie zapamiętać swój identyfikator, więc przyznawanie użytkownikom identyfikatorów w formie ciągu cyfr lub losowego ciągu znaków nie spełniałoby swojej roli. Dlatego loginy zazwyczaj mogą być wymyślane przez użytkowników (i weryfikowane przez system pod względem unikalności). Identyfikatory te są niepowtarzalne w zakresie danego serwisu.

nick – unikalna nazwa użytkownika przypisana do konta, widoczna dla innych użytkowników. Może, ale nie musi być równocześnie loginem. Podobnie jak login, jest wymyślany przez użytkownika, ponieważ powinien pozwalać na szybką i jednoznaczną identyfikację innym osobom. Nicki przyjmują zazwyczaj formę ciągu liter i cyfr i często posiadają semantykę. Unikalność nicków jest ograniczona do serwisu, w którym zostało zarejestrowane dane konto.

8.2.4. Ontologia

Ontologia jest dziedziną myśli filozoficznej zajmującą się opisem rzeczywistości. Mówiąc w najszerszym kontekście zajmuje się ona opisem istoty wszystkiego, co istnieje materialnie i niematerialnie. Ontologia stara się wykraczać poza granice poznania, co w najwcześniejszych formach wyrażone było np. koncepcją Idei Platona. W praktyce jednak zadanie to jest niemożliwe lub bardzo trudne do realizacji, ponieważ pojmowanie rzeczywistości jest ograniczone przez aparat percepcyjny istot dokonujących jej opisu. Rzeczywistość, którą jest w stanie opisać człowiek jest innym rodzajem rzeczywistości, którą mogłaby opisać mucha, pies lub dowolna inna istota. Trudnym pytaniem jest, czy to, co postrzegamy jesteśmy w stanie uogólnić do obiektywnych stanów rzeczywistości.

W technologiach informatycznych ontologią zaczęto nazywać formalną reprezentację wiedzy, na którą składa się zestaw pojęć oraz relacji między nimi. Taki zapis może służyć do wnioskowaniach o właściwościach opisywanych pojęć. W celu zapisania ludzkiej wiedzy w postaci zrozumiałej dla maszyn, każdy występujący w niej element musi mieć przyporządkowany swój własny unikalny identyfikator, zawsze jednakowo interpretowany. Rolę takiego UID pełni opisany już wcześniej URI, który może odnosić się zarówno do pojęć abstrakcyjnych, jak i konkretnych bytów. Idea stosowania unikalnych identyfikatorów jest podstawą technologii RDF, stanowiącej podstawę budowy tzw. semantycznego Internetu (ang. *Semantic Web*).

8.2.5. Zarządzanie identyfikatorami

W przypadku UID o zasięgu lokalnym, krajowym lub strukturowym, rolę administratorów pełnią instytucje będące właścicielami danej bazy wiedzy. Mogą nimi być organy administracji publicznej, administratorzy forów internetowych, zarządy firm (reprezentowane najczęściej poprzez oddelegowane osoby uprawnione do zarządzania systemem). W przypadku identyfikatorów globalnych (jak URI, SSCC, ISBN) administracją zajmują się międzynarodowe organizacje odpowiadające za przydzielanie stosowanych identyfikatorów. Przykładami takich organizacji są:

- World Wide Web Consortium (W3C) – odpowiada za URI,
- GS1 – odpowiada za SSCC,
- International ISBN Agency – odpowiada za ISBN.

Kwestia odgórnego zarządzania UID ma kluczowe znaczenie przy zachowywaniu ich unikalności, która stanowi podstawę ich wykorzystywania. Brak nadzoru nad postacią identyfikatorów doprowadziłby prawdopodobnie do ich powielania się, co zachwiałoby równowagą całej bazy wiedzy (jeden UID wskazywałby na kilka różnych obiektów).

8.3. Przykładowe metody generowania UID

Algorytmy generowania identyfikatorów unikalnych można podzielić na dwie podstawowe grupy:

1. Generatory automatyczne (komputerowe), które na podstawie skryptu tworzą nowe UID bez potrzeby autoryzacji ze strony człowieka.
2. Generatory proceduralne, oparte na zdefiniowanym formalizmie (najczęściej rozporządzeniu lub tabeli wartości porównawczych), wymagające „ręcznego" przyporządkowania wartości identyfikatorów.

8.3.1. Metody automatyczne

Przykładem identyfikatora generowanego komputerowo jest UUID. Składa się on z 32 heksadecymalnych znaków zorganizowanych w pięć grup oddzielonych myślnikami wg reguły:

$$\underbrace{\texttt{xxxxxxxx}}_{\text{8 znaków}} - \underbrace{\texttt{xxxx}}_{\text{4 znaki}} - \underbrace{\texttt{Mxxx}}_{\text{4 znaki}} - \underbrace{\texttt{Nxxx}}_{\text{4 znaki}} - \underbrace{\texttt{xxxxxxxxxxxx}}_{\text{12 znaków}}$$

co sumarycznie daje 36 znaków. Wyróżnione znaki `N` oraz `M` kodują, odpowiednio, wariant identyfikatora (jego formę) oraz wersję (sposób generowania).

Podczas tworzenia UUID wykorzystywany jest **timestamp** – 60-bitowy ciąg kodujący liczbę 100-nanosekundowych interwałów, które upłynęły od momentu przyjęcia kalendarza gregoriańskiego do chwili generowania identyfikatora (zobacz tab. 8.1). Sekwencja zegara jest wartością zapobiegającą powstawaniu duplikatów identyfikatorów związanych z niemonotonicznością zegara.

Tab. 8.1: Opis ogólny poszczególnych pól UUID.

Pole	Długość	Opis
time_low	8	Najmniej znaczące bity **timestamp**
time_mid	4	Średnio znaczące bity **timestamp**
time_hi_and_version	4	Najbardziej znaczące bity **timestamp** połączone z numerem wersji
clock_seq_hi_and_reserved	2	Najbardziej znaczące bity sekwencji zegara połączone z numerem wariantu
clock_seq_low	2	Najmniej znaczące bity sekwencji zegara
node	12	Unikalny numer węzła

Wyróżnianych jest pięć metod tworzenia UUID:

1. na podstawie adresu MAC (ang. *Media Access Control Address*) – wersja pierwotna, opiera się na połączeniu fragmentów adresu MAC urządzenia generującego identyfikator ze wspomnianą liczbą interwałów czasowych: pole **node** przyjmuje wartość adresu (zazwyczaj adres hosta). Metoda ta jest często krytykowana za możliwość identyfikacji komputera, który utworzył dane UUID oraz momentu jego powstania. Zasadniczą zaletą tej wersji jest deterministyczna unikalność generowanych identyfikatorów.
2. z zastosowaniem DCE Security (ang. *Distributed Computing Environment Security*) – metoda analogiczna do poprzedniej z tą różnicą, że znaczący bajt sekwencji zegara jest zastępowany identyfikatorem domeny lokalnej, a cztery pierwsze bajty **timestamp** – identyfikatorem POSIX UID użytkownika.
3. wersja haszująca MD5 – działa na zasadzie przekształcenia podanego ciągu znaków (URL, nazwy domeny, identyfikatora itd.) oraz identyfikatora UUID przestrzeni nazw (np. `6ba7b810-9dad-11d1-80b4-00c04fd430c8` dla domeny). W tym celu użyte UUID jest przekształcane do postaci ciągu znaków odpowiadającego poprzednim wartościom heksadecymalnym, łączone z wprowadzoną nazwą i haszowane funkcją MD5 przy użyciu 128 bitów. Pod koniec, podmienione zostają znaki wariantu oraz wersji UUID.
4. losowa – wartości poszczególnych znaków są wyznaczane na podstawie generatora liczb pseudolosowych (ustalone są tylko znaki wariantu oraz wersji).
5. haszująca SHA-1 – analogicznie do wersji haszującej MD5, jednak z wykorzystaniem funkcji skrótu SHA-1.

Dzięki schematowi generowania identyfikatorów opartemu na prostych regułach uwzględniających zasoby komputera tworzone są powszechnie wykorzystywane systemy automatycznego wyznaczania UUID.

8.3.2. Metody proceduralne

Przykładem identyfikatorów generowanych proceduralnie są numery porządkowy akt oraz innych dokumentów używanych w administracji publicznej. Reguły ich tworzenia opisano w oficjalnych dokumentach. W dniu 13 marca 2012 r. zaczęło obowiązywać Zarządzenie nr 6 Ministra Spraw Wewnętrznych w spra-

wie instrukcji kancelaryjnej oraz jednolitego rzeczowego wykazu akt Ministerstwa Spraw Wewnętrznych [10]. Zawiera ono m.in. Instrukcję Kancelaryjną (reguły konstruowania identyfikatora sprawy) oraz Wykaz Haseł Klasyfikacyjnych (zestaw identyfikatorów odpowiadającym poszczególnym tematom spraw).

Zgodnie z rozporządzeniem, każde akta ministerstwa zostają opatrzone identyfikatorem sprawy postaci:

$$\underbrace{\text{<symbol kom. organizacyjnej>}}_{\text{literowy lub literowo-cyfrowy}} \text{ - } \underbrace{\text{<nr hasła klasyfikacyjnego>}}_{\text{czterocyfrowy}} \text{ / } \underbrace{\text{<rok rozpoczęcia>}}_{\text{dwie ostatnie cyfry}}$$

Przykładowy ID akt: `AB-IV-0000/06`, gdzie `AB-IV` to symbol komórki organizacyjnej zajmującej się daną sprawą, `0000` – ciąg cyfr kodujący tematykę dokumentów, `06` – dwie ostatnie cyfry roku rozpoczęcia danej sprawy. Taki identyfikator nie jest globalnie unikalny, ale pozwala na efektywną archiwizację i późniejsze odszukiwanie danych.

Pole z numerem hasła klasyfikacyjnego składa się z czterech głównych elementów:

- jednocyfrowego kodu ogólnego działu tematycznego – przyjmuje on wartości od 0 do 9 (bez cyfry 7) i oznacza odpowiednio:
  0 – `Zarządzanie`
  1 – `Kadry`
  2 – `Środki rzeczowe`
  3 – `Finanse`
  4 – `Ochrona bezpieczeństwa państwa`
  5 – `Ochrona bezpieczeństwa obywateli i porządku publicznego`
  6 – `Sprawy społeczne`
  8 – `Kontrola zarządcza`
  9 – `Audyt`
- oraz trzech cyfr specyfikacji, odpowiadających kolejnym poziomom zagnieżdżenia tematu.

Dla przykładu, identyfikator `0493` oznacza:

0 – `Zarządzanie`
  04 – `Informatyka/Informatyzacja`
    049 – `Dokumenty wytworzone z użyciem środków informatyki`
      0493 – `Dokumentacja techniczna`

Nadanie aktom numeru jest jednoznaczne z zapoznaniem się z ich tematyką i dopasowaniem symbolu sprawy zgodnie z tabelą Wykazu Haseł Klasyfikacyjnych.

8.4. Uniwersalny generator unikalnych identyfikatorów

W obliczu powszechnego stosowania technologii UID, nasuwa się następujące pytanie: „Czy generator identyfikatorów może zostać wydzielony z systemu, w którym dane UID są wykorzystywane? Jeżeli tak, to w jakiej formie? Jakie cechy wyróżniają takie rozwiązanie?”. Niniejszy podrozdział zawiera dyskusję autorów na ten temat.

System korzystający z konkretnych rodzajów identyfikatorów musi mieć dostęp do mechanizmu ich tworzenia. W tym celu najczęściej stosowane są generatory zintegrowane – występujące jako części składowe danego systemu. Oczywiście takie rozwiązanie pociąga za sobą konieczność implementacji wybranej procedury generowania UID (podrozdział 8.3). Takie działanie nie tylko zmusza twórców do rozbudowy systemu, ale prowadzi do powstania wielu takich samych (albo bardzo zbliżonych) generatorów UID, niepowiązanych ze sobą. Można więc postawić pytanie: czy można stworzyć jeden mechanizm, wydzielony z systemu i dostępny dla wielu systemów-klientów? Oczywiście jest to możliwe. Co więcej, samodzielny generator nie musi ograniczać się do pojedynczej procedury tworzenia identyfikatorów. Można przecież wydzielić parametry związane z wyznaczaniem postaci konkretnego identyfikatora (zmienne wpływające na generowane wartości oraz strukturę UID) – tworząc w ten sposób interfejs pomiędzy systemem a generatorem identyfikatorów. Od strony technicznej, można go utworzyć w postaci biblioteki (dołączanej do źródeł systemu na zasadzie wtyczki) lub niezależnej aplikacji (z którą system mógłby się komunikować). Poniżej opisano charakter tych rozwiązań:

biblioteka – zestaw funkcji obsługujących dołączone „z zewnątrz" procedury tworzenia UID, umieszczone w oddzielnej bibliotece. Metoda polega na „zintegrowaniu" zewnętrznego generatora z systemem. Zaletą tego rozwiązania jest uniezależnienie się od zewnętrznego serwera generatora. Jest to korzystne rozwiązanie dla systemów obsługujących rozległe bazy wiedzy, wymagających wysokiej niezawodności.

samodzielna aplikacja – generator w postaci oddzielnego programu, z którym system może się komunikować. Wymusza synchronizację pracy pomiędzy klientem a serwerem generatora. Aplikacja może zawierać graficzny interfejs użytkownika, który ułatwia współpracę z klientem (w sytuacji, gdy użytkownikiem programu jest człowiek). Umieszczenie generatora na ogólnodostępnym serwerze pozwala na dostęp dużej grupie klientów zewnętrznych. Dodatkową zaletą tego rozwiązania jest fakt, iż udostępniony generator jest ukończonym, w pełni działającym produktem, którego nie trzeba dodatkowo oprogramowywać (za wyjątkiem podania definicji i procedur tworzenia konkretnego identyfikatora).

Wybór metody zależy od potrzeb docelowej grupy klientów generatora. W przypadku stosowania generatora UID wydzielonego z systemu, szczególnie ważnym jest zagwarantowanie unikalności uzyskanych identyfikatorów. Zasadniczym pytaniem jest: „Który element powinien sprawować tę kontrolę – generator czy klient?". Odpowiedź na to pytanie jest ściśle związana w zastosowaniem konkretnego generatora:

- jeżeli obsługuje on wyłącznie jeden system, możliwe jest zaprojektowanie generatora w sposób gwarantujący unikalność tworzonych identyfikatorów,
- dla generatora „wielodostępowego" – unikalność należy zapewnić po stronie klientów.

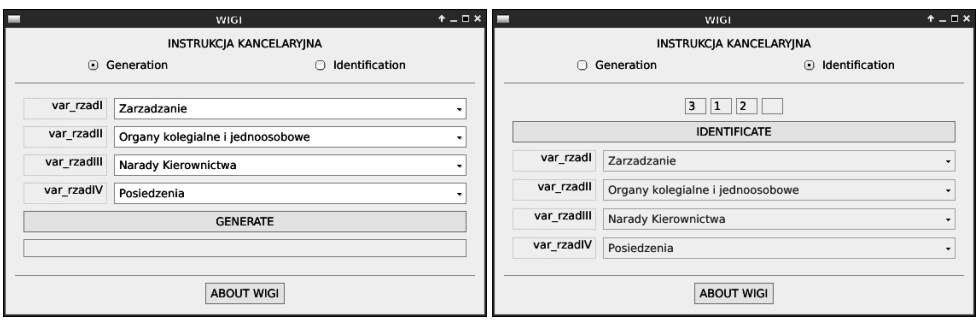
W prawie każdym przypadku jest możliwe wymuszenie kontroli unikalności identyfikatorów po stronie generatora, jednak jest to rozwiązanie nieefektywne, które prowadzi m.in. do wzrostu złożoności procedur tworzenia UID. Oczywiście, istnieją wyjątki związane ze specyfiką generowania konkretnych identyfikatorów tj. UUID (opisanego w sekcji 8.3.1), która samoistnie wymusza ich unikalność.

8.4.1. Przykładowa realizacja

Poniżej opisano autorską realizację uniwersalnego generatora UID w postaci zewnętrznego programu – WIGI (Wszechstronna Identyfikatorów Generacja i Interpretacja). Aplikacja powstała jako element wolnego oprogramowania i jest udostępniona na zasadach licencji GPL v3 [11].

WIGI: Charakterystyka

WIGI to przykład zewnętrznego, uniwersalnego generatora unikalnych identyfikatorów. Powstał w formie samodzielnej aplikacji z interfejsem graficznym z wykorzystaniem bibliotek `Xercess-C++` i `Qt`. Kalibracja programu (wybór typu UID) odbywa się poprzez dostarczenie przez użytkownika pliku `XML` definiującego strukturę identyfikatora oraz procedurę jego generowania (plik taki, o nazwie `id.xml`, umieszczany jest w katalogu źródeł programu). Takie rozwiązanie pozwala definiować identyfikatory dowolnej postaci, dzięki czemu generator jest w pełni elastyczny. Jedynym ograniczeniem jest zgodność załączanego pliku `XML` z zadaną przez autorów składnią (plik `id.xsd`). Ze względu na swój uniwersalizm, WIGI abstrahuje od unikalności tworzonych identyfikatorów – cechę tę nadzoruje klient programu (poprzez plik definicji `XML` lub postać zgłaszanego żądania). Ponadto aplikację wyposażono w dwa tryby pracy: *Generate* (tryb generowania identyfikatora na podstawie danych od użytkownika, domyślny) oraz *Identificate* (tryb identyfikacji danych na podstawie analizy zadanego UID). Wygląd interfejsu użytkownika zależy od wybranego trybu pracy. Na rys. 8.1 przedstawiono przykład ilustrujący pracę programu z identyfikatorami akt MSW generowanymi zgodnie z Instrukcją Kancelaryjną.



Rys. 8.1: Wygląd interfejsu użytkownika WIGI w zależności od trybu pracy: *Generate* (po lewej stronie) oraz *Identificate* (z prawej stronie). W przykładzie generowany/identyfikowany jest numer akt MSW wg Instrukcji Kancelaryjnej.

Elementy składowe systemu generatora UID, to:

Kontenery – komórki składowe identyfikatora, ich szereg tworzy jego postać finalną. Każdy kontener ma ściśle określoną długość (liczbę znaków).

Zmienne wejściowe – parametr mający bezpośredni lub pośredni wpływ na wartość poszczególnych kontenerów identyfikatora. W wielu przypadkach zmiennym można przypisać interpretację fizyczną w realnym świecie (jak data urodzenia lub płeć w przypadku numeru PESEL).

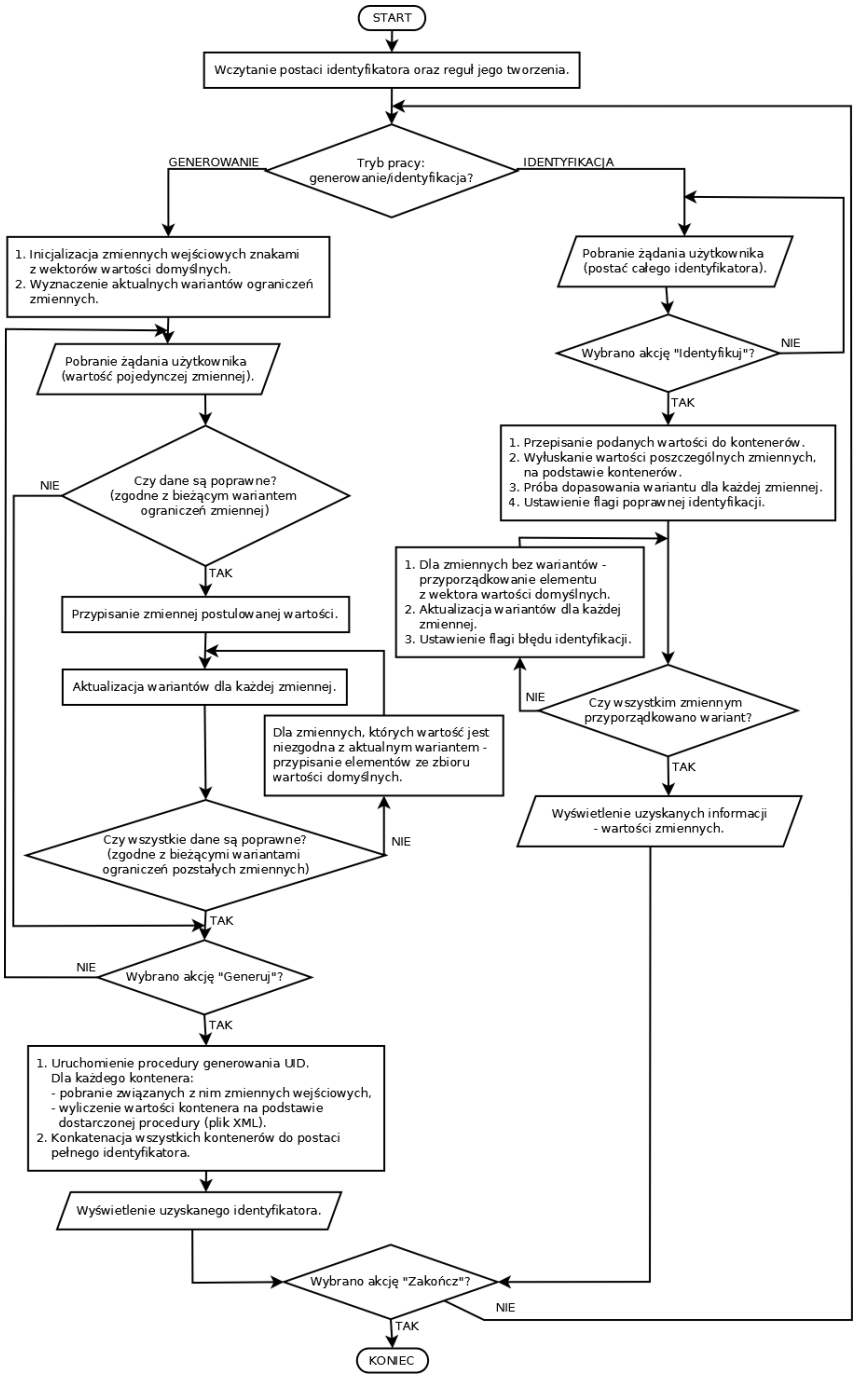
Warianty ograniczeń zmiennych – reguły określające zbiór poprawnych wartości zmiennej. Mogą, choć nie muszą, zależeć od wartości innych zmiennych (np. w przypadku numeru PESEL, dla daty urodzenia ograniczenie na dzień zależy od miesiąca i roku).

Relacje pomiędzy zmiennymi i kontenerami – opisują zależności pomiędzy zawartością poszczególnych kontenerów a wartościami zmiennych.

WIGI: Schemat działania

Schemat działania aplikacji zilustrowano na rys. 8.2. Przy starcie programu wczytywany jest plik `id.xml`, który zawiera informacje o strukturze oraz procedurze tworzenia identyfikatora. Następnie wybierany jest tryb pracy aplikacji – generowanie nowego identyfikatora albo identyfikacja wartości zmiennych na podstawie istniejącego UID. W pierwszym wypadku, użytkownik wprowadza wartości kolejnych zmiennych (w postaci pól lub rozwijalnej listy). Każde wprowadzenie danych powoduje aktualizację wariantów zmiennych, czyli sprawdzenie, czy ich wartość odpowiada aktualnym ograniczeniom. W przypadku błędu, zmiennej przypisywana jest wartość z tzw. wektora wartości domyślnych. Po zakończeniu procedury wprowadzania danych i zgłoszeniu żądania generacji UID (przycisk `GENERATE`), następuje utworzenie identyfikatora. W przypadku pracy w trybie *Identificate*, użytkownik wprowadza gotową postać identyfikatora, który w momencie zgłoszenia żądania identyfikacji (przycisk `IDENTIFICATE`) zostaje poddany procedurze odzyskiwania danych – wartości powiązanych z nim zmiennych.

W przypadku generowania UID nie ma możliwości wprowadzenia niepoprawnych danych – każda błędna wartość zostanie skorygowana na etapie jej podawania. Dzięki temu wysłanie żądania identyfikacji może wystąpić wyłącznie dla poprawnego zestawu danych. Inna sytuacja ma miejsce w przypadku żądania identyfikacji UID. Wartości wprowadzone przez użytkownika są weryfikowane dopiero po rozpoczęciu procedury odzyskiwania danych. Z tego powodu operacja, poza standardowym wyświetleniem wartości zmiennych, zwraca również status poprawności przeprowadzonej identyfikacji – pozytywny, gdy podany identyfikator jest prawidłowy, lub negatywny, w przypadku gdy podany identyfikator nie istnieje (jest niemożliwy do utworzenia w oparciu o reguły zdefiniowane w pliku `id.xml`). Oczywiście, w procesie identyfikacji zakończonej statusem negatywnym, nie ma możliwości odtworzenia wszystkich danych. W takiej sytuacji program koryguje wprowadzony identyfikator do „najbliższej" poprawnej postaci, dla której wyświetlone zostają wartości poszczególnych zmiennych.



Rys. 8.2: Schemat działania algorytmu WIGI.

WIGI: Przykład działania

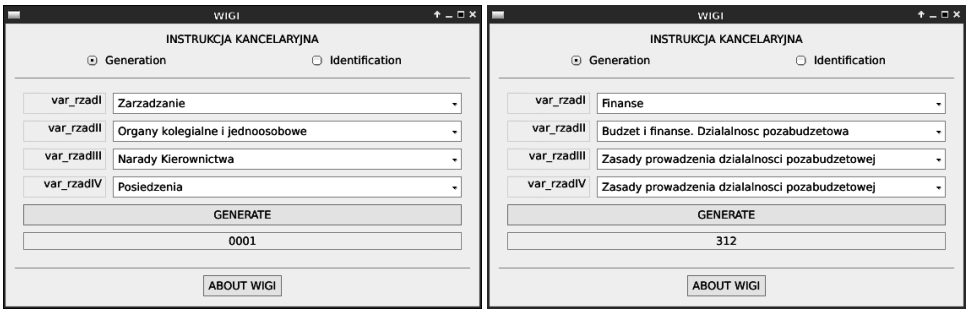
Na potrzeby demonstracji działania programu zaimplementowano schematy generowania i identyfikacji dwóch identyfikatorów: numeru PESEL oraz numeru porządkowego akt Ministerstwa Spraw Wewnętrznych (na podstawie Instrukcji Kancelaryjnej). Poniżej omówiono działanie programu dla numeru akt MSW (jako przypadek bardziej złożony).

Numer porządkowy akt MSW, omówiony w podrozdziale 8.3.2, należy do grupy identyfikatorów generowanych proceduralnie. W celu nadania sprawie identyfikatora należy przyporządkować jej czterocyfrowy numer tematyczny, zgodny z Wykazem Haseł Klasyfikacyjnych zamieszczonym w Instrukcji Kancelaryjnej. Powyższy sposób tworzenia ID jest podatny na błędy oraz mało efektywny. W celu usprawnienia tego procesu, zdefiniowano zestaw reguł opisujących powyższą procedurę — Wykaz Haseł Klasyfikacyjnych przetransformowano do postaci pliku `XML`. Tak zdefiniowany identyfikator składa się z czterech zmiennych odpowiadających odpowiednim poziomom zagnieżdżenia tematyki akt. Podawanie przez użytkownika wartości poszczególnych zmiennych ogranicza się do wyboru odpowiedniego opisu tematyki z umieszczonego obok menu rozwijalnego. Każdy kolejny wybór powoduje aktualizację dostępnych wartości (opisów) zmiennych. Dzięki temu, użytkownik ma do dyspozycji prosty i intuicyjny interfejs zapobiegający popełnianiu błędów przy generowaniu identyfikatora. Analogicznie, chcąc odcyfrować tematykę danych akt wystarczy wpisać do programu konkretny numer sprawy, a poszczególne pola zmiennych zostaną automatycznie uzupełnione w wyniku procesu identyfikacji.

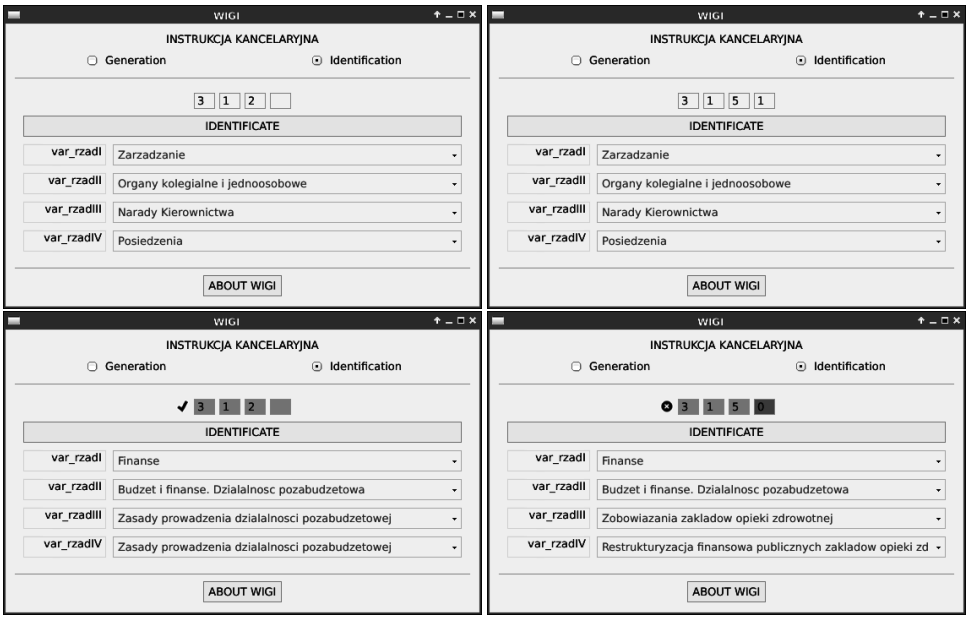
Na rys. 8.3 przedstawiono wygląd interfejsu programu pracującego w domyślnym trybie *Generate*. Widać na nim efekt zgłoszenia żądania wygenerowania identyfikatora na podstawie ustalonych wartości zmiennych dla dwóch różnych zestawów danych wejściowych. Proces interpretacji identyfikatora przedstawiono na rys. 8.4. W trybie tym program generuje dane opisowe na podstawie zadanego identyfikatora (na początek wyświetlane są dane domyślne, które po uruchomieniu identyfikacji zamieniane są wartościami znalezionymi). W lewej kolumnie tego rysunku zamieszczono kolejne okna podczas poprawnie zakończonej interpretacji (znaleziono poprawny identyfikator). W prawej kolumnie tego rysunku przedstawiono sytuację, w której interpretacja zakończyła się niepoprawnie (użytkownik podał niepoprawny identyfikator UID, dlatego program zwrócił wynik odpowiadający UID najbliższemu do zadanego).

8.5. Podsumowanie

Technologie IT należą do najprężniej rozwijających się dziedzin przemysłu i nauki. W ich skład wchodzą metody jednoznacznej identyfikacji zasobów. Leżą one u podstaw budowanych obecnie systemów informatycznych. Waga identyfikatorów, zainteresowanie jakie wzbudzają oraz podejmowane próby udoskonalania istniejących rozwiązań stały się motywacją do napisania niniejszego rozdziału. Przedstawione opracowanie miało na celu przybliżenie czytelnikowi te-



Rys. 8.3: Przykład generowania identyfikatora dla dwóch różnych akt MSW.



Rys. 8.4: Proces interpretacji identyfikatora: poprawny (lewa kolumna) oraz zakończony niepowodzeniem (prawa kolumna, program zwraca „najbliższy” poprawny UID).

matyki generowania unikalnych identyfikatorów, jak również zainteresowanie go autorskim rozwiązaniem generatora UID.

Powstała aplikacja pozwala na wydzielenie generatora UID poza właściwy system zarządzania bazą wiedzy. Zaprezentowany pomysł wydzielenia generatora nie doczekał się jeszcze rozwiązania na skalę państwową, nie mówiąc już o międzynarodowej. Autorzy mają nadzieję, że stworzone rozwiązanie okaże się pierwszym, ale nie ostatnim etapem na drodze prowadzącej do uniwersalizacji generatorów.

Literatura

[1] *Wielka Encyklopedia PWN, Tom 11*. Wydawnictwo Naukowe PWN, wydanie pierwsze, 2002. ISBN 83-01-13735-5.

[2] Instrukcja Techniczna O-4: Zasady prowadzenia państwowego zasobu geodezyjnego i kartograficznego, Warszawa - 1987 r.

[3] B. Green, M. Bide. Unique Identifiers: a brief introduction, February 1999. http://www.bic.org.uk/files/pdfs/uniquid.pdf.

[4] Biblioteka Narodowa. Zasady nadawania ISBN. Na podstawie: *The International Standard ISO 2108:2005* oraz *ISBN User's Manual, International edition*, Fifth ed. 2005, http://www.bn.org.pl/programy-i-uslugi/isbn/zasady-nadawania-isbn.

[5] Urząd Publikacji Unii Europejskiej. Międzyinstytucjonalny przewodnik redakcyjny: Identyfikatory, Marzec 2010. http://publications.europa.eu/code/pl/pl-240400.htm#i441.

[6] L. M. T. Berners-Lee, R. Fielding. Uniform Resource Identifier (URI): Generic Syntax, January 2005. http://tools.ietf.org/pdf/rfc3986.pdf.

[7] T. Berners-Lee. Uniform Resource Locators (URL), March 1994. http://www.w3.org/Addressing/URL/url-spec.txt.

[8] I. K. R. Petke. Registration Procedures for URL Scheme Names, November 1999. http://www.ietf.org/rfc/rfc2717.txt.

[9] D. Z. L. Masinter, H. Alvestrand, R. Petke. Guidelines for new URL Schemes, November 1999. http://www.ietf.org/rfc/rfc2718.txt.

[10] P. Stachańczyk. Instrukcja Kancelaryjna Ministerstwa Spraw Wewnętrznych, z dnia 27 grudnia 2011 r.

[11] GNU General Public License, June 2007. http://www.gnu.org/licenses/gpl.html.

Od redaktora i wydawcy

Rozwój ludzkiej cywilizacji na przestrzeni dziejów charakteryzował się różną dynamiką. Występowały w nim okresy względnej stabilności, przeplatane przedziałami spowolnienia i przyspieszenia. Jednak to, co nastąpiło ostatnimi laty nie zdarzyło się nigdy wcześniej. W codziennym życiu ludzi na dobre zagościły komputery, które dzięki zdolnościom do przetwarzania ogromnych ilości danych stały się motorem postępu. Ich obecność jest zauważalna niemal w każdym miejscu – od gospodarstw domowych po hale przemysłowe i urządzenia w przestrzeni publicznej. W laboratoriach naukowych nieustannie trwają prace nad uczynieniem z komputerów maszyn inteligentnych – rozumiejących potrzeby swoich użytkowników, wspierających ich w wykonywaniu obowiązków i różnorakich czynności. Powstające maszyny przestały być dziełem jednego tylko twórcy. Obecnie są one wynikiem prac całych zespołów i oferują funkcje będące wypadkową wielu wykształconych w nich cech. Ma to swoje reperkusje. W futurystycznych opowiadaniach często można spotkać opisy różnych relacji człowiek-komputer. Dokonując ich projekcji na płaszczyznę teraźniejszości dwie z nich, oparte na „wzajemnym zrozumieniu", stają się bardzo ciekawe. Wszelkie próby stworzenia sztucznej inteligencji polegały na nauczeniu komputerów rozumowania w ludzki sposób. Czy jednak nie jesteśmy w przededniu epoki, w której to ludzie będę musieli nauczyć się sposobu rozumowania komputerów? Prowadzenie podobnych analiz stało się elementem filozoficznych rozważań i akademickich kursów.

W niniejszej książce zebrano opracowania studentów II roku studiów magisterskich Wydziału Elektroniki Politechniki Wrocławskiej, kierunku Automatyka i robotyka, specjalności Robotyka, wykonane w semestrze letnim roku akademickiego 2012/2013 podczas realizacji prowadzonego przeze mnie kursu Komputerowe przetwarzanie wiedzy. W opracowaniach tych dokonano syntetycznego przeglądu wybranych aspektów przetwarzania danych i informacji oraz pozyskiwania wiedzy. Przedstawiono w nich również autorskie rozwiązania konkretnych problemów. Zakres tematyczny opracowań można zawrzeć w następującym zestawieniu:

- **Przetwarzanie tekstów i budowa słowników**
- **Ocena sprawności wyszukiwania informacji**
- **Automatyczne generowanie rekomendacji**
- **Wykorzystanie grafowej i dokumentowej bazy danych**
- **Systemy decyzyjne**
- **Tworzenie i wykorzystanie unikalnych identyfikatorów**

Mam nadzieję, że lektura tych opracowań okaże się interesująca dla czytelnika.

Tomasz Kubik
Wrocław, styczeń 2014

ISBN 978-83-930823-4-6
9 788393 082346