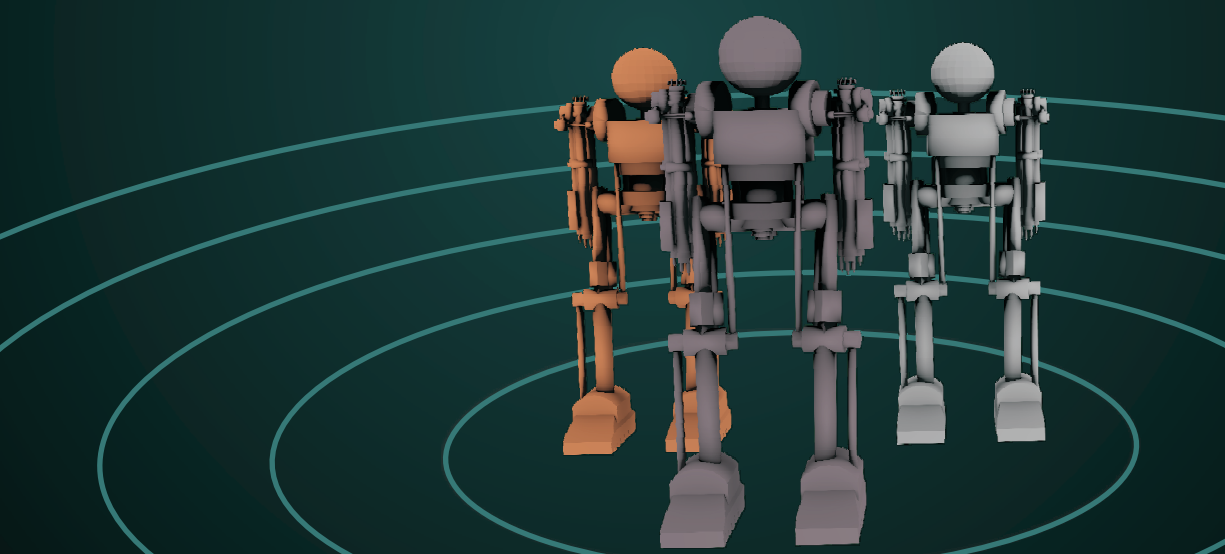
KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2011/2012
pod redakcją Tomasza Kubika



KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2011/2012
pod redakcją Tomasza Kubika

Skład komputerowy, projekt okładki
Tomasz Kubik

ISBN 978-83-930823-3-9

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

Kolejna książka *Komputerowe przetwarzanie wiedzy, Kolekcja prac 2011/2012* dokumentuje projekty wykonane przez studentów II roku 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 letnim roku akademickiego 2011/2012. Projekt miał na celu przeanalizowanie wybranych metod i narzędzi do przetwarzania wiedzy oraz ich implementację w postaci programów komputerowych służących do rozwiązania konkretnych zadań. Niniejsza *Kolekcja* składa się z dziewięciu prac, które dotyczą następujących obszarów tematycznych:

- Modelowanie
- Przetwarzanie języka naturalnego
- Walidacja danych i modeli
- Ontologie
- Analiza danych sensorycznych
- Detekcja tekstu
- Drzewa decyzyjne

Przyjmując zasadę *pars pro toto*, jako ilustrację treści *Kolekcji* wymienimy cztery przykładowe projekty:

- A. Ciż, M. Gulanowski, P. Wybieralski, *Sztuczna komórka – modelowanie procesów biologicznych na poziomie komórkowym.* Autorzy projektu opisali dostępne środowiska programowe przeznaczone do modelowania zjawisk zachodzących w komórkach organizmów żywych. Szczególną uwagę poświęcono środowisku VCell (virtual cell), którego możliwości zilustrowno na przykładzie zadania *pozyskiwania i przetwarzania energii w komórce roślinnej.*
- M. Frontkiewicz, P. Kaczmarek, M. Pochna, *Modelowanie tłumu i paniki.* Opis projektu przedstawia podstawowe pojęcia charakteryzujące genezą zjawiska paniki w tłumie. Projekt koncentruje się na modelowaniu paniki powstającej podczas ewakuacji. Opierając się na dostępnych narzędziach programowych, do symulacji zjawiska paniki opracowano aplikację typu agentowego. Działanie aplikacji zilustrowano symulacjami komputerowymi.

- M. Kot, M. Ogorzelec, M. Paluch, *Walidacja modeli konceptualnych zapisanych w UML*. Autorzy projektu zajęli się zagadnieniem walidacji modeli systemów mającej na celu sprawdzenie poprawności składniowej modelu systemu i jego zgodności ze specyfikacją. Przedstawiono wybrane pojęcia dotyczące specyfikacji i walidacji modeli. Jako przykład ilustracyjny przeprowadzono walidację prostego modelu konceptualnego z dziedziny robotyki.
- B. Kułak, M. Wcisło, Ł. Żygadło, *Projekt i zastosowanie prostej ontologii*. Opis projektu zawiera objaśnienie pojęcia ontologii używanego w inżynierii wiedzy i przegląd algorytmów wnioskowania w ramach ontologii. Omówiono elementy języka OWL (*Web Ontology Language*) pozwalającego na zapis i wnioskowanie w ramach ontologii. Rozważania teoretyczne zilustrowano przykładem ontologii, która umożliwia utworzenie bazy wiedzy o tematyce żywnościowej.

Dzięki przyjętej koncepcji i starannej redakcji tekstów opisy zadań projektowych przedstawione w *Kolekcji* stanowią spójną i zamkniętą całość. Można oczekiwać, że podobnie jak jej poprzedniczki wydane w ubiegłym roku, niniejsza książka stanie się pożytecznym źródłem informacji dla Czytelników pragnących uzyskać podstawowe rozeznanie w zadaniach, metodach i możliwościach komputerowego przetwarzania wiedzy.

Prof. Krzysztof Tchoń,
opiekun specjalności Robotyka,
Wrocław, wrzesień 2012

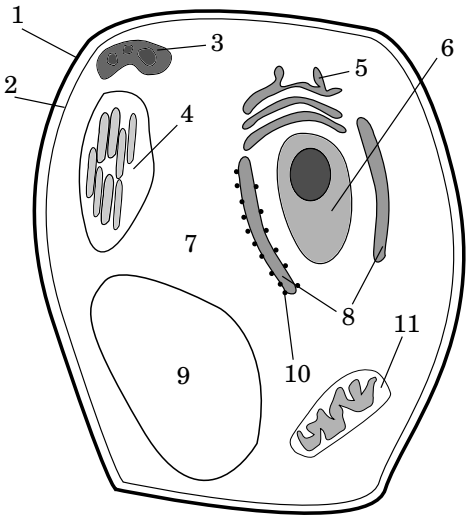
ROZDZIAŁ 1

SZTUCZNA KOMÓRKA – MODELOWANIE PROCESÓW BIOLOGICZNYCH NA POZIOMIE KOMÓRKOWYM

A. Ciż, M. Gulanowski, P. Wybieralski

Systemy komputerowe umożliwiające przeprowadzanie badań symulacyjnych są wykorzystywane nie tylko jako narzędzia do prac inżynierskich. Znajdują one zastosowanie w wielu innych dziedzinach, na przykład w bioinformatyce, gdzie służą do modelowania procesów zachodzących w układach biologicznych. Stworzone modele i wyniki przeprowadzonych z ich wykorzystaniem symulacji porównywane są z wynikami rzeczywistych eksperymentów. Jeżeli wynik uzyskany symulacyjnie jest zgodny z wynikiem rzeczywistym, model może być stosowany do przeprowadzania eksperymentów metodą wyłącznie symulacyjną [1]. Wyniki symulacji pozwalają również zaplanować doświadczenia wykonywane na rzeczywistych układach.

Ponieważ podstawową jednostką strukturalną i funkcjonalną układów biologicznych jest komórka, modelowanie często dotyczy zjawisk odbywających się na jej poziomie. W niniejszym rozdziale podjęto próbę przedstawienia systemu służącego do symulacji procesów biologicznych zachodzących właśnie na poziomie komórkowym. Na początku rozdziału umieszczono krótkie wprowadzenie w zagadnienia teoretyczne (podrozdział 1.1), przytaczając podstawowe informacje dotyczące budowy i funkcjonowania komórki. Następnie w podrozdziale 1.2 dokonano przeglądu istniejących systemów do symulacji procesów zachodzących w układach biologicznych na poziomie komórkowym. Do dalszej analizy wybrano system VCell. Motywy tej decyzji oraz architektura i funkcjonalność tego programu opisano w podrozdziale 1.3. Kolejne podrozdziały (1.4 i 1.5) poświęcono na przeprowadzenie czytelnika przez proces badań symulacyjnych z wykorzystaniem systemu VCell, przedstawiając kolejno sposób określenia danych wejściowych, realizację samej symulacji, wreszcie analizę danych wyjściowych. Dopełnieniem całego opracowania jest podrozdział 1.6, w którym opisano funkcjonalność systemu VCell z punktu widzenia zarządzania wiedzą.



Rys. 1.1: Schemat komórki roślinnej: 1 – ściana komórkowa, 2 – błona komórkowa, 3 – lizosom, 4 – chloroplast, 5 – aparat Golgiego, 6 – jądro komórkowe, 7 – cytozol, 8 – retikulum endoplazmatyczne, 9 – wakuola, 10 – rybosom, 11 – mitochondrium.

1.1. Struktura komórki

W niniejszym opracowaniu skupiono się na modelu komórki organizmu eukariotycznego, tzn. komórki, która posiada jądro komórkowe (w odróżnieniu od organizmów prokariotycznych – bezjądrowych, czyli np. bakterii). Dlatego opisano struktury występujące w takiej komórce i ich ogólnie pojętą rolę w procesach życiowych, korzystając przy tym z podręcznika [2].

Komórka organizmu eukariotycznego składa się z szeregu organelli zawieszonych w cytozolu. Procesy komórkowe zachodzą zarówno w cytozolu, jak i wewnątrz organelli. Na rysunku 1.1 przedstawiono uproszczony schemat komórki roślinnej oraz wymieniono jej podstawowe organella. Komórka roślinna odróżnia się od komórki zwierzęcej m.in. obecnością ściany komórkowej, chloroplastów i wakuoli. Poniżej zamieszczono krótki opis funkcji pełnionych przez wymienione organella.

Jądro komórkowe – zawiera informację genetyczną komórki w postaci cząsteczek DNA, które przed podziałem komórki kondensuje się do postaci chromosomów.

Chloroplasty – wykorzystują barwnik zwany chlorofilem do pobierania energii bezpośrednio ze światła słonecznego, tj. do prowadzenia fotosyntezy, czyli wytwarzania wysokoenergetycznych cząsteczek cukru z wody oraz dwutlenku węgla.

Mitochondria – prowadzą procesy utleniania cząsteczek pokarmu, przez co wytwarzają energię chemiczną potrzebną komórce w postaci cząsteczek ATP (adenozynotrifosforanu). Proces ten nazywa się oddychaniem komórkowym, gdyż substratem tej reakcji jest tlen, a produktem – woda i dwutlenek węgla.

Rybosomy – są *maszynami molekularnymi*, których zadaniem jest synteza białek.

Lizosomy – dostarczają środowiska, w którym zachodzą procesy rozkładu różnych substancji.

Retikulum endoplazmatyczne – służy do wytwarzania składników potrzebnych do budowy błon występujących w komórce oraz produkcji substancji wydzielanych przez komórkę na zewnątrz.

Aparat Golgiego – pełni funkcje związane z modyfikacją cząsteczek oraz kierowaniem ich we właściwe miejsce w komórce lub na zewnątrz.

Wakuola – stanowi miejsce przechowywania np. substancji zapasowych.

Błona komórkowa – oddziela komórkę od środowiska zewnętrznego, zawiera struktury kontrolujące wymianę substancji z otoczeniem.

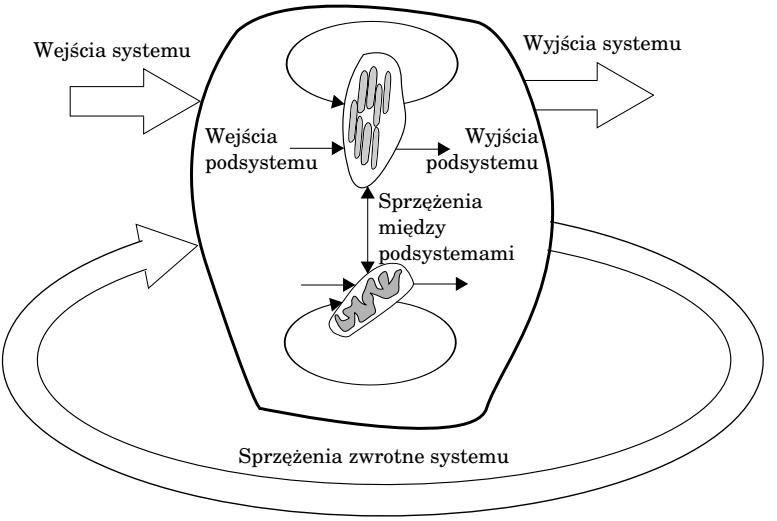
Ściana komórkowa – stanowi wzmocnienie komórki i utrwala jej kształt.

Cytozol – można określić jako *wolną przestrzeń* w komórce, tj. tę, która nie zawiera się wewnątrz błon oddzielających poszczególne organelle. Przestrzeń ta jest wypełniona bardzo gęstym żelem zawierających liczne substancje. To właśnie w cytozolu zachodzi proces syntezy białek.

W świetle powyższych informacji możliwa jest próba opisu komórki w ujęciu cybernetycznym. Schemat takiego układu przedstawiono na rysunku 1.2. Komórkę można traktować jako złożony system typu MIMO [1], objęty pętlą sprzężenia zwrotnego od stanu. Każde organellum komórkowe samo w sobie także stanowi tego rodzaju układ, choć na mniejszą skalę (m.in. taka obserwacja leży u źródeł teorii endosymbiozy, która stanowi, że mitochondria mogły pierwotnie być autonomicznymi organizmami, które zostały wchłonięte przez komórkę). Rolę sygnałów przesyłanych między organellami oraz komórkami stanowią wyspecjalizowane substancje (cząsteczki), głównie białka. Cząsteczki te mogą być wydzielane przez organelle do cytozolu, a także przez komórkę do środowiska zewnętrznego. Mechanizmy ich działania mogą być różnorodne: cząsteczki sygnałowe mogą pełnić funkcję aktywatorów (substancji zwiększających aktywność) lub inhibitorów (substancji zmniejszających aktywność) kompleksów pełniących określone funkcje.

Określenie wektora zmiennych stanu komórki jest niezwykle trudne ze względu na złożoność tego układu. Dlatego też próba symulacji zachodzących

[1] *M*ulti-input multi-output – system o wielu wejściach i wielu wyjściach.



Rys. 1.2: Przykładowy schemat komórki w ujęciu cybernetycznym.

w komórce procesów musi opierać się na wyborze zjawisk istotnych w konkretnych badaniach i ograniczenia modelowania komórki do tych zjawisk. Na rysunku 1.2 przedstawiono właśnie taki przykład, gdzie ograniczono liczbę organelli, pozostawiając jedynie chloroplast oraz mitochondrium. Taka konfiguracja może być właściwa w modelowaniu gospodarki energetycznej komórki: procesu pozyskiwania energii (chloroplast) oraz jej zużywania (mitochondrium).

1.2. Przegląd dostępnego oprogramowania

Istnieje szereg środowisk symulacyjnych, które umożliwiają modelowanie procesów komórkowych – każde z tych środowisk obejmuje inny zakres zjawisk i posiada swoje specyficzne cechy [1]. Poniżej omówiono kilka wybranych programów symulacyjnych.

VCell (`www.nrcam.uchc.edu/`) – środowisko stworzone w celu umożliwienia przewidywania wyników eksperymentów biologicznych na poziomie komórkowym. Wykorzystywane jest głównie do modelowania procesów rozprzestrzeniania się substancji w cytoplazmie i przenikania przez błony komórkowe, np. dynamiki jonów wapniowych w neuronach lub składania i przekazywania sekwencji RNA.

E-Cell (`www.e-cell.org`) – służy do modelowania procesów związanych z reakcjami metabolicznymi np. gospodarki energetycznej bakterii *Escherichia coli*.

MCell (`www.mcell.cnl.salk.edu/`) – modeluje reakcje pomiędzy dużymi, trójwymiarowymi cząsteczkami (mikrofizjologia), np. transmisja synaptyczna.

StochSim (`www.pdn.cam.ac.uk/groups/comp-cell/StochSim.html`) – skoncentrowany na stochastycznych modelach reakcji chemicznych, wykorzystywany np. do symulowania chemotaksji (reakcji ruchowej na kierunkowe bodźce chemiczne) komórki bakteryjnej.

1.3. Architektura i funkcje systemu VCell

Pełna nazwa systemu `VCell` to `Virtual Cell` (w literaturze używane są zamiennie). Jest to *de facto* program służący do numerycznego rozwiązywania równań różniczkowych, zatem teoretycznie może być użyty do symulowania dowolnych układów opisywanych w ten sposób. W zamierzeniu twórców program ten miał być jednak stosowany do symulowania procesów biologicznych zachodzących na poziomie komórkowym. W trakcie symulacji obliczane są stężenia (a ściślej: przestrzenne rozkłady stężeń) każdej substancji (zarówno substratów, jak i produktów) w każdej chwili $t \in (0, t_{max})$.

Wprowadzenie modelu komórki w programie `VCell` polega na:

1. Określeniu generalnej architektury komórki, tzn. wydzieleniu obszarów oddzielonych błonami, określeniu ich wzajemnych relacji (zawierania się) oraz substancji w nich występujących.
2. Zdefiniowaniu zachodzących we wnętrzu komórki przemian, polegających na:
   a) przenikaniu substancji przez błony,
   b) zachodzeniu reakcji chemicznych przekształcających pewne substancje (nazywane substratami reakcji) w inne (nazywana produktami reakcji).
3. Opcjonalnie, na określeniu geometrii komórki, tzn. wielkości i kształtu samej komórki oraz występujących w jej wnętrzu organelli otoczonych błonami. Prowadzi to do powstania wyróżnionych obszarów wewnątrz i na zewnątrz komórki, i pozwala na uwzględnienie wpływu dyfuzji substancji na przebieg reakcji, a także na obserwowanie wynikowych stężeń w różnych partiach komórki.
4. Podaniu początkowych wartości stężeń substancji w każdym wyróżnionym obszarze.

Program `VCell` składa się z następujących elementów:

1. Interfejsu graficznego, umożliwiającego wprowadzenie modelu symulowanych zjawisk, wykonanie symulacji oraz analizę jej wyników.
2. Silnika symulacyjnego, który podzielony jest na dwie części: uruchamianą lokalnie oraz centralną.
   a) Część uruchamiana lokalnie służy do symulowania układów, które mogą być przedstawione w postaci równań różniczkowych zwyczajnych, a zatem takich, w których nie jest brane pod uwagę przestrzenne rozmieszczenie substratów i produktów reakcji.

   b) Część centralna silnika, uruchamiana na komputerze typu mainframe udostępnionym przez Center for Cell Analysis and Modeling (CCAM), pozwala na symulowanie układów, w których rozpatrywany jest rozkład przestrzenny modelowanych zjawisk (są one więc opisane przez równania różniczkowe cząstkowe).

3. Systemu wymiany danych symulacyjnych, umożliwiającego przechowywanie modeli symulacyjnych oraz wyników obliczeń na zdalnym serwerze, udostępnianie ich innym użytkownikom systemu, dostęp do symulacji innych użytkowników (oznaczonych jako publiczne), a także do przykładów instruktażowych.

System VCell posiada podwójny interfejs, służący do definiowania symulowanych modeli. Pierwszy rodzaj (`BioModel`) jest przeznaczony dla biologów, nieposiadających doświadczenia W matematycznym modelowaniu procesów dynamicznych. Dostarcza on graficznych i intuicyjnych metod określania topologii zjawisk, geometrii błon oraz rodzajów i parametrów reakcji chemicznych. Drugi rodzaj interfejsu (`Math`) umożliwia wprowadzenie matematycznego modelu procesu z wykorzystaniem dedykowanego języka `VCMDL`.

1.4. Symulacja przykładowego systemu

Jako proces do modelowania wybrano *pozyskiwanie i przetwarzanie energii w komórce roślinnej*[2]. Jako że celem projektu nie było wykonanie badań z dziedziny biologii komórki, a jedynie zaprezentowanie środowiska służącego do symulowania procesów i użytecznego z punktu widzenia komputerowego przetwarzanie wiedzy, dokonano dużych uproszczeń. Aby móc przeprowadzić symulacje tego procesu, należało wybrać, jakie elementy biologii komórki brane będą pod uwagę, a jakie zostaną pominięte. Zgodnie z podejściem zaprezentowanym powyżej wyróżniono wymienione niżej podprocesy.

1. Przenikanie cząsteczek CO_2 (dwutlenku węgla) ze środowiska otaczającego komórkę do chloroplastów wewnątrz komórki.
2. Proces fotosyntezy, w którym wytwarzana jest glukoza.
3. Przenikanie cząsteczek glukozy do mitochondriów.
4. Oddychanie komórkowe, które zachodzi w mitochondriach.

Poniżej omówiono teoretyczne aspekty dwóch głównych procesów: fotosyntezy oraz oddychania komórkowego.

1.4.1. Aspekty teoretyczne

Fotosynteza – służy przetwarzaniu energii słonecznej na zgromadzoną w postaci energii wiązań chemicznych glukozy (która jest cukrem prostym). Przebiega

[2] Projekt jest dostępny publicznie w programie `VCell` pod nazwą *Fotosynteza* w katalogu użytkownika *serafac*.

zgodnie z równaniem [3]:

$$6\,H_2O + 6\,CO_2 + \text{energia świetlna} \longrightarrow C_6H_{12}O_6 + 6\,O_2$$

Zdolność prowadzenia fotosyntezy jest specyficzna dla roślin i wymaga posiadania w komórkach chloroplastów, w których zachodzi powyższa reakcja. Zanim jednak do tego dojdzie, dwutlenek węgla musi zostać dostarczony do chloroplastu.

Oddychanie komórkowe – służy pozyskiwaniu energii użytecznej biologicznie w postaci adenozynotrójfosforanu (ATP), posiadającego wysokoenergetyczne wiązania. Energia ta pozyskiwana jest w procesie rozkładu glukozy. Reakcja przebiega następująco [3]:

$$C_6H_{12}O_6 + 6\,O_2 + 36\,ADP + 36\,Pi \longrightarrow 6\,CO_2 + 6\,H_2O + 36\,ATP,$$

gdzie ADP – adenozynodifosforan, do którego zostaje przyłączona reszta fosforanowa Pi.

1.4.2. Sposób wykonania symulacji

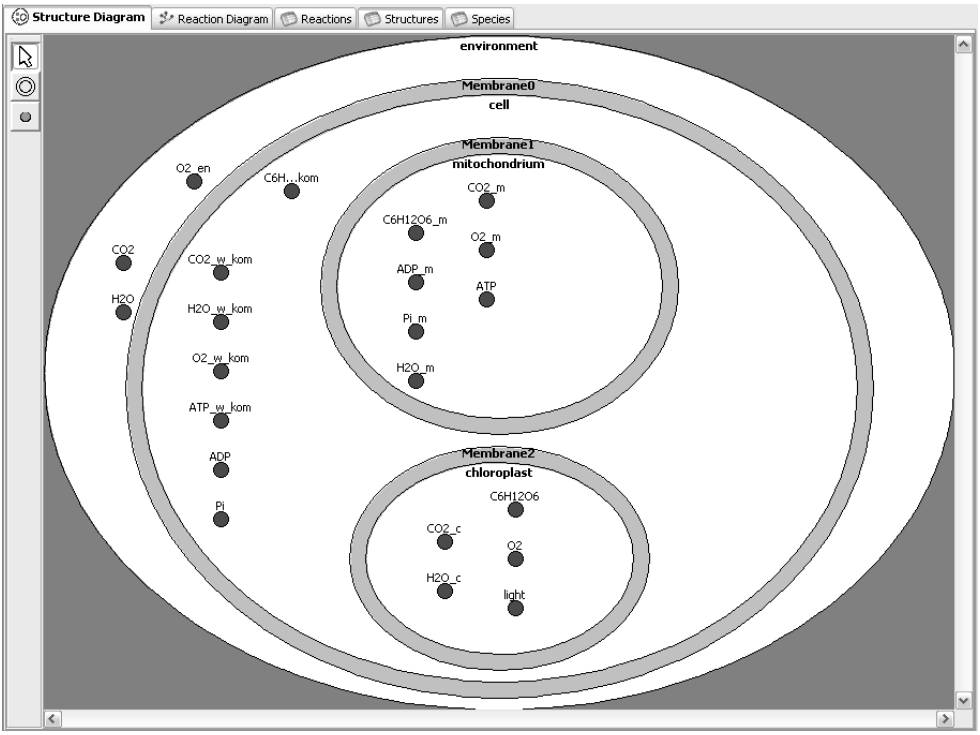
Przeprowadzenie symulacji wymagało wykonania kroków opisanych w podrozdziale 1.3. Zdefiniowano geometrię komórki, posiadającej następujące obszary:

- obszar zewnętrzny,
- wnętrze komórki (cytozol),
- chloroplast,
- mitochondrium.

Na rysunku 1.3 przedstawiono fragment zrzutu ekranu z programu `VCell`, na którym widoczna jest zdefiniowana geometria komórki. Na schemacie struktury komórki umieszcza się nie tylko błony oddzielające poszczególne obszary, ale także symbole wszystkich substancji w każdym obszarze, w którym one występują. Istotnym wymaganiem jest, aby w każdym obszarze dana substancja była oznaczona innym symbolem (informacja, że te różne symbole oznaczają de facto tę samą substancje, zostanie wprowadzona w kolejnym etapie). I tak na przykład tlen został oznaczony symbolami: `O2_en` (w środowisku zewnętrznym), `O2_w_kom` (w cytozolu), `O2` (wewnątrz chloroplastu) oraz `O2_m` (wewnątrz mitochondrium).

Kolejnym etapem jest wprowadzenie przemian, które zachodzą w ramach modelowanego procesu. Jak podano w podrozdziale 1.3, przemiany te obejmują zarówno przenikanie substancji przez błony biologiczne, jak i reakcje chemiczne zachodzące między substancjami. Na rysunku 1.4 przedstawiono diagram przemian uwzględniający wymienione poniżej zjawiska.

1. Przenikanie
    a) CO_2, O_2 i wody ze środowiska zewnętrznego do wnętrza komórki,



Rys. 1.3: Geometria symulowanej komórki.

b) CO_2 i wody z cytozolu do chloroplastu,
c) O_2 i glukozy z chloroplastu do cytozolu,
d) O_2, glukozy, ADP i Pi z cytozolu do mitochondrium,
e) CO_2, wody i ATP z mitochondrium do cytozolu.

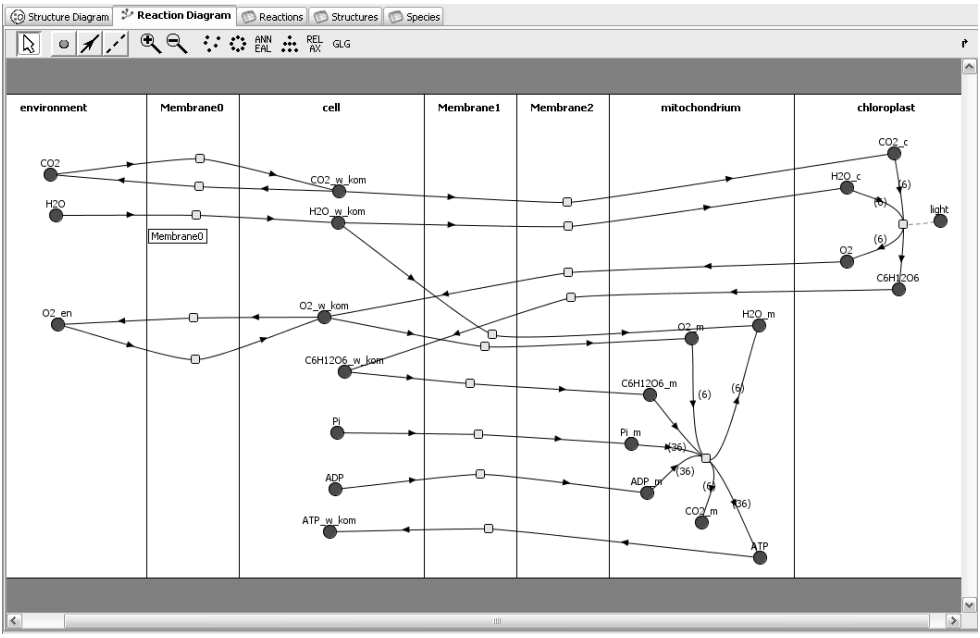
2. Reakcje (podane w podrozdziale 1.4.1)

a) fotosyntezy – zachodzącą w chloroplaście,
b) oddychania komórkowego – zachodzącą w mitochondrium.

1.5. Analiza otrzymanych wyników

Wynikiem wykonanych symulacji są wartości stężeń poszczególnych substancji w całej powierzchni komórki w poszczególnych chwilach. Program `VCell` umożliwia ich wizualizację – wartość stężenia oznaczana jest wówczas za pomocą kolorów. Na rysunkach 1.5 i 1.6 przedstawiono wizualizacje wyników symulacji odpowiednio dla glukozy oraz ATP. W każdym wierszu umieszczono rozkład stężeń danej substancji dla innego obszaru komórki, natomiast kolejne kolumny odpowiadają chwilom: $t = 0$, $t = 1{,}5$s, $t = 2{,}9$s.

Wartości danych wyjściowych uzyskane w wyniku symulacji można także uzyskać w następujących postaciach:



Rys. 1.4: Diagram przemian w komórce.

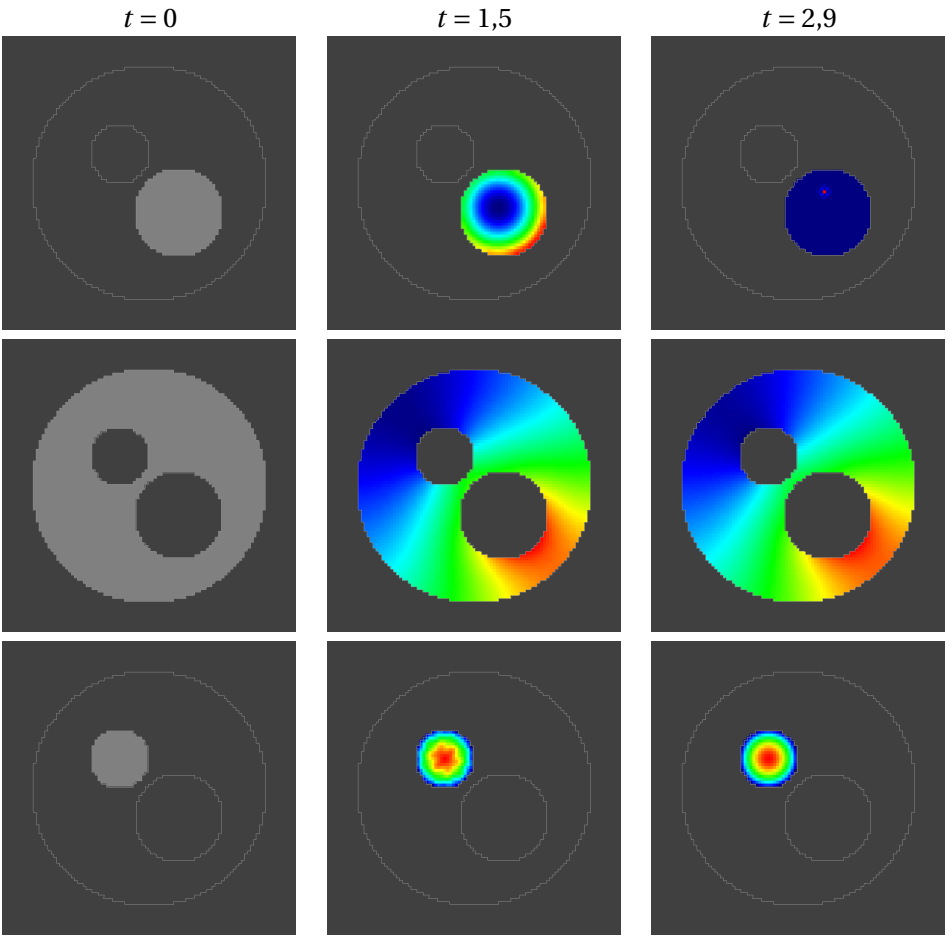
1. Dane liczbowe w formacie CSV (ang. *Comma-separated values* – wartości oddzielone przecinkami).
2. Wykresy ukazujące zmiany stężenia wybranej substancji w danym miejscu komórki.
3. Wykresy ukazujące statystyczne parametry wartości stężeń danej substancji dla wybranego obszaru komórki (np. cytozol, mitochondrium itp.). Przykładowy wykres tego rodzaju przedstawiono na rysunku 1.7.
4. Animacje obrazujące rozprzestrzenianie się wybranej substancji w danym obszarze (w formacie `.mov` lub `.gif`).

1.6. Zarządzanie wiedzą w systemie VCell

Zarządzanie wiedzą w ujęciu programu `VCell` można podzielić na dwie części – część opisującą możliwości współpracy z dedykowaną bazą danych oraz możliwości współpracy z innymi bazami i część opisującą sposób zapisywania gotowych modeli i możliwości eksportowania go do innych standardów.

1.6.1. Współpraca z bazami danych

System `VCell` opiera swoje działanie na zewnętrznej, relacyjnej bazie danych, która nie tylko udostępniona jest dla każdego użytkownika, ale architektura systemu wymaga korzystania z niej. Każdy gotowy model, geometria oraz komponenty są zapisywane na zewnętrznym serwerze, przed każdorazowym urucho-

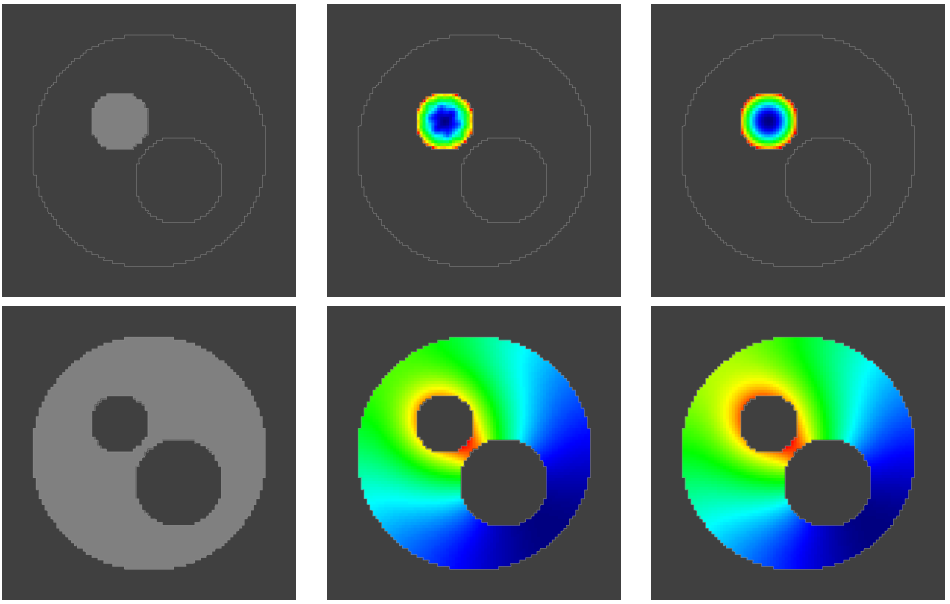


Rys. 1.5: Przemiany glukozy. Odpowiednio: produkcja w chloroplaście, rozprzestrzenianie się w cytozolu, konsumpcja w mitochondrium.

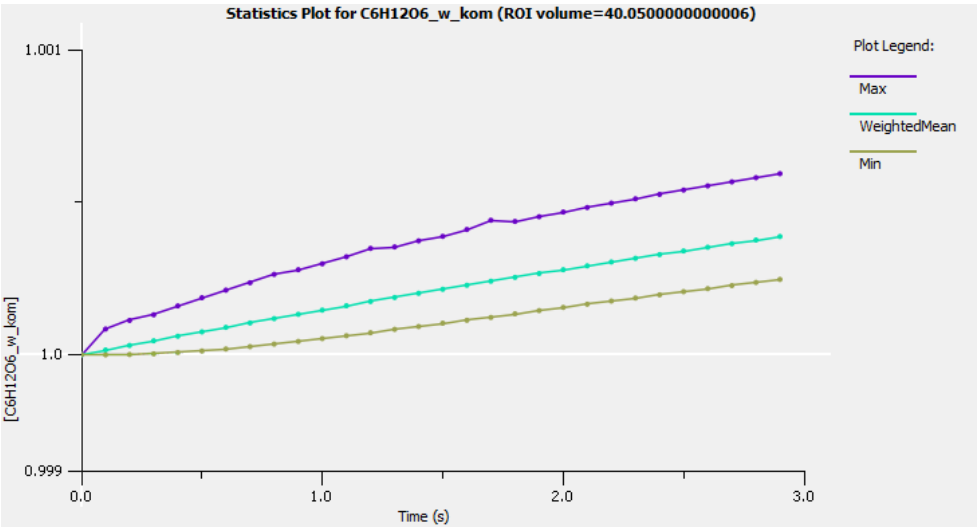
mieniem symulacji, jak również same symulacje są obliczane na maszynie zewnętrznej. Ma to szereg zalet:

- użytkownik nie musi posiadać potężnego komputera, aby wykonać skomplikowane obliczenia w rozsądnym czasie;
- wyniki symulacji są dostępne z każdego komputera posiadającego dostęp do internetu;
- gotowy model jest przechowywany na serwerze, skąd można go pobrać w dowolne miejsce i kontynuować pracę.

Dodatkowo przechowywanie modeli w bazie danych pozwala na dzielenie się nimi z innymi użytkownikami. Możliwe jest ich udostępnianie w dwojaki sposób:



Rys. 1.6: Przemiany ATP. Odpowiednio: produkcja w mitochondrium, rozprzestrzenianie się w cytozolu.



Rys. 1.7: Wykres parametrów statystycznych rozkładu glukozy w cytozolu.

- można dzielić się nimi z konkretnymi osobami poprzez indywidualne zaproszenie,
- można uczynić je modelami publicznymi, co oznacza, że każdy użytkownik `VCell`'a może do nich zajrzeć.

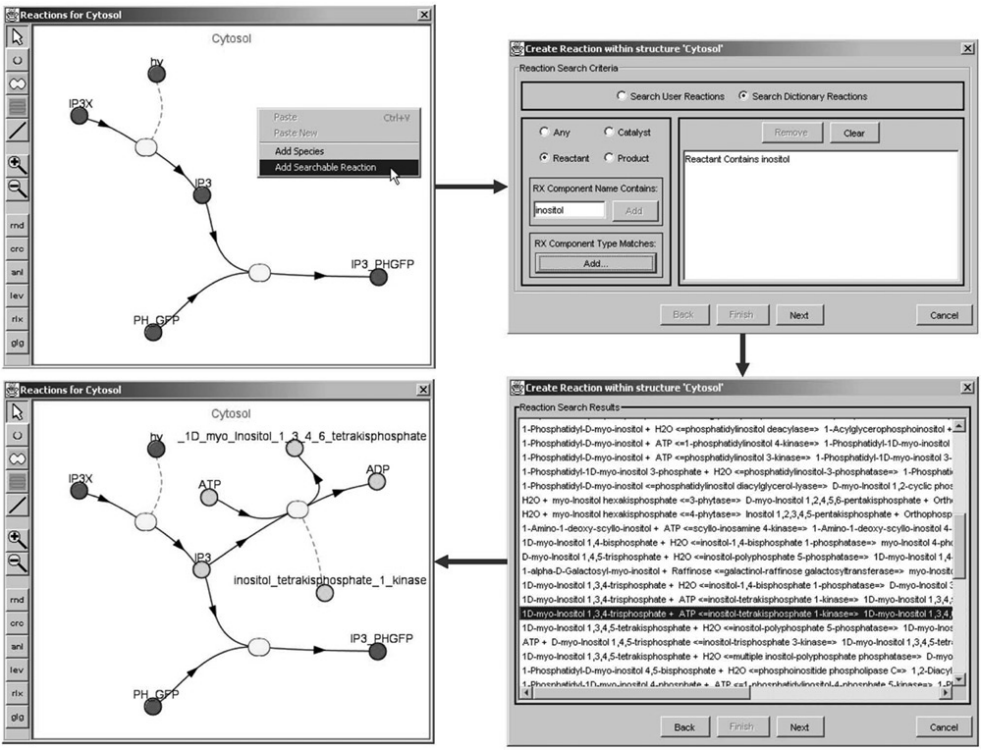
Aby jednak pracować na nich i edytować je trzeba pobrać kopię udostępnionego modelu na swoje konto.

Dzięki zastosowaniu relacyjnej bazy danych, która przechowuje informacje o każdym elemencie systemu (tzn. wszystkich substancjach i obszarach komórki), tworzenie własnego modelu jest wspierane na każdym poziomie projektowania (biologicznym – BioModel, matematycznym – MathModel oraz geometrycznym – Geometries). Istnieje możliwość przeszukania bazy pod kątem ponownego wykorzystania już zaimplementowanych komponentów. Przykładem takiego wykorzystania może być edytor BioModel, posiadający narzędzie do wyszukiwania określonych reakcji wśród wszystkich udostępnionych modeli. Wyszukana reakcja zostaje następnie wstawiona do tworzonego przykładu i powiązana z jego elementami. Przydatne również jest narzędzie komparatora, którego celem jest porównanie reprezentacji wybranych modeli w języku XML oraz wyświetlenie różnic.

Dodatkowym udogodnieniem w procesie tworzenia jest możliwość korzystania z zewnętrznych baz danych, oferujących dodatkowe źródła modelowania oraz dane. Aby jednoznacznie zidentyfikować rodzaje cząsteczek `VCell` umożliwia powiązanie ich z terminami ze słownika substancji pochodzących z *KEGG* (ang. *Kyoto Encyclopedia of Genes and Genomes*) dla molekuł, metabolitów, lipidów oraz enzymów i *SwissProt* (części Szwajcarskiego Instytutu Bioinformatyki) – bazy danych protein. Program umożliwia również bezpośredni import danych z zewnętrznych baz w celu automatyzacji procesu definiowania oraz przypisywania nowych rodzajów substancji i generowania fragmentów schematów reakcji. Przykład takiego zastosowania z wykorzystaniem bazy danych *KEGG* pokazano na rysunku 1.8.

1.6.2. Zapisywany model

Ideą twórców `VCell` było stworzenie oprogramowania ogólnodostępnego, dla osób o różnym stopniu zaawansowania zarówno w dziedzinach biologicznych, jak i informatycznych. Tworzone za jego pomocą modele powinny być dobrą bazą dla prowadzenia kolejnych badań, np. dla podobnych problemów w różnym kontekście eksperymentalnym albo jako fragment większej całości. Istotną cechą dla osiągnięcia tego założenia jest przenośność. Niejednokrotnie programy korzystają z różnych standardów opisu modelu obliczeniowego, a przez to problemem dołączanie ich do własnej pracy. Próbą zaradzenia temu problemowi było stworzenie języków formalnych, przeznaczonych do kodowania modeli, takich jak *SBML* (ang. *Systems Biology Markup Language*, format komputerowych modeli procesów biologicznych) czy *CellML* (standard oparty na XMLu, służący służący opisowi matematycznych modeli komputerowych), jak również odpowiednich ontologii, jak np. *BioPAX* (umożliwiającej integrację, wymianę, wizualiza-



Rys. 1.8: Schemat dołączania przykładowej reakcji z bazy danych *KEGG*.

cję oraz analizę danych biologicznych) czy *SBGN* (ang. *Systems Biology Graphical Notation*, przechowujących notacje graficzne używane w mapach procesów biologicznych). Próby te spotkały się z pozytywnym odbiorem przez środowiska związane z tym problemem.

Środowisko `VCell` posiada narzędzia do wspierania importu oraz eksportu modeli poprzez języki *SBML* oraz *CellML* oraz środowisko *Matlab* w celu wymiany danych z innymi narzędziami badawczymi. Problemem jednak jest fakt, że języki te nie obsługują modelowania przestrzennego oraz mają ograniczone wsparcie dla problemu transportu membranowego w komórce. Dodatkowo *SBML* nie obsługuje zjawisk elektrofizycznych. W celu zapewnienia pełnej funkcjonalności `VCell` korzysta z dialektu XML dla zapisu wszystkich elementów modelu. Język ten nazywa się *VCML* i używany jest do łatwego transferu dokumentacji i modelu poza dedykowany serwer oraz wewnętrznie do efektywnego transportu danych pomiędzy modułami programu.

Trwają starania, aby stworzyć system wymiany jednolity dla modeli ilościowych. Aby tego dokonać należy sformułować standardy modelu i adnotacji. Społeczność spotykająca się w pracy z tego rodzaju problemem rozpoczęła pracę nad projektem *MIRIAM* (ang. *Minimal Information Required In the Annotation of Models* – wymaganie minimalnej ilości informacji w adnotacji modelu), czyli repozytorium zweryfikowanych i publikowanych modeli obliczeniowych. Aplikacja

VCell została rozszerzona o wsparcie dla adnotacji *MIRIAM*, pozwalając użytkownikowi na przeglądanie, dodawanie, edytowanie i usuwanie adnotacji w standardzie *MIRIAM*.

1.7. Podsumowanie

Rolą tego rozdziału było przedstawienie możliwości komputerowego modelowania systemów biologicznych na poziomie komórkowym. Używa się do tego dedykowanych programów, które popularnie nazywa się *sztuczną* (lub *wirtualną*) *komórką*. W tym przypadku wybrane zostało powszechnie i nieodpłatnie dostępne środowisko VCell. Wykorzystano je do zdefiniowania modelu układu komórkowego – komórki roślinnej, w której zachodzą zjawiska fotosyntezy oraz oddychania komórkowego, odpowiadające pobieraniu energii ze środowiska i przetwarzaniu ją do postaci użytecznej dla organizmu. Budowa modelu wymagała zdefiniowania obszarów komórki (takich jak cytozol, mitochondrium i chloroplast), substancji, które się w nich znajdują wraz z ich stężeniami początkowymi oraz reakcji, które zachodzą pomiędzy substancjami. Następnie wykonano symulację, która pozwoliła określić, jak stężenia określonych substancji zmieniają się w czasie, a zatem zaobserwować działanie mechanizmów gospodarki energetycznej komórki.

Wyniki symulacji przeprowadzanych za pomocą programu VCell stanowią pewien zasób wiedzy, który może być przetwarzany, przechowywany i udostępniany, co także zostało omówione. Warto zwrócić uwagę na podejmowane obecnie starania w kierunku tworzenia jednolitych i bardziej uniwersalnych modeli opisu symulacji na poziomie komórkowym, co może w przyszłości ułatwić korzystanie z wyników uzyskanych za pomocą różnych środowisk symulacyjnych i przyczynić się do zwiększenia ich użyteczności.

Literatura

[1] Boris M. Slepchenko, James C. Schaff, John H. Carson, and Leslie M. Loew. Computational cell biology: Spatiotemporal simulation of cellular events. *Annu. Rev. Biophys. Biomol. Struct.*, 2002.

[2] B. Alberts, D. Bray, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Podstawy biologii komórki. Wprowadzenie do biologii molekularnej*. Wydawnictwo Naukowe PWN, 1999.

[3] A.L. Lehninger. *Bioenergetyka*. Państwowe Wydawnictwo Naukowe, 1978.

ROZDZIAŁ 2

PRZETWARZANIE ZAPYTAŃ SFORMUŁOWANYCH W JĘZYKU NATURALNYM

Ł. Czyż, T. Jordanek

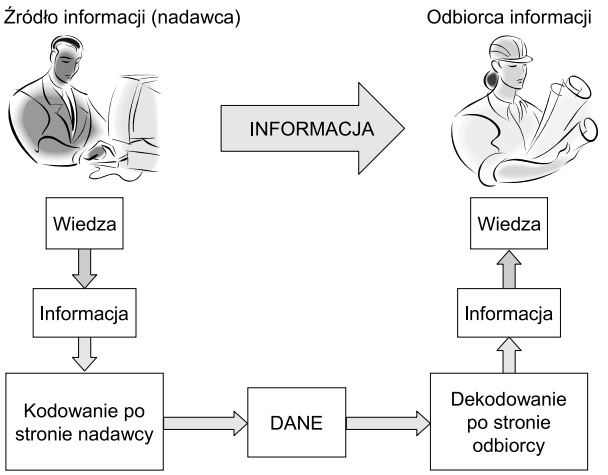
Język naturalny jest podstawowym, powszechnie stosowanym sposobem formułowania zdań w międzyludzkiej komunikacji. Jednak jego zastosowanie w komunikacji człowieka z maszyną nie jest już tak powszechne. Wiąże się ono bowiem z problemami analizy na linii człowiek-maszyna (przekształcania wyrażeń złożonych na wyrażenia proste, zrozumiałe dla systemu komputerowego) oraz syntezy na linii maszyna-człowiek (generowanie poprawnych logicznie i strukturalnie zdań złożonych odpowiednio do danych zgromadzonych w systemie komputerowych). Analiza i synteza zdań wyrażonych w języku naturalnym, choć nie należą do zadań łatwych, od dawna budzą duże zainteresowanie naukowców oraz praktyków. Zainteresowanie to przełożyło się na gwałtowny rozwój dziedziny o nazwie *przetwarzanie języka naturalnego* (ang. *natural language processing*, NLP) [1], wiążącej w swych ramach aspekty sztucznej inteligencji oraz językoznawstwa. Niniejszy rozdział ma przybliżyć czytelnikowi wybrane zagadnienia związane z NLP oraz przedstawić przykład rozwiązania prostego problemu praktycznego.

2.1. Znaczenie NLP w budowie systemów informatycznych

Zaprojektowanie i wykonanie interfejsu użytkownika umożliwiającego sprawną komunikację na linii człowiek-maszyna i maszyna-człowiek mają niezwykłą wagę dla budowanych systemów informatycznych. To właśnie ergonomia, przyjazność oraz naturalność interfejsu systemu decydują o poziomie trudności jego obsługi oraz mają wpływ na ocenę systemu przez użytkowników. Szczególnie wysoko są oceniane systemy pozwalające na komunikację w języku naturalnym lub zbliżonym do naturalnego. Język naturalny może służyć do: formułowania żądań przez użytkowników, sygnalizowania stanu procesów biznesowych w systemie, opisywania danych, zarządzania wiedzą itp.

2.1.1. System informacyjny i system informatyczny

Systemem informacyjnym można nazwać wielopoziomową strukturę różnych elementów, pozwalającą użytkownikowi na przetwarzanie informacji wejściowych na wyjściowe wg zadanych procedur i określonych algorytmów [2]. W bardziej formalnym ujęciu systemem informacyjnym nazywany jest zespół środków materialnych, finansowych, technicznych, informacyjnych, proceduralnych i ludzi, zapewniający sprawne wykonywanie określonych zadań, jak np. zarządzanie przedsiębiorstwem (na rysunku 2.1.1 przedstawiono schemat przepływu informacji w takim systemie). Dobrze funkcjonujące systemy informacyjne mają decydujący wpływ na skuteczność i efektywność zarządzania, decydują o przewadze konkurencyjnej, osiąganiu strategicznych celów itp. Wśród funkcji systemu informacyjnego wyróżnia się: gromadzenie informacji, przetwarzanie informacji, przechowywanie informacji, prezentację informacji, przesyłanie informacji, pozyskiwanie nowej wiedzy w oparciu o posiadane dane.



Rys. 2.1: Schemat przepływu informacji w systemie informacyjnym.

Pojęcie systemu informacyjnego jest często mylone z pojęciem systemu informatycznego. W ogólnym ujęciu system informatyczny to zbiór powiązanych ze sobą elementów, którego funkcją jest przetwarzanie danych z użyciem technik komputerowych, a w szczególności sprzętu i oprogramowania. Mimo mocnego związku tej definicji z techniką, system informatyczny nie może być postrzegany jedynie jako narzędzie programowe i sprzęt. Jego funkcjonowanie zależy w równej mierze od środków technicznych, jak i od użytkowników, istniejących elementów organizacyjnych (procedur), informacyjnych (wiedzy) oraz środowiska (otoczenia). System informatyczny jest bowiem częścią systemu informacyjnego, służącą do udoskonalania funkcji, szybkości i precyzji działania algorytmów oraz do zwiększenia możliwości przetwarzania, zabezpieczania i przekazywania informacji użytkownikom [3].

2.1.2. Interfejs komunikacja systemu z użytkownikiem

Badania nad metodami przetwarzanie języka naturalnego prowadzone są już od dawna. Na przełomie kilku ostatnich lat nastąpił gwałtowny ich rozwój, co w sporej części uwarunkowane było osiągnięciami dokonanymi w obszarze technologii informacyjnych, w dziedzinie sprzętu i oprogramowania. Z roku na rok zaczęły powstawać coraz to doskonalsze interfejsy systemów informatycznych, pozwalające na komunikację użytkownika z komputerem w sposób naturalny. Często motywacją do podejmowania prac w tym obszarze była chęć zdobycia lub zwiększenia popularności danego rozwiązania na rynku, bądź też chęć podtrzymania już uzyskanej pozycji. Odzwierciedlił to wzrost konkurencji wśród firm zajmujących się usługami internetowymi, telekomunikacyjnymi czy też ogólnie technologiczno-produkcyjnymi.

Obecnie istnieje wiele rodzajów interfejsów wspomagających komunikację człowieka z komputerem oraz wiele miejsc ich wykorzystania. W większości przypadków metody przetwarzanie języka naturalnego spotkać można w implementacjach systemów zaawansowanego wyszukiwania informacji. Poniżej zamieszczono opis kilku takich systemów.

- Google – firma Google Inc. stworzyła wyszukiwarkę internetową o bardzo dużych możliwościach. Swoją wielką sławę zawdzięcza zaawansowanej technologii, setkom milionom zindeksowanych stron, precyzyjności i szybkości działania. Do wyszukiwania i indeksowania zasobów internetu Google korzysta z rozmieszczonych w różnych częściach świata serwerów połączonych w sieć. Odpowiedzialnym za indeksowanie jest program o nazwie Googlebot, który okresowo pobiera zawartość stron ze swojego spisu i dodaje do niego znalezione na pobranych stronach nowe linki. Działanie wyszukiwarki opiera się na wyznaczaniu rankingu znalezionych stron. Najwyższej są oceniane strony, które posiadają w swej strukturze najwięcej odnośników ze stron cenionych przez Googla. Te i inne zalety, jak np. prostota w działaniu oraz brak reklam, przyczyniły się do rozkwitu firmy i uczynienia z niej rozpoznawalnej marki.
- Apple – firma Apple Inc. tworzy oprogramowanie, które z dnia na dzień podbija coraz więcej serc na całym świecie. W firmie tej postawiono bardzo duży nacisk na jakość produktów i usług. Słynna jest intuicyjność i przyjazność interfejsu użytkownika w telefonach jak i komputerach tej firmy. Dynamiczny rozwój oprogramowania i wzrastający na nie popyt stworzyły dodatnią pętle sprzężenia zwrotnego - a więc motor do coraz intensywniejszej ewolucji rozwiązań informatycznych. Ciekawie na tym polu wyglądają postępy w implementacjach zapewniających przetwarzanie mowy, nastawionych na umożliwienie komunikacji z urządzeniami za pomocą naturalnej mowy ludzkiej. Wybieranie głosowe numerów z listy nie stanowi już problemu, podobnie jak obsługa menu czy też rozpoznawanie liter pisanych ręcznie i wyszukiwanie informacji na podstawie zwykłych teksów wpisywanych w formularze.
- Hakia – jest to wyszukiwarka, udostępniająca API do przetwarzania języka naturalnego. Jej interfejs dzieli się na dwie grupy. Pierwsza z nich to kwerendy wyszukiwań, pozwalające użytkownikom dodawać do aplikacji możli-

wość wyszukiwania w sieci Internet. Druga to wywołania kanałów informacyjnych (tzw. feedów XML), zawierające mechanizmy przetwarzania języka naturalnego. Hakia to oprogramowanie komercyjne, stąd liczba jego użytkowników jest dużo mniejsza niż liczba użytkowników Google czy Apple. Mimo to Hakia jest rozwija się równie dynamicznie, tak jak konkurencyjne rozwiązania (`http://hakia.com/`).

2.1.3. Metody analizy języka naturalnego

W przetwarzaniu języka naturalnego można wyróżnić dwa główne etapy: przetwarzanie mowy oraz przetwarzanie tekstu. Przetwarzanie mowy wiąże się z rozpoznaniem dźwięków i zapisaniem ich jako tekst (ang. *speech recognition*, ASR) oraz wygenerowaniem dźwięków odpowiadającej zadanemu tekstowi (ang. *text to speech*, TTS).

Przetwarzanie mowy wymaga znajomość fonetyki i fonologii[4]. Są to działy lingwistyki zajmujące się systemami dźwiękowymi języków. Umożliwiają zdobycie wiedzy o poprawnej wymowie zarówno pojedynczego słowa jak i ich sekwencji. Ponadto w algorytmach przetwarzania mowy wykorzystywane są metody przetwarzania sygnałów i klasyfikacji.

Przetwarzanie tekstu często jest wykonywane z wykorzystaniem analizy morfologicznej. W analizie tej badane są wszystkie możliwe formy gramatycznych danego słowa. Wyróżnia się w niej dwa odmienne podejścia: leksemiczne oraz morfemiczne. Pierwsze z nich opiera się na *leksemie* – abstrakcyjnej jednostce języka, która łączy znaczenie, cechy gramatyczne oraz formę wyrazową. W podejściu morfemicznym podstawową jednostką analizy jest *morfem*[5]. Jest to najmniejsza część wyrazu pozwalająca na odczytanie jego znaczenia.

Do poprawnej analizy i syntezy zapytań w języku naturalnym konieczna jest znajomość reguł rządzących kolejnością występowania słów w zdaniu, ich kontekstem i znaczeniem. Mają one diametralny wpływ na poprawność interpretacji zapytań oraz sensowność generowanych odpowiedzi. Przetwarzanie tekstu wiąże się z analizą sekwencji znaków tworzących wyrazy i zdania. Kojarzone są z tym dwa pojęcia: semantyka i syntaktyka.

Semantyka to dział językoznawstwa zajmujący się badaniami nad znaczeniem słów, zwrotów, zdań i tekstów w takim kontekście. Semantyka definiuje relację formy znaku językowego do treści oznaczanej w ujęciu synchronicznym i diachronicznym. Mówiąc prościej: semantyka to dział badający relacje między podstawowym znaczeniem wyrazu a jego znaczeniem w konkretnej wypowiedzi.

Syntaktyka to dziedzina zajmująca się budowaniem odpowiednich relacji pomiędzy słowami w zdaniu. Mają one ogromne znaczenie, zwłaszcza przy generowaniu zdań w języku naturalnym. Sam wybór odpowiednich słów do budowy zdania może okazać się bezużyteczny oraz niezrozumiały dla odbiorcy, jeśli nie zostaną one właściwie uporządkowane.

Język polski w perspektywie semantycznej zaliczyć można do grupy bardzo trudnych języków. Poprawność tego stwierdzenia potwierdza fakt, iż zawiera on wiele wyrazów o znaczeniu zależnym od kontekstu oraz miejsca użycia. Przykładowo, znaczenie frazeologizmów często odbiega od dosłownego, podręczni-

kowego znaczenia słów w nich użytych. Ponadto w języku polskim występuje wiele końcówek i przedrostków, nadających wyrazom dodatkowy sens. Rodzi to wiele problemów podczas projektowania i implementacji algorytmów NLP. Pomimo trudności z tym związanych powstają coraz lepsze programy komputerowe pozwalające maszynom dobrze interpretować zdania wyrażone w języku naturalnym (jednak przy pewnych ograniczeniach).

2.2. Przykładowe zastosowania

2.2.1. Chatboty

Wykorzystanie metod NLP najczęściej można dostrzec na przykładach wdrożeń *chatbotów*. Chatboty to agenci konwersacyjni, mający sprawiać wrażenie „inteligentnych" rozmówców. Ich zadaniem jest komunikacja z człowiekiem.

Do oceny inteligencji chatbotów można użyć testu Turinga, zaproponowanego w 1950 r. Polega on na przeprowadzeniu dwóch krótkich rozmów: człowieka (sędziego) z systemem komputerowym oraz człowieka z innym człowiekiem, przy czym rozmówcy pozostają dla siebie niewidoczni. Zadaniem jest rozpoznanie, czy rozmówcą człowieka jest inny człowiek, czy też może komputer. Test uważa się za zaliczony, jeśli sędziowie pomylą się, stwierdzając podczas rozmowy z systemem iż rozmawiają z człowiekiem. Do tej pory sztuka ta nie udała się jeszcze żadnemu chatbotowi.

Typowym podejściem udawania inteligencji w implementacjach chatbotów jest próba odejścia od „niewygodnego" tematu. Na stwierdzenie użytkownika: „*Uważam iż jesteś botem.*" większość chatbotów udzieli odpowiedzi podobnej do: „*Dlaczego uważasz iż jestem botem?*" lub „*Porozmawiajmy o ...*". Do najbardziej znanych chatbotów należą: A.L.I.C.E. (zapis przykładowego dialogu z tym chatbotem zamieszczono na listingu 2.1) oraz Eliza. Wiele z nich zostało udostępnionych w internecie, dzięki czemu można samodzielnie testować ich możliwości (jak CleverBot, dostępny na stronie producenta `www.cleverbot.com`).

Listing 2.1: Przykładowa rozmowa z botem A.L.I.C.E.

```
U: Hi
B: Hi there!
U: How are you?
B: I'm doing fine thanks how are you?
U: I'm so so.
B: Do your friends call you so.
U: no
B: What is your name?
U: Łukasz
B: How can I help you, Łukasz.
U: I'm afraid you can't.
B: I think I can, don't you?
```

2.2.2. Wirtualni doradcy

Wirtualni doradcy to programy komputerowe, wspierające klientów w rozwiązywaniu ich problemów. Programy te cieszą się coraz większą popularnością wśród przedsiębiorców, zwłaszcza od chwili wdrożenia ich jako aplikacji sieciowych. Pozwalają one zmniejszyć koszty przeznaczone na obsługę klienta, bez większej szkody dla jakości tej usługi. Przykładem ich wdrożeń mogą być rozwiązania stosowane przez towarzystwa ubezpieczeniowe. Są to interaktywne strony internetowe, pozwalające dowolnemu klientowi uzyskać informację o wysokości składki ubezpieczenia oraz wartości zawieranej polisy po przejściu przez sekwencję pytań i udzieleniu na nie odpowiedzi.

Wirtualny doradca ma dostęp do dziedzinowej bazy wiedzy. Baza ta jest przeglądana na bieżąco, zgodnie ze scenariuszem wynikającym z przygotowanych pytań i dostarczanych odpowiedzi. Do cech takich aplikacji zaliczana jest przenośność i wielowątkowość. Oznacza to, że jednocześnie z programu może korzystać wielu ludzi, będąc w różnych miejscach na świecie.

2.2.3. Tworzenie zapytań do baz danych

Z punktu widzenia praktycznych zastosowań interesującym problemem jest przetwarzanie zapytań wyrażonych w języku naturalnym na język SQL. Jest to problem, w którym rozważa się bezpośrednie powiązanie interfejsu użytkownika z interfejsem dostępu do zasobów zgromadzonych w systemie informatycznym. Jego rozwiązanie polega na określeniu sensu zdania dostarczonego przez użytkownika oraz wygenerowaniu odpowiedniego zapytania w języku SQL. Wymaga to określenia kategorii tematycznej oraz elementów mających kluczowy wpływ na zwracany wynik. Podczas tej analizy wykorzystuje się różne mechanizmy badania składni i znaczenia (syntaktyki i semantyki) zdań.

W języku polskim wyrazy mające tą samą postać mogą mieć różne znaczenie, zależnie od kontekstu użycia, odmiany itp. Dlatego analizę zdań rozpoczyna się od oddzielenia podstaw poszczególnych wyrazów od przedrostków i końcówek. Choć taka analiza nie jest łatwa, to jednak powstają skomplikowane algorytmy, które w lepszym lub gorszym stopniu pozwalają na jej wykonanie. W przypadku zapytań SQL występujące w nich słowa posiadają zazwyczaj tylko jedną formę gramatyczną (są to słowa kluczowe, nazwy bazy danych, tabel i ich atrybutów, wartości atrybutów itp.). Ich dobór i postać muszą być precyzyjne, inaczej silnik bazy danych nie zwróci pożądanych wyników.

2.3. Narzędzia programowe

2.3.1. AIML

AIML (ang. *Artificial Intelligence Markup Language*) to bazujący na XML językiem znacznikowy, pozwalający definiować schemat dialogu bot-człowiek. Schemat ten bazuje na szablonach wzorzec-odpowiedź (*pattern-template*), zapisywanych jako baza wiedzy w oddzielnym pliku. Rozmowa z botem rozpoczyna się od

wczytania takiego pliku, a następnie, po otrzymaniu zapytania następuje dopasowanie do niego właściwej odpowiedzi. Możliwości komunikacyjne bota zależą więc tylko i wyłącznie od poziomu rozbudowy zapisanej bazy wiedzy.

AIML czasem nazywany jest językiem regułowym, mimo, że nie pozwala na wnioskowanie czy łączenie faktów. Określenie regułowy w tym przypadku wiąże się w możliwością definiowania szablonów wzorzec-odpowiedź. Szablony te są czułe na formę zapytań. Wzorzec zdefiniowany w pliku `.aiml` pozostanie bez dopasowania, jeśli forma gramatyczna zapytania będzie inna niż oczekiwana. Stąd zastosowania AIML ograniczają się do tworzenia prostych systemów komunikacyjnych. Przykładem niekomercyjnych programów pozwalających na pisanie oraz testowanie chatbotów są: ProgramD oraz GaitoBot AIML Editor.

2.3.2. Słowosieć

Słowosieć to leksykalna baza danych rozwijana w ramach projektu realizowanego na Politechnice Wrocławskiej, będąca odpowiednikiem angielskiego WordNetu. Pozwala na określenie znaczenia danego słowa poprzez podanie relacji znaczeniowych. Zaliczyć można do nich: synonimy, antonimy, hiperonimy, hiponimy czy meronimy. Do wyszukiwania w Słowosieci wykorzystywana jest wyszukiwarka dostępna pod adresem `http://plwordnet.pwr.wroc.pl/wordnet`. Do jej edycji oraz przeglądania w formie grafu służy aplikacja WordnetLoom (`http://nlp.pwr.wroc.pl/pl/narzedzia-i-zasoby/wordnetloom`). Słowosieć udostępniono na licencji otwartej do wszelkich zastosowań. Można ją pobrać w formatach: plwnXML (własnym), Princeton WordNet oraz VisDic (tylko część danych) ze strony: `http://nlp.pwr.wroc.pl/plwordnet/download/?lang=pl`. Więcej informacji o słowosieci można znaleźć na stronie Grupy Technologii Językowych G4.19 Politechniki Wrocławskiej dostępnej pod adresem `http://nlp.pwr.wroc.pl/`.

2.3.3. Korpus IPI PAN

Korpus IPI PAN to publicznie dostępny korpus języka polskiego (strona domowa `http://korpus.pl/`), zawierający ponad 250 mln segmentów anotowanym morfosyntaktycznie. Dostęp do tekstów wchodzących w skład korpusu jest możliwy zarówno z poziomu wyszukiwarki (aplikacji sieciowej udostępnionej pod adresem `http://korpus.pl/poliqarp/poliqarp.php`) jak i programu Poliqarp (dedykowanego narzędzie do korzystania z korpusu, udostępnione na zasadzie GPL). Wynikiem działania wyszukiwarki jest zwrócenie wszystkich wystąpień danego słowa, wraz z krótkimi fragmentami tekstu pozwalającymi na odczyt kontekstu. Dodatkową zaletą korpusu jest wykonywanie analizy morfologicznej danego słowa. Podobną funkcję oferuje program Morfeusz opisany w rozdziale 2.3.5.

2.3.4. NLTK

NLTK (ang. *Natural Language Toolkit*) to niezwykle rozbudowany toolkit do analizy i syntezy języka naturalnego w języku Python. Szczegółowy opis jego

funkcji i modułów można znaleźć na stronie producenta `www.nltk.org`. NLTK pozwala m.in. na: dostęp do wbudowanych korpusów, oznaczenie części mowy (ang. *part-of-speech tagging*), segmentację (ang. *chunking*), parsowanie, interpretację semantyczną, pisanie własnych chatbotów.

2.3.5. Morfeusz

Morfeusz to program udostępniony na licencji BSD do wykonywania analiz morfologicznych. Tekst wejściowy zostaje podzielony na pojedyncze słowa, które są poddawane interpretacji. Wynikiem działania jest uzyskanie wszystkich możliwych form znaczeniowych oraz znaczników opisujących ich kategorie gramatyczne w postaci sformatowanej, np.: `subst:sg:gen:m3` (co oznacza, odpowiednio: `rzeczownik:liczba pojedyncza:dopełniacz:rzeczownik męski rzeczowy`. Pełny opis znaczników jest dostępny w [6].

Morfeusz jest dostępny w postaci dynamicznej biblioteki oraz modułów umożliwiających jego wykorzystanie we własnych programach . Wspierane języki to: C/C++, Java, Perl, Python, SWI Prolog oraz PHP (strona domowa: `http://sgjp.pl/morfeusz/morfeusz.html`).

2.4. Autorskie rozwiązanie

Celem projektu było stworzenie aplikacji przetwarzającej zapytania w języku naturalnym na język SQL. Aplikacja została napisana w języku Python (wersja 2.7.2) z wykorzystaniem biblioteki NLTK w wersji 2.0. Źródło danych uczących zostało oparte na rzeczywistej bazie danych w MySQL. Poniżej zamieszczono jej szczegółowy opis.

2.4.1. Baza danych

Baza wiedzy wykorzystana w projekcie została dostarczona przez firmę Kraftfoods produkującą wyroby czekoladowe. Posiada ona strukturę, w której są ujęte wszystkie informacje dotyczące awarii urządzeń na liniach produkcyjnych. Rejestrowane są: miejsce awarii, czas zgłoszenia, czas trwania, czas zakończenia oraz opisy mające na celu dokładną diagnozę usterki. W tabeli występują następujące atrybuty:

- unikalny numer awarii,
- nazwiska techników wykonujących zadanie,
- nazwa linii, obszaru, urządzenia,
- typ awarii w celu działań prewencyjnych,
- opis problemu (objawy, powód, działania zapobiegawcze).

2.4.2. Program

Program ma następujący algorytm działania:

- wprowadzenie zapytania w języku naturalnym,
- rozkład wprowadzonego zapytania na ciąg wyrazów,

- parsowanie powyższego ciągu za pomocą bezkontekstowej gramatyki FCFG,
- zwrócenie wyniku w formie zapytania SQL w przypadku poprawnej analizy językowej,
- wygenerowanie wyniku zapytania do rzeczywistej bazy danych za pomocą pakietu MySQLdb,
- wyświetlenie wyniku,
- powrót do początku algorytmu dla niepoprawnej analizy zapytania.

2.4.3. Gramatyka

Wykorzystana gramatyka składa się z dwóch części: reguł leksykalnych oraz reguł gramatycznych. Pierwsze z nich przyporządkowywują każdemu słowu część mowy, co ilustruje listing 2.2.

Listing 2.2: Reguły leksykalne gramatyki bezkontekstowej.

```
N[SEM='Numer_awarii FROM tabela_awarii'] -> 'awarię'
N[SEM='WHERE Nazwa_linii = '] -> 'linii'
NN[SEM='"CR1"'] -> 'CR1'
V[SEM='SELECT'] -> 'Podaj' | 'podaj'
V[SEM='SELECT'] -> 'Wyświetl' | 'wyświetl'
V[SEM='Czas_trwania_awarii'] -> 'trwającej'
ADJ[SEM='MAX(Czas_trwania_awarii)'] -> 'najdłuższa'
ADJ[SEM='WHERE Status = "wykonano"'] -> 'wykonane'
```

Każdy wyraz, który może stanowić część zapytania w języku naturalnym powinien zostać zdefiniowany w gramatyce. Części mowy do których przyporządkowano słowa to: `N` - rzeczownik, `V` - czasownik, `ADJ` - przymiotnik. Ponadto istnieje możliwość określenia innych części mowy lub nazw własnych (symbol `NN`).

Kolejnym etapem budowy gramatyki jest skonstruowanie reguł gramatycznych. Określają one zasady budowy zdań (symbol S), które składają się z wyrażeń rzeczownikowych (symbol NP) oraz wyrażeń czasownikowych (symbol VP). Ponadto jest wymagane podanie schematu łączenia podstawowych części mowy w powyższe wyrażenia. Całość przedstawiono na listingu 2.3.

Listing 2.3: Skonstruowane reguły gramatyczne.

```
S[SEM=(?np + ?vp)] -> NP[SEM=?np] VP[SEM=?vp]
VP[SEM=?v] -> V[SEM=?v]
NP[SEM=(?d + ?n)] -> DET[SEM=?d] N[SEM=?n]
NP[SEM=(?n + ?d + ?a)] -> N[SEM=?n] DET[SEM=?d] ADJ[SEM=?a]
NP[SEM=?n] -> N[SEM=?n]
NP[SEM=(?a + ?n)] -> ADJ[SEM=?a] N[SEM=?n]
NP[SEM=(?n + ?a)] -> N[SEM=?n] ADJ[SEM=?a]
```

2.4.4. Wyniki i wnioski

Implementacja programu oraz gramatyki umożliwiła wprowadzenie podstawowych zapytań w języku naturalnym. Na listingu 2.4 przedstawiono przykład uzyskiwania danych z bazy MySQL bez znajomości składni języka SQL. Program

umożliwia, między innymi, sprawdzenie maksymalnego, najkrótszego czy średniego czasu awarii, a także liczby urządzeń na wybranej linii.

Listing 2.4: Przykład działania aplikacji (zapytania o średni i najdłuższy czas awarii oraz liczbę urządzeń na linii).

```
Wpisz zapytanie w języku naturalnym (quit aby zakończyć)
> Ile trwała średnia awaria?
SELECT  AVG(Czas_trwania_awarii) FROM tabela_awarii
97.6954
> Ile trwała najdłuższa awaria?
SELECT  MAX(Czas_trwania_awarii) FROM tabela_awarii
34620
> Podaj liczbę urządzeń na linii CR2.
SELECT  COUNT(Nazwa_urzadzenia) FROM Tabela_awarii
WHERE Nazwa_linii =  "CR2"
183
> quit
```

Na listingu 2.5 pokazano przykład pobierania z bazy danych numerów awarii w trakcie realizacji lub awarii, których czas trwania wynosił dokładnie 100 minut.

Listing 2.5: Przykład działania aplikacji (zapytanie o numery awarii w trakcie realizacji oraz o określonym czasie trwania).

```
Wpisz zapytanie w języku naturalnym (quit aby zakończyć)
> Wyświetl awarie w trakcie realizacji.
SELECT Numer_awarii FROM tabela_awarii
        WHERE Status = "w trakcie realizacji"
9224
> Wyświetl awarię trwającą 100 minut.
Spróbuj ponownie
> Wyświetl numer awarii trwającej 100 minut.
SELECT Numer_awarii FROM tabela_awarii
  WHERE Czas_trwania_awarii =100
7467
7406
9003
9351
7132
9040
> quit
```

Stworzona aplikacja pozwala na przetwarzanie zdań, ale tylko w określonym zakresie kontekstów i słów. Wynika to z konieczności implementacji każdego wyrazu w gramatyce, z której korzysta skrypt oparty o toolkit NLTK. Najbardziej problemowe do przetworzenia są liczby, które nie można opisać jedną składową, lecz każdą oddzielnie za pomocą zmiennych. Na tej podstawie, mając zdefiniowaną pulę najczęściej zadawanych pytań do bazy awarii występujących w firmie, wyszczególniono słowa odpowiedzialne za właściwy przekaz ich treści. Mimo

tych niedogodności, program poprawnie generuje zapytania w języku SQL oraz zwraca ich wynik, łącząc się z rzeczywistą bazy danych. Jego możliwości oraz funkcjonalność zależą tylko i wyłącznie od rozbudowy gramatyki o dodatkowe reguły leksykalne oraz gramatyczne. Pozwolą one wówczas na przetworzenie bardziej złożonych zapytań, wprowadzanych przez użytkownika w języku naturalnym.

Literatura

[1] Wikipedia: Przetwarzanie jezyka naturalnego. http://pl.wikipedia.org/wiki/Przetwarzanie_języka_naturalnego.

[2] System informacyjny. http://www.uci.agh.edu.pl/uczelnia/tad/PSI6/wyklady_html/wyklad01.htm.

[3] T. Kubik. Współdzielenie danych przestrzennych w systemach informatycznych. In A. Dyczko and A. Krawczyk, editors, *Geomatyka górnicza – praktyczne zastosowania*. Wydawnictwo Fundacji dla AGH, 2011.

[4] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 1 edition, 2000.

[5] Wikipedia: Morfologia (językoznawstawo). http://pl.wikipedia.org/wiki/Morfologia_(językoznawstwo).

[6] Marcin Woliński. System znaczników morfosyntaktycznych w korpusie IPI PAN. *Polonica*, pages 39–54, 2004.

ROZDZIAŁ 3

MODELOWANIE TŁUMU I PANIKI

M. Frontkiewicz, P. Kaczmarek, M. Pochna

3.1. Psychologiczno-socjologiczne podstawy tematu

Zachowanie się dużej grupy ludzi postrzeganej jako tłum stanowi przedmiot badań nie tylko socjologów, ale również psychologów czy politologów. Wiele zagadnień z tego obszaru dotyczy ważnych i praktycznych problemów. Pozwalają one spojrzeć m.in. na różne aspekty organizacji imprez masowych (jak np. koncerty, mecze, protesty), a w szczególności na aspekty związane z zapewnieniem bezpieczeństwa i unikania potencjalnych zagrożeń.

Niniejszy rozdział poświęcony jest modelowaniu i symulowaniu zjawiska paniki towarzyszącemu zachowaniu się tłumu za pomocą wybranych narzędzi programowych. Modelowanie zachowania się tłumu nie jest łatwe. Mimo to coraz częściej podmioty odpowiedzialne za bezpieczeństwo oraz kierowanie tłumem podejmują się tworzenia i wykorzystania takich modeli. Poprzez symulacje z ich wykorzystaniem próbują projektować drogi ewakuacyjne, planować scenariusze postępowania w sytuacjach zagrożenia itd.

3.1.1. Psychologia tłumu

Psychologia tłumu stanowi dział psychologii społecznej, w którym badane są zachowania jednostek w wielkiej masie ludzkiej. Na poziomie ogólnym badania te obejmują analizę zachowania i procesów myślowych poszczególnych jednostek tworzących tłum oraz tłumu jako całości. Za głównych myślicieli psychologii tłumu uznaje się takie osobistości, jak: Rene Girard, Gustave Le Bon, Wilfred Trotter, Gabriel Tarde, Zygmunt Freud, Elias Canetti, Steve Reicher czy Julia Constintine. Natomiast za jej ojca uważa się francuskiego lekarza, antropologa i socjologa Gustava Le Bon, który jako pierwszy zwrócił uwagę na zjawisko odmiennego zachowania się ludzi występujących pojedynczo i w grupie [1].

Powszechnie słowem „tłum" określane jest zgromadzenie osób, bez względu na ich narodowość, płeć czy wyznanie, a także bez uwzględniania przyczyny zgromadzenia. W psychologii pojęcie „tłum" posiada odmienne znaczenie. Według Gustawa le Bon [1] dopiero w przypadku zbiegu pewnych okoliczności, i tylko

w tych okolicznościach, zbiorowość ludzi nabiera zupełnie nowych właściwości. Właściwości te różnią się od indywidualnych właściwości poszczególnych osób. Cytując za [1] „w tłumie zanika świadomość własnej odrębności, uczucia i myśli wszystkich jednostek mają jeden tylko kierunek".

Tłum posiada określone cechy. Zalicza się do nich zanik świadomości własnego *ja* u poszczególnych osób i poddanie uczuć i myśli pewnemu wspólnemu kierunkowi działań. Cecha ta występuje bez względu na liczbę zgromadzonych w jednym miejscu osób. Przypadkowe wydarzenie może spowodować zmianę charakteru tłumu, a wraz z tym wystąpienie cechami specyficznych dla jego postępowania. Bez względu na to, jakie jednostki tworzą tłum, będąc jego częścią posiadają coś w rodzaju duszy zbiorowej. Dusza ta każe im inaczej myśleć, działać i czuć, niż myślałaby, działała i czuła każda jednostka z osobna. Rzecz ta bywa również ujmowana inaczej. Autorka [2] odróżnia pojęcie tłumu będącego stanem fizycznej bliskości od publiczności, którą charakteryzuje bliskość duchowa mimo znacznej odległości.

Podział tłumu

Istnieje wiele sposobów klasyfikacji tłumu. Zależnie od teorii przyjętej w trakcie badań można wyróżnić różne jego kategorie. Według [1] istnieją następujące rodzaje tłumów:

- **tłumy heterogeniczne** - składające się z bardzo różnorodnych jednostek (pod względem zawodowym, pod względem poziomu rozwoju umysłowego i pod innymi względami),
  - **bezimienne** - np. tłum uliczny, gromada gapiów,
  - **nieanonimowe** - np. parlament, ława przysięgłych,
- **tłumy homogeniczne** - charakteryzują się swego rodzaju jednorodnością cech tworzących je jednostek,
  - **sekty** - pierwszy stopień organizacji tłumów homogenicznych, obejmujący jednostki często różniące się wychowaniem, pochodzeniem i wykształceniem, które łączy tylko wspólna wiara lub wspólny cel (jak np. w sektach religijnych i partiach politycznych),
  - **kasty** - najwyższy stopień organizacji, łączący jednostki jednego i tego samego zawodu, pochodzące na ogół z tych samych sfer i wykazujące mniej więcej jednakowy stopień inteligencji (jak np. kasta wojskowa lub kapłańska),
  - **warstwy** - łączy jednostki różnego pochodzenia, zbliżone do siebie wspólnotą zajęć, podobieństwem sposobu życia i warunków otoczenia (jak mieszczaństwo, chłopi itd.),

Aby dobrze zrozumieć znaczenie tego podziału należy zwrócić uwagę na kontekst historyczny (pozycja [1] ukazała się w roku 1895 i odnosiła się do rewolucji przemysłowej oraz jej następstw politycznych).

Herbert Blumer, [3], skupiając się na znaczeniu interakcji pomiędzy uczestnikami zachowań zbiorowych, wyróżnił 4 rodzaje tłumu:

- **tłum przypadkowy** - charakteryzuje go słaba interakcja pomiędzy jego jednostkami lub nawet jej brak. Jednostki te przyciągnęło jakieś, często przypadkowe, wydarzenie. Przykładem takiego tłumu są osoby przyglądające się wypadkom czy też osoby zgromadzone wokół stoiska w hipermarkecie.
- **tłum konwencjonalny** - skupia jednostki zebrane w jakimś celu, który to cel jest jednak osiągany przez każdą z nich z osobna. Przykładem takiego tłumu są pasażerowie na przystanku lub widzowie w kinie. Tłum konwencjonalny czasem nazywany jest publicznością. Charakterystyczną jego cechą jest to, że zdania jednostek na temat osiąganego celu mogą być różne.
- **tłum ekspresyjny** - szczególną rolę odgrywa w nim jakiś ładunek emocjonalny i na nim oparta jest interakcja. Przykładem takiego tłumu są uczestnicy karnawału w Rio, parady miłości w Berlinie czy zabaw sylwestrowych na rynkach miast. Publiczność na koncertach rockowych często ma cechy tłumu ekspresyjnego. Tłum taki może przejawiać zachowania normalnie niedopuszczalne.
- **tłum aktywny** - nastawiony na działalność niszczycielską, której celem jest rozładowanie emocji lub zniszczenie jakiegoś zła czy przeciwnika. Przykładem takiego tłumu są agresywni kibice na meczach piłkarskich.

Powyższy podział został uzupełniony przez autorów [4] o piąty rodzaj tłumu:

- **tłum protestujący** - jest szczególnym przykładem tłumu, który wykazuje cechy tłumu konwencjonalnego (dość dobra organizacja) oraz tłumu aktywnego (działalność destruktywna).

W kolejnym podziale, biorącym pod uwagę przyczynę powstania, wyróżnia się następujące typy i podtypy tłumu:

- **tłum agresywny** - atakuje jednostkę, zbiorowość lub instytucję. Znane są trzy rodzaje tego typu tłumu:
  - **tłum linczujący** - powstaje w celu wymierzenia samosądu rzeczywistemu lub domniemanemu sprawcy ogólnie potępianego czynu (napad, morderstwo, gwałt itp.). Powstaje, gdy gromadzi się większa liczba osób wtapiających się w psychospołeczny klimat potężniejącego tłumu, który dąży do zemsty. Tłum ten zdolny jest do popełniania okrutnych czynów, przekonany, że działa w słusznej sprawie.
  - **tłum terroryzujący** - jest przeważnie skierowany przeciw jakimś mniejszościom narodowym, religijnym, seksualnym czy innym, podejrzanym o czyny wywołujące oburzenie znacznej części jakiejś społeczności. Przykłady takiego tłumu spotykać można w przypadku rozruchów kibiców na stadionach, gdy pseudo kibice jednej drużyny atakują kibiców drugiej. Rezultatem działań takich tłumów są niekiedy ofiary śmiertelne.
  - **tłum walczący** - do tego typu tłumu należą zgromadzenia podczas strajków i manifestacji, które są rozpraszane przez służby porządkowe.
- **tłum uciekający** - występuje w dwóch odmianach: tłumu ogarniętego paniką na ograniczonej przestrzeni, przeważnie podczas zbiegowiska zagrożonego niespodziewanym niebezpieczeństwem, oraz sformalizowanej grupy na-

potykającej zagrożenie rozpadu więzi formalnej. W tłumie uciekającym, bez względu na odmianę, jednostki ogarnięte strachem kierują się instynktem samozachowawczym.

- **tłum nabywający (rabujący)** - występuje w okresach kryzysów i gospodarczego niedostatku. Pojawiają się wówczas różnorodne zbiegowiska i tłumy rabujące sklepy, banki, a nawet mieszkania zamożniejszych rodzin. Tłumy rabujące powstają w różnorodnych warunkach i okolicznościach, często też wykształcają się z tłumów agresywnych, a nawet demonstrujących.
- **tłum demonstrujący (ekspresywny)** - w odróżnieniu od poprzednich jest zorganizowany. Jest on wyrazem wdzięczności, uznania, potępienia, pogardy, pochwały lub protestu. Demonstracja jest tłumem zorganizowanym po to, aby wyraźnym uczuciom dodać większej powagi i aby nie stwarzać szans na przekształcenie się tłumu w tłum agresywny, rabujący albo ogarnięty paniką.

Bez względu na rodzaj, w każdym tłumie uwidoczniają się pewne cechy wspólne. W tłumie obserwuje się zjawisko dezindywidualizacji (częściowego zaniku niektórych składników i cech osobowości jednostek na okres wtopienia się w zbiorową psychikę tłumu). Pojawia się także wzmożone naśladownictwo, spowodowane zanikiem indywidualnej refleksyjności jednostki. Na gruncie emocjonalnym dochodzi do zaraźliwości emocji, tj. wytwarzaniu się identycznych lub podobnych stanów emocjonalnego napięcia, powstającego na gruncie podobnych postaw, nastrojów i oczekiwań. Charakterystyczną cechą jednostek będących w tłumie i pod jego działaniem jest także podatność na sugestie i oddziaływania innych uczestników tłumu. Tłum rozpada się wówczas, gdy słabnie więź zespalająca go w skutek nowej podniety, wywołującej zupełnie inne niż dotychczas emocje.

Zgodnie z [5] w życiu tłumu można wyróżnić następujące etapy (definicje pobrane spod adresu: `http://portalwiedzy.onet.pl/133636,,,,reakcje_tlumu,haslo.html`):

- **Tworzenie się tłumu** – tłum tworzy się, gdy ludzie wspólnie napływają i gromadzą się na jakimś obszarze. W takim zgromadzeniu ważnym czynnikiem jest bliskość fizyczna. Charakterystyczne jest też przekonanie o wspólnym celu lub (podświadomy) bezrefleksyjny nacisk ukierunkowujący dążenia.
- **Polaryzacja** – tłum przestaje działać, gdy nie następuje żadna konfrontacja, tzn. nie ma żadnej pobudzającej przeszkody. Zawsze jest w stanie gotowości do określenia wroga.
- **Ramy przestrzenne** – aby tłum mógł powstać, konieczne jest określenie obszaru, czyli ustanowienie ram przestrzennych dla zbiorowych akcji (np. ukształtowanie terenu może nasilać stłoczenie). Pewne formy ukształtowania przestrzennego sprzyjają popychaniu tłumu do działań. Na stadionach miejsca stojące sprzyjają zbiorowym akcjom, które mogą wymknąć się spod kontroli, ponieważ fotele publiczności nie wyznaczają ram przestrzennych, co stwarza pokusę zdobycia terytorium, np. przez okrzyki i wrzaski lub atak fizyczny na terytorium wrogich kibiców. „Ukształtowanie sceny" wpływa w ten sposób w dużym stopniu na zachowanie uczestników działań zbiorowych.

- **Aktywne mniejszości** – problem przywództwa przedstawia się inaczej w tłumie niż w grupach. W grupie lider jest tym, kto wydaje polecenia, natomiast lider tłumu sam niewiele może osiągnąć, jeśli go nie wspiera grupa przywódcza. W trakcie działań tłumu na ogół to nie przywódcy oficjalni odgrywają znaczącą rolę, lecz aktywne mniejszości i ich charyzmatyczni reprezentanci. To oni mogą pociągnąć za sobą tłum, jeśli adekwatnie wyrażają jego dążenia. Mniejszość staje się przedstawicielem większości, tzn. uosabia wszystkich chętnych do działania i wyraża ich skryte lub otwarte pragnienia
- **Przebieg akcji** – każda akcja tłumu ma swoją dramaturgię, świadomie zaplanowaną lub przypadkową, która zwykle ma charakter eskalacji. Widoczna słabość przeciwnika wywołuje „reakcję łańcuchową", której nie można zatrzymać. W najwyższej ekstazie tłum jest zdolny do wszystkiego, jego członkowie szukają ryzyka, nie dbając o groźbę obrażeń. Dopiero gdy jest już po wszystkim, cieszą się, jeśli udało się im wyjść cało z opresji, bez przykrych konsekwencji. Podczas dłuższych działań, trwających wiele dni lub tygodni, następują z reguły okresy spokoju jak również zdarzają się momenty zwrotne – nagłe zmiany kierunku działania.
- **Reperkusje działań** – koniec działań tłumu nie oznacza jeszcze zamknięcia całej sprawy. Po akcji mają miejsce opóźnione reakcje, które też wpływają na zbiorowość. Uczestnicy działań mogą się chwalić, że przy tym byli i przez to zwiększać swój prestiż. Mogą zbierać inf. podawane przez media na temat zbiorowego działania, w którym brali udział, i przechowywać je w celu pielęgnowania wspólnych wspomnień. W zależności od rangi wydarzenia mogą tworzyć się w końcu wspólnoty weteranów.

3.1.2. Panika

Trudno o jedną, ścisłą definicję paniki. Termin ten jest bowiem używany w odniesieniu do wielu sytuacji i zachowań. Zgodnie z [6] „prawie każdy rodzaj społecznie dezorganizacji lub osobisty rodzaj działalności zaburzającej został określony jako panika". Zakres tego pojęcia obejmuje wszystko: od zjawisk związanych z psychiatrią, po zjawiska ekonomiczne (np. występujące zaburzenia na giełdzie). Dodatkowo, w każdym z różnorodnych przypadków istotę paniki stanowi inne zdarzenie. Niektóre behawioralne definicje paniki opisują ją jako zwierzęce zachowania, w których ludzie potrafią stratować się na śmierć [6]. Pojęcie to może odnosić się także do stanu wewnętrznego, w którym jednostka odczuwa (uzasadniony lub nieuzasadniony) intensywny strach. Jako panika postrzegane są także ogólnie bezużyteczne zachowania. Literatura dostarcza wielu różnych, spójnych definicji. W [7] zdefiniowano panikę jako wspólne podążanie, zachowanie w oparciu o histeryczną wiarę i przekonanie. Wskazuje na to, iż po przyjęciu przekonania o jakimś ogólnym zagrożeniu, ludzie uciekają, stosując ustalone wzorce interakcji społecznej w celu zachowania życia, mienia lub poczucia siły w obliczu zagrożenia. Tak ogólna definicja obejmuje wiele konkretnych sytuacji, w których może dojść do wybuchu paniki, np. pole bitwy, tonący statek, płonący budynek, a także rynek bankowy.

Panika jako reakcja na zagrożenie

W niniejszej pracy pojęcie paniki będzie postrzegane jako niekontrolowane zachowania tłumu w obliczu sytuacji zagrożenia. Może być ona rozumiana jako nagły i nieoczekiwany wybuch silnego i szybko rozprzestrzeniającego się zbiorowego strachu, wywołanego najczęściej urojeniem lub wyolbrzymionym niebezpieczeństwem, powodującego gwałtowną ucieczkę. Ucieczce tej towarzyszy zaćmienie świadomości ulegających jej osób. Definicja uwzględniająca histeryczną wiarę i przekonania jako przyczynę nieracjonalnych zachowań ściśle koresponduje z tą przedstawioną w [7]. W takim ujęciu, powstawaniu paniki sprzyjają często takie czynniki jak chaos, dezinformacja, bezradność, brak pomocy lub jednostki przywódczej.

Przyczyny paniki mogą być różne. Najczęściej są to przyczyny emocjonalne, w szczególności strach. Pod jego wpływem ludzie, kierując się instynktem szukania ratunku przed rzeczywistym lub rzekomym niebezpieczeństwem, wybierają ucieczkę. Strach może być tak silny, że człowiek ogarnięty paniką nie zastanawia się nad swoim zachowaniem, działa często instynktownie, a także irracjonalnie. Bezpośrednią przyczyną prowadzącą do wybuchu paniki są zwykle czynniki fizyczne i fizjologiczne, np. niedostateczna ilość pożywienia i wody, warunki klimatyczne, widok licznych ludzkich ofiar czy też zmęczenie, bezsenność, depresja oraz brak orientacji w danym terenie.

Do sytuacji, w których może dojść do wybuchu paniki, zalicza się: ataki terrorystyczne; pożary; powodzie; wypadki; niespodziewane dźwięki, zwłaszcza: strzały, krzyki, paniczne reakcje innych ludzi. Panika może wystąpić także podczas innych okoliczności, np. w trakcie koncertu czy meczu, gdy nagłe zdarzenie spowoduje reakcję tłumu.

Do czynników środowiskowych sprzyjających wybuchom oraz rozprzestrzenianiu się paniki zalicza się: złą widoczność (spowodowaną przez np. przez mgłę, dym lub porę dnia), złe warunki atmosferyczne (zbyt wysoka/niska temperatura), liczba ludzi, możliwości ewakuacyjne. Duży wpływ na zachowanie jednostki ma także postępowanie innych ludzi będących w tej samej sytuacji zagrożenia, a także postępowanie zespołu ratowniczego lub kierującego ewakuacją (nerwowe zachowania ratowników oraz brak odpowiednich informacji może sprzyjać rozprzestrzenianiu i nasilaniu się paniki).

Psychologiczne i socjologiczne ujęcie zachowań

Pod wpływem paniki ludzie zaczynają zachowywać się w sposób inny niż zazwyczaj. W specyficznej sytuacji zagrożenia i niepewności, z psychologicznego punktu widzenia ludzie

- tracą poczucie czasu,
- tracą umiejętność kontrolowania siebie,
- nie reagują na polecenia,
- nie poddają się perswazji,
- zachowują się w sposób irracjonalny.

Z socjologicznego puntu widzenia, uwzględniając wpływ tłumu na jednostkę, dochodzą takie zachowania i postawy, jak:

- otwartość na sugestie tłumu - zachowanie takie samo jak innych osób ogarniętych paniką,
- skupienie się na tej samej rzeczy lub idei co tłum,
- poczucie siły jakie daje tłum,
- obniżenie strachu przed sądem i karą,
- zamiana „ja" na „my".

3.2. Modelowanie zagadnienia paniki w tłumie podczas ewakuacji

W przypadku wystąpienia nagłych zagrożeń i konieczności ewakuacji dużej liczby ludności, może dojść do wybuchu paniki, która znacznie utrudnia akcję ratowniczą, a w skrajnych przypadkach prowadzi do ogromnych strat i tragedii. Aby nie dopuścić do paniki oraz aby zminimalizowania skutki jej wystąpienia prowadzi się różne badania, a ich wyniki stara się zastosować w rzeczywistości. Analiza tego zjawisko nie należy jednak do łatwych.

Rzeczywiste wystąpienie paniki jest trudne do przewidzenia, a jeśli już wystąpi, nie daje się odpowiednio zmierzyć i zbadać. Dlatego najczęściej badania tego zjawiska prowadzi się z wykorzystaniem symulatorów oraz specjalnie do tego celu stworzonych modeli. Narzędzia te umożliwiają obserwowanie zachowania tłumu oraz jego zmiany w zależności od występowania i nasilenia różnych czynników. W niniejszej pracy skupiono się na modelowaniu zachowań tłumu i paniki podczas ewakuacji z pomieszczeń zamkniętych.

3.2.1. Różnorodne ujęcie tematu

Modelowanie zachowania się tłumu podczas ewakuacji, uwzględniające możliwość wystąpienia paniki, może odbywać się w różnorodny sposób. Modelowanie tego typu zachowań jest powszechne, między innymi podczas projektowania wyjść bezpieczeństwa oraz planowania ewakuacji (z pomieszczeń bydynków, statków itp.). Istnieje wiele komercyjnych narzędzi wspierających takie modelowanie.

Modele wykorzystywane w trakcie wspomnianych badań można podzielić na trzy kategorie: modele przepływowe (ang. *flow-based models*), modele automatów komórkowych (ang. *cellular automata models*), modele agentowe (ang. *agent-based models*). Istnieją także modele powstające poprzez połączenie elementów wymienionych modeli.

Model przepływowy charakteryzuje globalne podejście do problemu symulacji zachowania się tłumu, tj. poszczególne osoby nie są symulowane jako jednostki, ale są traktowane jako element przepływu. Model ten uwzględnia fizyczne właściwości środowiska oraz przepływu.

Model automatów komórkowych opiera się na reprezentacji środowiska w postaci siatki komórek. Komórki mogą być zajęte przez jednostki, przeszkody

itp. Charakter przemieszczeń jednostek pomiędzy komórkami określa ich lokalna gęstość, prędkość oraz wielkość przepływu. Głównym czynnikiem wyróżniającym ten model jest dyskretyzacja przestrzeni.

Model agentowy pozwala na nadanie jednostce indywidualnych właściwości, które determinują jej zachowanie w danej sytuacji i wpływają na jej reakcje. Jest to istota tego modelu. Wymienione wcześniej modele nie pozwalały na przypisanie jednostce takich atrybutów (tj. atrybutów zależnych od jej interakcji z otoczeniem i tłumem).

3.2.2. Rozwinięcie metod agentowych

W podejściu agentowym główną rolę odgrywają czynniki zdefiniowane jako istotne i wpływające na ruch agenta w tłumie. Mogą to być zarówno czynniki fizyczne - jak na przykład charakterystyczna dla agenta prędkość poruszania się, lub psychiczne - takie, jak sposób postrzegania czy też określone reakcje na zagrożenie. W zależności od tego, jakie czynniki determinują postępowanie agenta, powstają różne modele.

Metody agentowe nadają się najlepiej do symulowania ruchu tłumu o dużej gęstości oraz ewentualnych wystąpień paniki, ponieważ jako jedyne oddają zróżnicowany charakter poszczególnych uczestników ruchu. Istnieje wiele rozwinięć i modyfikacji podstawowego podejścia agentowego. Szczególnie ciekawa wydaje się takie aspekty, jak: analiza reguł ruchu według których postępują jednostki; modelowanie na dwóch poziomach szczegółowości, pozwalające zauważyć wytworzone wzorce ruchu; analiza kluczowych cech psychologicznych agentów.

Podejście agentowe zostanie omówione dokładniej w dalszej części rozdziału, przy okazji prezentacji wyników symulacji dla różnych modeli zachowań agenta.

Heurystyczne reguły ruchu

Obserwacje zachowania tłumu ukazują dużą różnorodność samoorganizujących się wzorców ruchu, jak np. spontaniczne tworzenie się równoległych, jednokierunkowych alejek w przepływie pieszych czy też ruch typu *stój-idź* w miejscach zatłoczonych. Ruch tłumów o dużym zagęszczeniu również rządzi się pewnymi wzorcami i modelami, lecz są one zdecydowanie trudniejsze do opisania. Popularne modele oparte o oddziaływania społeczne (ang. *social force models*) [8, 9] opisują ruch agentów jako wypadkową sił przyciągających go do celu i odpychających od przeszkód i innych ludzi. Wadą tej metody jest rozpatrywanie sił pochodzących od każdej jednostki osobno, a następnie łączenie ich zgodnie z zasadę superpozycji. Pojawiające się problemy to: metody łączenia sił pochodzących od każdej z osób; określenie promienia, w którym inni agenci wpływają na ruch; rozstrzygnięcie, czy niewidoczne dla agenta przeszkody mogą wpływać na jego zachowanie.

W pracy [10] zaproponowano heurystyczne podejście do planowania trasy ruchu agenta, oparte na analizie jego pola widzenia. Heurystyki wykorzystujące informację wzrokową określają pożądany kierunek ruchu α_{des} i pożądaną prędkość v_{des}. Założeniem heurystyk jest półsekundowy okres czasu potrzebny agen-

towi na zatrzymanie się w celu uniknięcia zderzenia, który definiuje minimalną odległości do przeszkód w zależności od jego prędkości. Założenie to opiera się na eksperymentach przeprowadzonych w laboratoryjnych warunkach, opisanych w pracy [11].

W rozważanym modelu agenci charakteryzowani są przez:

- aktualną pozycję $\overrightarrow{x_i}$,
- aktualną prędkość $\overrightarrow{v_i}$,
- parametry budowy ciała - masę m_i i pole powierzchni - w uproszczeniu okrąg o promieniu $r_i = m_i/320$,
- prędkość komfortowego chodu v_i^o,
- położenie docelowe O_i - jest nim najczęściej wyjście z budynku,
- kąt widzenia w każdą stronę Φ_0.

Pierwszym krokiem algorytmu heurystycznego jest wyliczenie dla wszystkich kątów α z zakresu $[-\Phi_0, \Phi_0]$ (z rozdzielczością dostosowaną do maszyny, na której prowadzone są obliczenia) odległości do najbliższej przeszkody $f(\alpha)$. W wypadku braku kolizji dla danego kąta, wartość funkcji f równa jest $d_{\max}$, czyli zasięgowi wzroku agenta.

Obserwacje ruchu jednostek (pieszych) pokazują, że szukają oni tras pozbawionych przeszkód, przy jednoczesnym unikaniu zbyt dużych odchyleń od ścieżki prowadzącej do ich celu (określonej przez kierunek α_0). Celem algorytmu heurystycznego jest więc znalezienie kierunku ruchu α_{des} będącego kompromisem dla powyższych kryteriów, będących często w sprzeczności. Zaproponowany algorytm wybiera kąt α_{des} tak, aby minimalizował on równanie (3.1).

$$d(\alpha) = d_{\max}^2 + f(\alpha)^2 - 2 d_{\max} f(\alpha) \cos(\alpha_0 - \alpha) \tag{3.1}$$

Drugi krok algorytmu pozwala określić pożądaną prędkość ruchu v_{des}. Dobierana jest ona tak, aby w każdy momencie agent znajdował się na tyle daleko od przeszkód, aby mógł się przed nimi zatrzymać. Można ją opisać równaniem (3.2), w którym d_h oznacza odległość między agentem, a najbliższą przeszkodą w kierunku ruchu α_{des}, zaś τ to minimalny czas potrzebny na zatrzymanie agenta (jak wspomniano wcześniej wynosi on $0,5s$).

$$v_{des}(t) = \min(v_i^0, \frac{d_h}{\tau}) \tag{3.2}$$

Przedstawiony model nie pomija również zderzeń między agentami, które mogą wystąpić w wyniku nadmiernego ich zagęszczenia. Wpływają one na nieplanowane ruchy agentów, których nie da się wyjaśnić w ramach powyższej heurystyki. Siłę oddziaływania agenta z przeszkodą definiuje równanie (3.3), w którym k jest współczynnikiem proporcjonalności, $g(x)$ jest równe 0 gdy agent nie styka się z przeszkodą, zaś x w przeciwnym wypadku, $\overrightarrow{n_{ij}}$ to wektor o długości 1, skierowany od agenta j do i, zaś d_{ij} to odległość między środkami ciężkości obu agentów.

$$\overrightarrow{f_{ij}} = kg(r_i + r_j - d_{ij})\overrightarrow{n_{ij}} \tag{3.3}$$

Biorąc pod uwagę równania (3.2) i (3.3), a także traktując analogicznie do sił związanych ze zderzeniami z agentami siłę f_{iW} pochodzącą od kontaktu agenta ze ścianą (w równaniu sił, $r_j - d_{ij}$ zastąpione jest odległością agenta do ściany $-d_{iW}$), otrzymujemy równanie (3.4), opisujące wypadkowe przyspieszenie agenta. Równanie to tylko pozornie przypomina równanie oddziaływań społecznych, ponieważ wartości siły $\overrightarrow{f_{ij}}$ są większe od zera tylko przypadku bezpośredniego kontaktu fizycznego między agentami.

$$\frac{d\vec{v_i}}{dt} = \frac{(\overrightarrow{v_{des}} - \vec{v_i})}{\tau} + \sum_j \frac{\overrightarrow{f_{ij}}}{m_i} + \sum_W \frac{\overrightarrow{f_{iW}}}{m_i} \tag{3.4}$$

Można również zaproponować miarę oddziaływań występujących w tłumie, która pozwoli wnioskować o prawdopodobieństwie wybuchu paniki i stratowania ze względu na zbyt duży ścisk. Miara sił odczuwanych przez pojedynczy agent jest wyrażona równaniem (3.5), zaś lokalny opis ścisku w danym punkcie x wyliczyć można za pomocą równania (3.6), gdzie $f(d_{ix})$ jest Gaussowską funkcją wagową, argumentem której jest odległość danego agenta od rozważanego punktu.

$$C_i(t) = \sum_j \left|\left|\overrightarrow{f_{ij}}(t)\right|\right| \tag{3.5}$$

$$C(x,t) = \frac{\sum_i C_i(t) f(d_{ix})}{\sum_i f(d_{ix})} \tag{3.6}$$

Symulacje opisane przez autorów przedstawionego modelu potwierdzają, że odwzorowuje on poprawnie zarówno zjawiska płynnego ruchu pieszych (tworzenie się jednokierunkowych ścieżek, stabilizacja prędkości ruchu) jak i przejście do ruchu z turbulencjami i falami idź-stój, które powstają przy zbyt dużej gęstości agentów. Inne zaobserwowane sytuacje, to okresowy charakter ruchu z turbulencjami (wysoka korelacja punktów x_2 i $x_1 = x_2 - X$ po określonym czasie T), a także tworzenie się obszarów dużego ścisku wokół zwężeń w przejściach.

Modelowanie dwupoziomowe

Autorzy artykuły [12], naukowcy z Singapuru oraz Stanów Zjednoczonych, wskazali i rozwinęli nowe podejście do, jak sami przyznają, bardzo trudnego zagadnienia modelowania tłumu w różnych warunkach. Stąd właśnie symulacje oparte na scenariuszach „co by było, gdyby" oraz pewnych założonych parametrach tłumu i pojedynczych jednostek. Biorąc pod uwagę skalę obserwacji zjawiska podzielili oni modele zachowań tłumu na dwie kategorie modele: mikro oraz makro. Każdy z nich stosowany jest do analizy tłumu na innym poziomie ogólności.

W skali mikro badane są indywidualne cechy jednostek znajdujących się w tłumie, ich psychologiczne i społeczne zachowania (np. emocje i intencje), komunikacja między jednostkami oraz podejmowanie indywidualnych decyzji. Wynikiem tych badań jest opis, w jaki sposób każdy z uczestników tłumu zachowuje

się w kolejnych odstępach czasu trwania symulacji. W makroskali tłum badany jest jako całość, a więc określane są czynniki zewnętrzne lub charakterystyczne dla danego środowiska wpływają na jego zachowanie. Wynikiem tego podejścia jest ogólny proces zachowania się tłumu. Typowym modelem dla makroskali jest model przepływowy, natomiast dla mikroskali - model agentowy.

Model przepływowy jest przydatny w symulowaniu tłumu, gdy brane są pod uwagę jego cechy ogólne. Tłum w tym ujęciu charakteryzowany jest przez prędkość i gęstość w punkcie (x, y) (znajdującym się w układzie współrzędnych związanym z pomieszczeniem, wewnątrz którego tłum się znajduje) oraz przepływ, który może być obliczony na podstawie tych wielkości. **Gęstość** jest określana jako ilość pieszych wewnątrz obszaru związanego z punktem (x, y) w czasie t. Można też określić średnią gęstość jako stosunek ilości pieszych na danym obszarze do jego wielkości. **Prędkość** tłumu w punkcie (x,y) i czasie t jest po prostu wypadkową prędkością pieszych w danym punkcie i czasie. **Przepływ** jest parametrem określającym, jak wielu pieszych przemieszcza się wzdłuż konkretnej krzywej w czasie t. Jest on definiowany jako całka po krzywej funkcji będącej złożeniem funkcji gęstości i prędkości tłumu. Znak wyniku określa, w którą stronę przemieszcza się tłum po krzywej.

Model agentowy używany jest wszędzie tam, gdzie istotniejsze są symulacje zachowania się pojedynczych jednostek niż całości. Autorzy przyjmują, że pojedynczy agent (pieszy) nie podejmuje decyzji o dalszych ruchach samodzielnie, lecz powiela zachowania jednostek otaczających. Agent w tym ujęciu jest charakteryzowany jedynie przez kilka czynności, które wykonuje w procesie znajdowania kierunku poruszania się:

- wyszukiwanie celu - agent przyjmuje cel największej liczby agentów wokół niego, jako swój własny;
- wyszukiwanie lidera - agent wyszukuje w swoim otoczeniu lidera, za którym będzie podążał. Aby inny agent mógł zostać wybrany liderem, musi spełniać następujące warunki: posiada taki sam cel, jak agent; kąt między kierunkiem agenta a potencjalnego lidera musi być jak najmniejszy; odległość między agentem a potencjalnym liderem musi być najmniejsza spośród wszystkich z tym samym kątem;
- podążanie - jeśli agent znajdzie lidera, wykonywany jest schemat podążania za nim;
- liderowanie - jeśli nie zostanie znaleziony żaden lider, agent podąża wzdłuż preferowanej ścieżki bez powielania zachowania kogokolwiek;
- omijanie sąsiadów - cały czas agent realizuje czynność polegającą na omijaniu sąsiadów (zostało to zaimplementowane w prosty sposób, korzystając z własności narzędzia zastosowanego do symulacji);
- wybieranie obszaru o mniejszej gęstości - agent stara się wybrać obszar o mniejszej gęstości, niż ten w kierunku którego skierowany jest wektor jego prędkości. Jeśli zostanie znaleziony taki obszar, wektor prędkości ustawiany jest w jego kierunku, zachowując jednocześnie poprzednią szybkość.

Oba opisane powyżej modele mają zalety, jak również wady. Aby do maksimum wykorzystać oferowane przez nie możliwości zdecydowano się na zastosowanie podejścia łączonego, tzw. modelu o wielu rozdzielczościach (ang. *multi-resolution model*). Wyniki symulacji dla takiego podejścia dzielą charakter wyników zarówno dla modelu przepływowego, jak i agentowego. Dzieje się tak dlatego, że każde z tych podejść wykorzystywane jest w określonych warunkach:

- model przepływowy - symuluje i wizualizuje tłum w stabilnych warunkach (wynikiem symulacji jest ogólna tendencja zmian charakterystyk opisujących ogół tłumu),
- model agentowy - symuluje i wizualizuje pojedyncze jednostki z tłumu.

Przełączanie się między modelami następuje płynnie. Przez większość czasu trwania symulacji stosuje się model przepływowy. Jednakże w wypadku zaistnienia nagłego zdarzenia (eksplozji bomby, wybuchu pożaru i innych) symulacja przełącza się na model agentowy, z którego - po ustabilizowaniu się sytuacji, znów wchodzi w model przepływowy.

Aby modele nie działały w oderwaniu od siebie zaproponowano system komunikacji między tymi „warstwami" (różniącymi się poziomem ogólności), opierający się o **agregację** i **deagregację** danych. Deagregacja jest transformacją informacji z modelu przepływowego na agentowy. Model agentowy potrzebuje takich parametrów jak: liczba agentów, ich położenia oraz prędkości. Można je wyprowadzić z gęstości oraz wielkości komórki obszaru. Agregacja jest transformacją w odwrotnym kierunku, z danych na poziomie agentowym do poziomu przepływowego. Parametrami wynikowymi tej operacji są: gęstość i prędkość tłumu w punkcie (x, y) oraz czasie t (zgodnie z modelem przepływowym).

Połączenie dwóch podejść do symulacji zachowania tłumu daje dobre rezultaty - tworzy sprawnie działający oraz prawdopodobny obraz rzeczywistego tłumu. Dużą zaletą takiego podejścia jest znaczne zmniejszenie czasu wykonywania obliczeń (tłum jest głównie modelowany za pomocą przepływu, a model agentowy włączany jest jedynie w sytuacjach krytycznych). Model połączony jest więc dobrym punktem wyjścia do dalszych rozważań na temat symulacji zachowania się tłumu.

Znajomość budynku i rola wytrenowanych liderów

Większość modeli opisujących ruch tłumu w czasie ewakuacji zakłada pełną znajomość budynku i jego dróg ewakuacyjnych przez każdą z jednostek. Założenie to pozwala na ucieczkę z wykorzystaniem tras, które podczas normalnego ruchu są rzadko używane lub całkowicie zamknięte. Obserwacja rzeczywistych sytuacji pokazuje jednak, że ludzie preferują poruszanie się najlepiej znanymi ścieżkami - w wypadku pracowników lub mieszkańców budynku będą to drogi, którymi codziennie wchodzą i wychodzą z budynku, zaś dla gości danego miejsca będzie to często jedyna droga, którą poznali w czasie wejścia do budynku.

Opisane powyżej zachowanie tłumu ma kluczowy wpływ na przebieg ewakuacji, gdy preferowana droga zostanie odcięta poprzez czynnik wymuszający ewakuację, np. pożar lub wybuch. Nagły brak drogi ucieczki spowodować może

śmiertelną w skutkach panikę. Uniknąć jej można dzięki komunikacji między jednostkami i podążaniem za osobami dobrze znającymi topologię budynku i możliwe drogi ewakuacji. W zależności od miejsca i sytuacji mogą nimi być kierownicy biur, obsługa stadionu czy też przybyłe na miejsce służby mundurowe.

Zagadnienia te rozważono w pracy [13], gdzie zaproponowano podejście rozbudowujące niskopoziomowy model lokalnego ruchu (np. model Helbinga, [8]) o wyższy poziom abstrakcji związany z:

- poruszaniem się po preferowanych ścieżkach,
- poszukiwaniem nowych dróg w sytuacji zablokowania wybranej drogi ucieczki,
- przekazywanie informacji między osobami w tłumie,
- zróżnicowaną samodzielność jednostek.

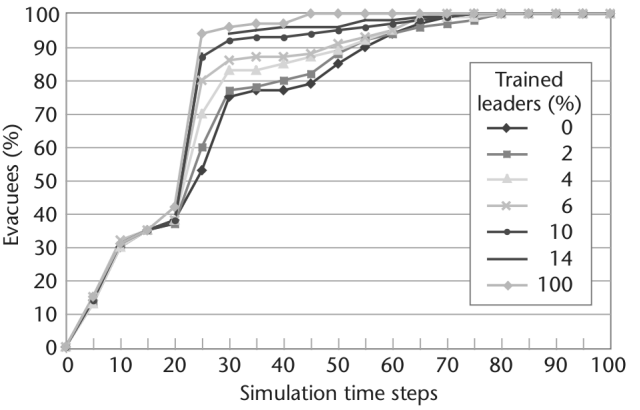
Poszukiwanie nowych dróg jest procesem budowania mapy budynku w oparciu o otrzymywane od innych jednostek informacje (np. o zablokowanych przejściach), podążanie za innymi osobami, a także samodzielną eksplorację pomieszczeń (w czasie ruchu wiedza jednostki na temat najkrótszej trasy pozwalającej opuścić budynek może być znacznie poszerzona poprzez znaki informujące o wyjściach ewakuacyjnych). Nowatorskim elementem tego podejścia jest zróżnicowanie osób pod względem charakteru. Wyróżniono w nim trzy podstawowe typy jednostek:

- wytrenowani liderzy - posiadają pełną wiedzą na temat dróg ewakuacji w budynku, wpływają na innych, pomagając im opuścić budynek,
- niewytrenowani liderzy - dobrze znoszą stres związany z ewakuacją, nie wpadają w panikę, systematycznie szukają drogi ucieczki (algorytm przeszukiwania wgłąb), mogą kierować lokalnymi grupami osób,
- osoby zależne - nie znają dróg ewakuacyjnych, łatwo wpadają w panikę, tracąc możliwość podejmowania własnych decyzji, ich działanie ogranicza się często do naśladowania grupy.

Autorzy pracy [13] podjęli próbę zbadania wpływu poszczególnych czynników wysokiego poziomu na czas ewakuacji, wykorzystując do tego celu oprogramowanie bazującego na modelu ruchu Helbinga. Seria wykonanych przez nich symulacji (ruch agentów odbywał się w 100 połączonych pomieszczeniach, zaś ich liczba wahała się od 20 do 200) pozwoliła wyciągnąć następujące wnioski (zobacz rysunek 3.1):

- komunikacja między agentami pozwala dwukrotnie skrócić czas ewakuacji,
- jeśli agenci mogą przekazywać sobie informacje, zwiększanie ich liczby skraca czas ewakuacji,
- powyższy efekt zanika, gdy liczba agentów jest zbyt duża w stosunku do liczby i szerokości przejść - zaczynają występować zatory przy drzwiach,
- zwiększanie liczby wytrenowanych liderów pozwala dwukrotnie skrócić czas ewakuacji, przy czym już niewielki ich procent znacznie skraca czas ewakuacji.

Opisane powyżej wyniki potwierdzają duży wpływ czynników wysokiego poziomu takich jak komunikacja między agentami, stopień znajomości budynku



Rys. 3.1: Wpływ liczby wytrenowanych liderów na czas ewakuacji (źródło: [13]).

czy też samodzielność agentów na czas ewakuacji. Dodanie ich do klasycznych metod agentowych bazujących na oddziaływaniach między jednostkami pozwala na uzyskanie modelu pełniej oddającego rzeczywiste zjawiska występujące w tłumie.

3.3. Metody symulacji ewakuacji

Różnorodność modeli zachowania się tłumu i paniki oraz złożoność modelowanych zjawisk pociąga za sobą konieczność wykorzystania odpowiednich narzędzi programowych. Dobrą platformą do prowadzania badań są komercyjne rozwiązania [14]. Istnieją też otwarte rozwiązania. Niniejszy podrozdział zawiera krótki przegląd wybranych modeli oraz narzędzi programowych.

3.3.1. Przykładowe narzędzia i modele

Środowiska wykorzystujące model przepływowy

Przykładem środowiska, w którym wykorzystano model przepływowy do badania przebiegu ewakuacji w budynku, jest EVACNET4. Środowisko to pozwala zamodelować otoczenie jako sieć połączonych węzłów. Węzły reprezentują struktury fizyczne, takie jak pokoje, schody, hole, korytarze. Są one połączone i stanowią jednolitą strukturę, w której odbywa się ewakuacja. Użytkownik definiuje „zajętość" wszystkich węzłów, m.in. określa, jak wiele osób może zawierać węzeł. Stan węzłów jest opisany przez parametry wysokiego poziomu, np. obecność, interpersonalny rozstaw średni lub inne, jakościowe parametry. Przejście pomiędzy dwoma elementami środowiska jest modelowane jako łuk łączący dwa sąsiadujące węzły. Należy zdefiniować pojemność każdego z łuków oraz wymagany czas przebycia łuku. Użytkownik powinien zdefiniować także stan początkowy - tj. liczbę ludzi w węzłach oraz pozostałe parametry początkowe węzłów oraz liczbę węzłów docelowych, będących wyjściami awaryjnymi. Wynikiem

działania EVACNET4 jest znaleziona najszybsza drogi ewakuacji oraz odpowiedni przepływ ludzi podczas ewakuacji. Brak możliwości zdefiniowania atrybutów agenta uniemożliwia rozważenie socjologicznych aspektów zachodzących w grupie istotnych w każdej ewakuacji, jak np. procesów decyzyjnych. Nieuwzględnienie czynników zmiennych, takich jak zachowanie ewakuowanych, utrudnia uzyskanie wyników zbliżonych do rzeczywistych sytuacji.

W oparciu o przepływowe modele działają także takie środowiska jak EESCAPE i EGRESSPRO. W środowiskach tych aspekt zachowań grupowych i czynniki społeczne zostały pominięte. Symulacja jest sformułowana przy założeniu, że jeśli użytkownik może manipulować szybkością chodzenia i fizycznymi ograniczeniami w przejściach i klatkach schodowych, gęstością i rozmieszczeniem osób w całym budynku, to jest to model wystarczający do oceny przepływu i przebiegu procesu ewakuacji, nie biorąc pod uwagę zachowań społecznych poszczególnych ewakuowanych [15, 16, 14].

Środowiska wykorzystujące model automatów komórkowych

Model komórkowy jest wykorzystany w aplikacji EGRESS. Komórki mają tam kształt ośmiokątów, co pozwala na lepsze estymowanie kierunków przemieszczania się. Niektóre komórki są zdefiniowane jako cele (mogą oznaczać np. wyjście bezpieczeństwa), inne są zdefiniowane jako przeszkody. W pierwszym kroku jest obliczany dystans pomiędzy każdą komórką a najbliższą komórką docelową. Symulacja rozpoczyna się z punktu startowego i dla każdej jednostki możliwy jest do wykonania jeden z czterech scenariuszy ruchu:

- przemieszczenie się bliżej do wyjścia,
- przemieszczenie się dalej od wyjścia,
- przemieszczenie się do komórki o takiej samej odległości od wyjścia,
- pozostanie w miejscu.

W celu ustalenia, która z akcji zostanie podjęta, przypisuje się im odpowiednie prawdopodobieństwo wystąpienia. W modelu tym nie uwzględniono interakcji społecznych oraz zachowania wynikające z przebywania jednostki w tłumie. W oparciu o model komórkowy działają także takie aplikacje jak Pathfinder oraz TIMTEX. [15, 14].

Środowiska wykorzystujące model agentowy

Przykładem oprogramowania działającego w oparciu o model agentowy jest SIMULEX. Pozwala ono na „indywidualizację" ruchu określonych grup. Dla każdej jednostki oznacza to posiadanie zestawu własnych atrybutów oraz możliwość decydowania o własnym tempie przemieszczania się. Dodatkowo uwzględniane są czynniki występujące w algorytmach wyznaczania ruchu, np.: fizyczne ruchy i gesty (kołysanie i skręcanie się ciała), bliskość innych ewakuowanych, kształtu konstrukcji budynku oraz płeć (kobieta lub mężczyzna) i wiek (parametry określone dla osób od 12 do 55 roku życia). Mogą one być postrzega jako czynniki

mające znaczenie społeczne, ale bez oparcia w konkretnej koncepcji lub bez informacji o stosunkach społecznych, kulturowych, czy integracji grupy.

W trakcie symulacji program zakłada obecność racjonalnego agenta, będącego w stanie ocenić optymalną drogę ucieczki, omijać przeszkody fizyczne oraz „wyprzedzać" inne osoby (które stają się przeszkodami). Mapa, po której poruszają się agenci, stanowi dyskretne pole wektorowe. Ruch na nim odbywa się po gradiencie prowadzącym agentów do określonego celu. Aplikacjami wykorzystującymi agentowy model do symulacji ewakuacji są także EXIT89 oraz EXODUS. [15, 14].

3.3.2. Wybrane narzędzia symulacji bazujące na modelu agentowym

Narzędzia programowe pozwalające na symulacje w oparciu o model agentowy powinny uwzględniać indywidualne cechy oraz zjawiska towarzyszące poszczególnym agentom, a także oddziaływania występujących między nimi. Poniżej omówiono dwa takie narzędzia: bibliotekę OpenSteer oraz środowisko NetLogo.

OpenSteer

OpenSteer jest biblioteką języka C++ służącą do modelowania obiektów będących z natury w ruchu. Wraz z biblioteką dostarczane są przykłady symulacji pieszych, ptaków i pojazdów. Każdy ruchomy obiekt w bibliotece OpenSteer powinien dziedziczyć z klasy `AbstractVehicle` oraz `SimpleVehicle`. Klasy te stanowią bazę dla innych, bardziej szczegółowych opisów poruszających się obiektów. Uwzględniają one takie atrybuty obiektów, jak:

- masę obiektu,
- promień okręgu otaczającego obiekt (potrzebne np. do omijania przeszkód),
- wektor prędkości (zawierający trzy składowe: x, y, z),
- wartość wektora prędkości (szybkość),
- maksymalną siłę sterowania,
- maksymalną szybkość obiektu.

OpenSteer dostarcza mechanizmów, dzięki którym zadania przypisane obiektom są prostsze w opisie i realizacji. Dzięki nim (np. funkcji zawartej w klasie `SteerLibraryMixin`, zwracającej siłę sterującą) łatwo zrealizować następujące zachowania:

- wędrowania - funkcja `steerForWander` zwraca siłę sterującą, która realizuje wędrowanie bez celu.
- śledzenia celu - funkcja `steerForSeek` realizuje śledzenie celu podanego jako argument. Wynikiem działania funkcji jest siła sterująca, która zapewnia ustawienie obiektu w kierunku celu oraz jak najszybsze dążenie do niego. Jeśli funkcji tej nie towarzyszą żadne dodatkowe warunki, obiekt przechodzi przez cel i wyliczana jest nowaa siłę sterującą dla tego celu.
- ucieczkę - funkcja `steerForFlee` realizuje ucieczkę od obiektu podanego w argumencie.

- podążanie za ścieżką - zadanie to realizowane jest za pomocą funkcji `steerToFollowPath` oraz `steerToStayOnPath`. Za ich pomocą można uzyskać próbę śledzenia zadanej ścieżki oraz ścisłe poruszanie się po zadanej ścieżce.
- omijanie przeszkód - funkcja `steerToAvoidObstacle(s)` zwraca wektor prędkości, który zapewnia omijanie przeszkody zadanej jako argument funkcji. Jeśli przeszkód jest więcej, w celu ich omijania używa się funkcji `steerToAvoidObstacles`. Jej argumentem jest wektor przeszkód (w znaczeniu biblioteki STL), zaś wartością zwracaną – prędkość zapewniająca ominięcie najbliższej przeszkody.
- omijanie sąsiadów - gdy na scenie znajduje się dużo ruchomych agentów, powinny one mieć możliwość bezkolizyjnego omijania się. Zapewniona to funkcja `steerToAvoidNeighbours`, która zwraca prędkość pozwalającą na ominięcie najbliższego sąsiada. Sąsiedzi do omijania zadawani są jako argument tej funkcji w postaci referencji do obiektu klasy `AVGroup` (będącej odpowiednikiem kontenera `vector` z biblioteki STL).
- ściganie celu - jest to zadanie podążania za innym ruchomym obiektem, realizowane za pomocą funkcji `steerForPursuit`. Funkcja ta zwraca wektor prędkości, który zapewnia ścigania ruchomego obiektu określonego w argumencie.
- utrzymanie szybkości - funkcja `steerForTargetSpeed` realizuje zadanie utrzymania zadanej szybkości. Wartość ta jest wyznaczana wzdłuż osi przód/tył obiektu.

Jak można zauważyć, funkcje wymienione powyżej pozwalają realizować pożądane zachowania prawie automatycznie. Jedynym wymogiem jest dobre zdefiniowanie argumentu wywołanej funkcji (celu, obiektu, ścieżki itd.).

Realizacja konkretnego problemu w bibliotece OpenSteer sprowadza się do stworzenia jednego pliku *.cpp, w którym zawarta jest cała jego instancja. Jest to możliwe dzięki systemowi pluginów, które dołącza się do wywołania w demo dostarczonym wraz z biblioteką. Ważne jest odpowiednie przygotowanie własnego pluginu, czyli dziedziczenie odpowiednich klas (m.in. klas `SimpleVehicle` oraz `PlugIn`).

3.3.3. NetLogo

NetLogo, którego twórcą jest Uri Wilensky, stanowi język programowania wraz z całym zintegrowanym środowiskiem zawierającym narzędzia do przeprowadzania symulacji połączonych z wizualizacją. Bazuje on na utworzonym w 1967 r. języku programowania Logo i jest obecnie rozwijany przez CCL (ang. *Center for Connected Learning and Computer-based Modeling*), należący do Northwestern University w Evanston, Illinois [17]. Język ten ma umożliwiać łatwą pracę początkującym, a jednocześnie sprostać potrzebom zaawansowanych użytkowników (zasada ta określona jest jako „niski próg i brak sufitu").

Do głównych zalet NetLogo należy możliwość łatwego tworzenia modeli agentowych, w których może dochodzić do interakcji pomiędzy poszczególnymi zaprogramowanymi elementami środowiska (rzeczywistości). Dzięki temu można eksperymentalnie oceniać konsekwencji takich oddziaływań. Istotną ce-

chą Netlogo jest możliwość rozwoju danego układu w czasie. Oprogramowanie to znalazło szerokie zastosowanie, między innymi przy budowie modeli symulacyjnych w fizyce i biologii, a także w naukach społecznych. We wszystkich tych przypadkach podstawowymi elementami modelowanego układu są cząstki lub osobniki (agenci) [18].

Środowisko NetLogo działa w oparciu o język programowania Java. Możliwe jest jego uruchomienie zarówno w systemie operacyjnym Windows, jak również na innych platformach (Mac i Linux). Dodatkową zaletę środowiska NetLogo jest licencja pozwalająca na bezpłatne użycie oraz szczegółowa dokumentacja wraz z podręcznikami użytkownika, dostępna na stronie twórców [17].

Świat w NetLogo jest zbudowany z agentów, którzy mogą wykonywać określone instrukcje. Istnieją cztery rodzaje agentów:

- **turtles** – są to agenci, którzy poruszają się w dwuwymiarowym świecie pomiędzy określonymi elementami zbudowanymi z agentów patches;
- **patches** – są to elementy dwuwymiarowego świata, po których poruszają się agenci turtles (pozwalają one rozszerzyć model agentowy o cechy modelu komórkowego i przepływowego);
- **links** – są to połączenia pomiędzy dwoma agentami turtles;
- **observer** – agent, który nie posiada określonego miejsca, ale może obserwować świat zdefiniowany poprzez agentów patches, w którym poruszają się turtles, oraz wydawać instrukcje agentom.

Główną areną symulacji jest czarny prostokąt reprezentujący płaską, dwuwymiarową przestrzeń odzwierciedlającą modelowaną rzeczywistość. Przestrzeń ta jest zbudowana z określonej liczby agentów typu patches. W opcjach `Settings` możliwe jest ustawienie zarówno liczby występujących agentów patches, jak również ich wielkości. Program pozwala na ustalenie położenie środka układu współrzędnych względem którego zostaną wyznaczone współrzędne każdego agenta typu patches.

Agentom można przypisać różne kolory (dla celów wizualizacji) i różne cechy (które z czasem mogą ulegać zmianie). Pozwala to na budowę złożonych modeli i przeprowadzanie skomplikowanych symulacji, zarówno w środowisku statycznym, jak i dynamicznym. Agenci turtles mogą mieć określony wiek, rozmiar, kształt, masę ciała, płeć i dowolne inne cechy. Pozwala to na zdefiniowanie wielu różnych agentów oraz różnicowanie ich cech w obrębie rodzajów.

Istnieją instrukcje, które pozwalają na wymuszenie określonego działania lub zmiany cech agentów typu patches i turtles w czasie. W ten sposób możliwe jest zdefiniowanie, między innymi, działań podejmowanych przez agentów.

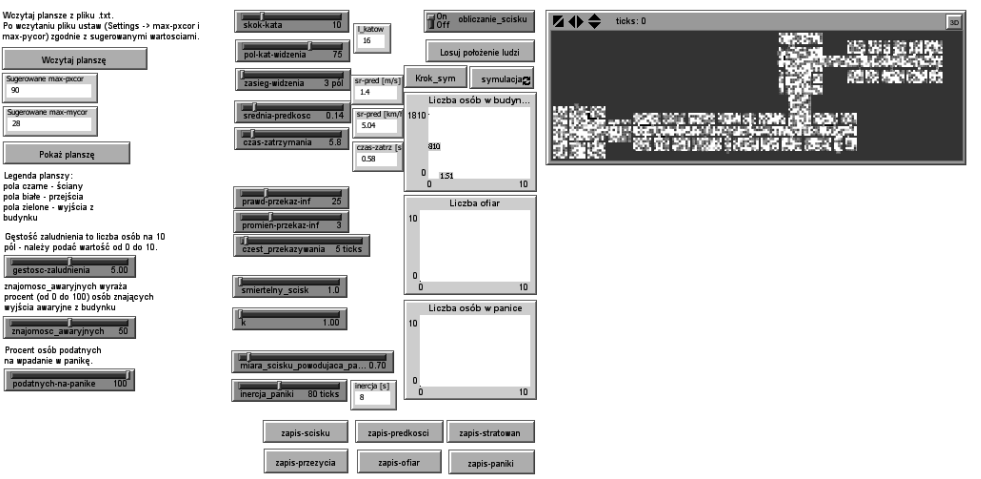
Czas jest zdefiniowany jako jeden krok, w którym zostaną wykonane jednorazowo instrukcje w stosunku do wszystkich obecnych w modelu agentów. Czas ten jest określony jako *tick*, a jego przebieg oznacza odliczanie kolejnych kroków życia agentów.

3.4. Wykorzystane środowisko badań symulacyjnych

Zarówno OpenSteer jak i NetLogo dają możliwości symulowania zachowań podczas ewakuacji, w tym wybuchu paniki. W obydwu narzędziach pewne elementy (związane z samym ruchem lub też interfejsem graficznym) są dostarczone jako gotowe rozwiązania. OpenSteer stanowi jednak jedynie bibliotekę języka C++, natomiast NetLogo stanowi kompletne i przenośne środowisko symulacyjne, które pozwala w łatwy sposób zamodelować odpowiednie elementy środowiska oraz agentów. Z powyższych powodów zdecydowano się na wykorzystanie w badaniach środowiska NetLogo.

3.4.1. Logika zaimplementowana w aplikacji

W celu przeprowadzenia symulacyjnych badań dotyczących wpływu różnych czynników na występowanie zjawiska paniki, a co za tym idzie, zaburzenie procesu ewakuacji, stworzono aplikację w środowisku Netlogo. Główne okno tej aplikacji przedstawiono na rysunku 3.2. Jej działanie opiera się na podejściu zaprezentowanym w artykule [10]. Polega ono na tym, że decyzje podejmowane przez agentów są wyznaczane w oparciu o dane wzrokowe. Agenci analizują widoczną dla siebie przestrzeń i wybierają kierunek prowadzący do celu, przy jednoczesnym unikaniu przeszkód. Dodatkowo zaimplementowany został mechanizm częściowej znajomości wyjść ewakuacyjnych i przekazywania informacji o nich między agentami.



Rys. 3.2: Główne okno aplikacji z kontrolkami sterującymi i widokiem areny.

W aplikacji wykorzystano dwa podstawowe rodzaje agentów dostarczonych przez NetLogo: patches (do modelowania pomieszczeń) i turtle (do modelowania ludzi). Posłużono się przy tym zarówno parametrami wbudowanymi, pozwalają-

cymi na zdefiniowanie określonych właściwości agentów, jak również cechami dodatkowymi (zaimplementowanymi specjalnie na potrzeby symulacji).

Agenci typu patches

NetLogo domyślnie pozwala na zdefiniowanie zarówno położenia agenta jak i jego koloru. Rolę i znaczenie agentów typu patches w przeprowadzanych symulacjach odzwierciedlały kolory:

- biały – przestrzeń dostępna dla agentów turtle,
- czarny – ściany, oraz przestrzeń niedostępna dla agentów turtle,
- zielony – wyjścia.

Wygląd areny w danej symulacji wynikał z odpowiedniego ułożenie agentów patches. Ułożenie to określano, przypisując agentom położenie (x,y). Dodatkowo, agenci typu patches zostali opisani następującymi parametrami:

- `exits_dists` - lista zawierająca wartości odległości do kolejnych wyjść ewakuacyjnych,
- `miara_scisku` - ważona miara ścisku panująca na polu w danej chwili,
- `sumaryczna_miara_scisku` - sumaryczna miara ścisku panująca na polu w danej chwili,
- `miara_predkosci` - ważona miara prędkości w danej chwili,
- `sumaryczna_miara_predkosci` - sumaryczna miara prędkości w danej chwili,
- `stratowanie` - liczba stratowań na danym obszarze.

Agenci typu turtle

Agenci turtle, stanowiący w symulacji jednostki poddane ewakuacji, zostali opisani wbudowanymi parametrami, takimi jak: położenie (zadane przez współrzędne x,y) oraz orientacja (widoczna na ekranie jako kierunek strzałki). Dodatkowo, zdefiniowano dla każdego agenta następujące parametry:

- `radius` - promień danego agenta (proporcjonalny do masy wylosowanej na początku symulacji, promień=$\frac{masa}{320}$),
- `kier_do_wyjscia` - kierunek prowadzący do najbliższego wyjścia (wyliczony na podstawie algorytmu czoła fali),
- `odl_do_przeszkod` - lista wartości odległości do przeszkód dla danych kątów,
- `heurystyka` - lista wartości heurystyki na kolejnych kątach,
- `kat_ruchu` - kąt, pod jakim porusza się agent,
- `podatny_na_panike` - parametr określający, czy dany agent może wpaść w panikę,
- `w_panice` - parametr określający, czy dany agent obecnie znajduje się w stanie paniki,
- `v_i` - losowana na początku podstawowa prędkość poruszania się agenta (na jej wartość wpływa parametr średniej prędkości),
- `v` - prędkość, z jaką agent porusza się w danej chwili,
- `v_x` - składowa prędkości `v` w kierunku x,

– `v_y` - składowa prędkości `v` w kierunku y,
– `d_h` - odległość od przeszkody, w wybranym kierunku ruchu,
– `a_x` - obecne przyspieszenie w kierunku x,
– `a_y` - obecne przyspieszenie w kierunku y,
– `fx` - wartość siły w kierunku x (dodatnia w prawo),
– `fy` - wartość siły w kierunku y (dodatnia w górę),
– `scisk` - suma norm wszystkich sił oddziałujących z zewnątrz na agenta,
– `poprzednie_orientacja` - orientacja agenta w poprzedniej iteracji.

Przebieg działania aplikacji

W każdym kroku symulacji, dla każdego agenta wykonywano następujące działania:

```
 1. tick
 2. if ( (ticks mod czest_przekazywania ) = 0);
    [ przekaz_informacji ]
 3. znajdz_pozadany_kierunek
 4. analizuj_pole_widzenia
 5. wybierz_opt_kier
 6. wylicz_predkosc
 7. analizuj_zderzenia
 8. wylicz_scisk_w_punkcie
 9. analiza_paniki
10. analiza_sil a
11. wylicz_pred_w_punkcie
12. test-czy-koniec s
13. if not any? turtles [ stop ];;
    end
```

W kolejnych krokach wykonywano następujące operacje:

1. W pierwszym kroku, za pomocą funkcji NetLogo `tick` zwiększano liczbę *ticków* (funkcja ta służy w środowisku NetLogo jako zegar i pozwala na symulowanie upływu czasu)
2. Następnie sprawdzano warunek określający możliwość przekazania informacji dotyczących wyjść awaryjnych. Każdy agent turtle z zadanym prawdopodobieństwem może uzyskać informację o wyjściach awaryjnych od wszystkich agentów rozmieszczonych w określonym promieniu od niego. Częstotliwość przepływu informacji, jak i promień jej rozpowszechniania, jest zadawana na początku symulacji. Jeżeli w aktualnym momencie należy przekazać informację, wywoływana jest funkcja `przekaz_informacji`.
3. W bieżącej iteracji każdy agent wybierał kierunek prowadzący do wyjścia. Agent do wyboru ma jeden z 8 kierunków. Wyboru dokonuje poruszając się po malejącym gradiencie odległości do najbliższego znanego mu wyjścia. In-

formacje o odległości do wyjścia odczytuje, korzystając z parametru agenta patch, na którym aktualnie stoi, oraz z 8 sąsiednich agentów typu patch.
4. Zgodnie z heurystycznym podejściem prezentowanym w artykule [10], każdy agent turtle analizował swoje pole widzenia. Analiza ta wykonywana jest ze skokiem o dany kąt oraz dla zadanej dla agenta odległości. Wynikiem tej analizy jest wartość heurystyki wyznaczona dla każdego kąta.
5. Następnie każdy agent turtle przeglądał wartość heurystyki dla poszczególnych kątów i wybierał ten kąt, dla którego heurystyka ma najmniejszą wartość.
6. Każdy agent wyliczał prędkość pozwalającą zatrzymać się mu przed przeszkodą (parametr `czas-zatrzymania`), a także przyspieszenia w osi x i y, które wpływały na zmianę wartości aktualnej prędkości na wartość wyliczoną.
7. Każdy agent analizował możliwość zderzenia z innymi agentami turtle i ścianami. Każde zderzenie powoduje zmianę sił działających na agenta w osiach x i y oraz zwiększenie ścisku.
8. Określano sumaryczny ścisk w punkcie, zliczając ściski sąsiadujących agentów turtle z gaussowską funkcją wagową, zgodnie z równaniem (3.6).
9. Kolejny krok stanowiła analiza zachowań agentów podatnych na panikę. Oceniano, czy agenci podatni na panikę wpadną w nią ze względu na panujący zbyt duży ścisk w ich otoczeniu (ścisk liczony był z funkcją wagową) lub na skutek obecności ofiary w polu widzenia.
10. Analizowano siły pochodzące od zderzeń oraz siły generowane przez agentów na skutek chęci zmiany kierunku ruchu. Dla każdego agenta wyliczano wypadkowe przyspieszenie działające w osi x i y, a następnie zmieniano pozycję agenta. W przypadku, gdy wartość panującego ścisku jest zbyt duża, agent tracił życie.
11. W kolejnym kroku wyliczano średnią prędkość w punkcie, korzystając z wagowej funkcji Gaussa (analogicznie jak dla ścisku w punkcie)
12. Sprawdzano warunek usunięcia agenta z areny (agenci, którzy dotarli do wyjścia, mogli być usunięci z areny).
13. Symulacja kończyła się, gdy na arenie nie pozostał już żaden agent.

3.4.2. Interfejs graficzny zaimplementowanej aplikacji

Aplikacja składa się z dwóch głównych części. Pierwszą część stanowi znajdujący się po lewej stronie rysunku 3.2 panel sterowania, który pozwala m.in. na wykonanie wszelkich operacji potrzebnych do zdefiniowania parametrów agentów, parametrów symulacji oraz pozostałych działań umożliwiających przeprowadzenie symulacji. Dokładny wykaz elementów panelu wraz z ich funkcjami przedstawiono w tabelach 3.1 oraz 3.2. Drugą część aplikacji stanowi arena, na której możliwe jest obserwowanie ruchu agentów podczas symulacji dla zdefiniowanych wcześniej parametrów (prawy - górny róg rysunku 3.2).

Tab. 3.1: Wykaz poleceń aplikacji.

Polecenia umożliwiające przeprowadzenie symulacji	
Wczytaj Planszę	Polecenie pozwala na wczytanie z pliku wcześniej zdefiniowanej areny (sceny symulacji). W pliku zapisano następujące informacje: rozmiar planszy, liczba wyjść, położenie (współrzędne x,y), typ wyjść (0– główne , 1– awaryjne), specyfikacja ścian (kolejne punkty oraz ich kolor, np. 0 0 65 oznacza punkt 0,0 koloru zielonego).
Pokaż Planszę	Polecenie powoduje wyświetlenie wybranej areny w oknie po prawej stronie aplikacji. Wyświetlane są poszczególne punkty zawarte w pliku definiującym scenę, następnie wykonywany jest algorytm czoła fali - każdemu agentowi typu patch przypisuje się listę odległości do kolejnych wyjść. Umiejscowieni na planszy agenci turtle będą poruszać się tak, aby ich odległość do wyjść malała. Punkty znajdujące się poza budynkiem zostają zamalowane na czarno (jest to przestrzeń wyłączona z użytku, aby w podczas losowania położenia agentów, nie występowali oni poza budynkiem).
Losuj położenie ludzi	Polecenie powoduje przypisanie losowych pozycji na planszy wybranej liczbie agentów. Zostaje im przypisana losowa orientacja oraz masa z rozkładu jednostajnego od 60 do 100.
Symulacja	Polecenie powoduje uruchomienie symulacji dla wybranych parametrów.
Krok sym	Polecenie powoduje przejście jednego kroku symulacji.
Polecenia umożliwiające zapis danych do pliku	
Zapis-scisku	Zapis sumy ścisków z całego okresu symulacji dla poszczególnych punktów.
Zapis-predkosci	Zapis sumy prędkości z całego okresu symulacji dla poszczególnych punktów.
Zapis-stratowan	Zapis liczby stratowanych agentów turtle w poszczególnych punktach patches.
Zapis-przezycia	Polecenie pozwala na zapisanie do pliku liczby osób, które żyją i są na planszy w poszczególnych momentach czasu (tzn., że nie są wliczane osoby, które opuściły już budynek).
Zapis-ofiar	Polecenie pozwala na zapisanie do pliku łącznej liczby ofiar w poszczególnych momentach czasu.
Zapis-paniki	Polecenie pozwala na zapisanie do pliku liczby osób, które są w stanie paniki w poszczególnych momentach czasu.

Tab. 3.2: Tabela parametrów symulacji ustawianych w części sterującej aplikacji.

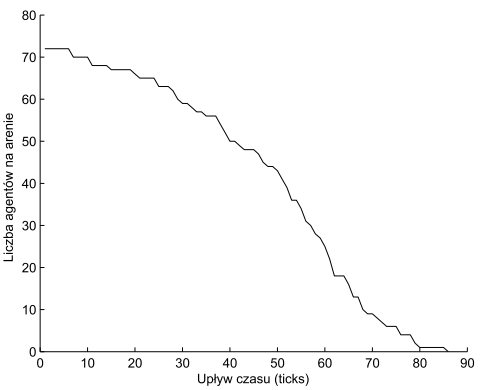
Parametry symulacji	
Gestosc-zaludnienia	Liczba ludzi przypadającą na 10 pól (jednostek przestrzeni). Służy do określenia liczby osób znajdującej się w budynku i może przyjmować wartość z przedziału [0,10].
Znajomosc-awaryjnych	Procent ludzi znajdujących się w budynku znających wyjścia awaryjne. Może przyjmować wartości z przedziału [0,100]).
Podatnych-na-panike	Procent ludzi znajdujących się w budynku, mających tendencję do wpadania w panikę. Przyjęto, że czynnikiem wywołującym atak paniki jest znalezienie się ofiary w obszarze widzenia agenta lub przekroczenie zadanej maksymalnej wartość ścisku w danym punkcie.
skak-kata	Zmiana kąta widzenia (co ile stopni zmienia się kąt).
pol-kat-widzenia	Połowa kąta widzenia agenta.
zasieg-widzenia	Odległość, na którą widzi agent (jego wartość wyrażona jest liczbą pól).
srednia-predkosc	Wartość średniej prędkości agenta (prędkości poruszania się agentów bez obecności przeszków są losowane z rozkładu normalnego o wartości średniej, zadanej położeniem suwaka).
czas-zatrzymania	Czas zatrzymania się agenta (definiuje to bezpieczną odległość od innych agentów, jaką dany agent będzie starał się utrzymać).
prawd-przekaz-inf	Prawdopodobieństwo rozprzestrzeniania się informacji (np. informacji dotyczących instrukcji ewakuacji).
promien-przekaz-inf	Przestrzeń rozprzestrzeniania się informacji.
czest-przekazywania	Częstotliwość rozpowszechniania informacji.
smiertelny-scisk	Wartość ścisku dla danego agenta, przy której następuje jego śmierć.
k	Współczynnik definiujący wartość sił oddziałujących na agentów podczas zderzeń ze sobą lub ścianami.
miara-scisku-powodujaca-panike	Wartość ścisku, przy której agent wpada w panikę (pod uwagę brany jest średni ścisk określony w polu, w którym znajduje się agent).
inercja-paniki	Czas trwania paniki po wystąpieniu bodźca wywołującego jej atak.

3.5. Badania symulacyjne

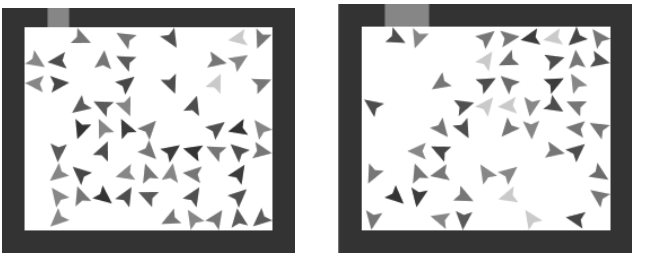
Przeprowadzono dwa rodzaje badań symulacyjnych. Pierwszy rodzaj dotyczył ewakuacji grupy ludzi z pojedynczego pomieszczenia. Mimo zastosowanego uproszczenia symulacje tego typu dostarczają cennych danych, mogących mieć kluczowe znaczenie przy ocenie różnych aspektów związanych z bezpieczną ewakuacją. Drugi rodzaj dotyczył symulacji przeprowadzanych na arenie odzwierciedlającej cechy dużego kompleksu budynków. Takie symulacje pozwalają zaobserwować kluczowe dla bezpiecznej ewakuacji ciągi komunikacyjne i potencjalne zagrożenia.

3.5.1. Symulacja pojedynczego pomieszczenia

W symulacjach tego typu badano wpływu liczby osób w pomieszczeniu na proces ewakuacji. Uzyskaną krzywą, reprezentującą zmianę liczby agentów na arenie wraz z upływem czasu, przedstawiono na rysunku 3.3. Jej kształt był podobny dla wszystkich przeprowadzonych symulacji. Ponieważ położenie ludzi w pomieszczeniu (przedstawionym na rysunku 3.4a) jest zadawane losowo, wyliczono wartość średnią z wielu przeprowadzonych eksperymentów dla zadanych parametrów. Zauważono, iż przy występowaniu braku paniki i średniej prędkości około 5km/h, do pierwszego stratowania jednostki doszło, gdy w sali znajdowało się około 60 osób (parametr gęstość równy 5,5). Przy 70 osobach w sali śmierć ponosiła już większa liczba osób (2-4 osób). Wskaźnik ten utrzymywał się dla większych gęstości. Miejsce newralgiczne, w którym następuje zgniecenie, znajduje się tuż przy wyjściu z pomieszczenia. W przypadku wystąpienia osób podatnych na panikę, i to nawet w niewielkiej liczbie (10%), pierwsze stratowania pojawiają się już przy gęstości równej 5 (dla 50 osób w sali). W przypadku zwiększenia średniej prędkości poruszania się agentów oraz oraz odpowiedniego współczynnika podatnych na panikę, tj. do poziomu ponad 50%, można zaobserwować wzrost liczby ofiar już przy 50 osobach w pomieszczeniu.



Rys. 3.3: Zmiana liczba agentów na arenie wraz z upływem czasu.



(a) Jedno wąskie wyjście. (b) Jedno szerokie wyjście.

(c) Dwa wąskie wyjścia. (d) Dwa szerokie wyjścia.

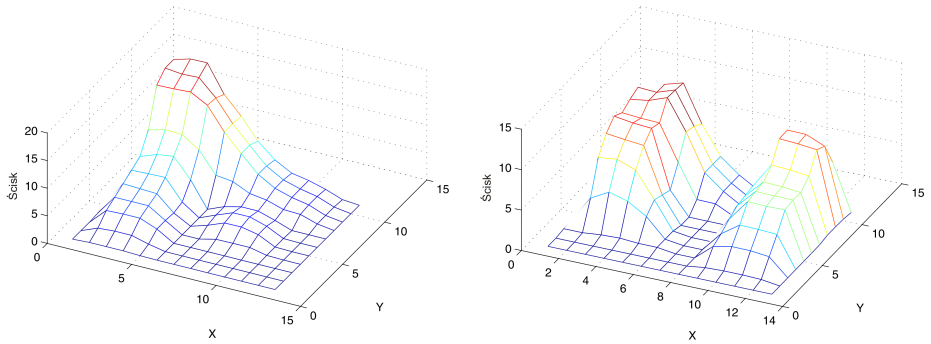
Rys. 3.4: Wygląd pomieszczeń, dla których wykonano symulacje (pomieszczenia różnią się liczbą i szerokością wyjść).

Ze względu na losowość zjawiska paniki oraz zdefiniowanie jej jako losowej modyfikacji kierunku prowadzącego do celu, nie jest możliwe zaobserwowanie jasnych zależności pomiędzy wzrostem liczby osób w panice, a liczbą ofiar. Ponieważ orientacja osób będących w panice jest losowa, zdarza się, iż poprzez ich losowe ruchy zmniejsza się ścisk przy wyjściu (nie są one jednoznacznie zorientowane na poruszanie się zgodnie z malejącą odległością od wyjścia). Ponadto ruch agentów podczas ewakuacji jest niezwykle trudny do przewidzenia, gdyż obok czynników charakterystycznych dla danego agenta, istotne jest także otoczenie w jakim się porusza.

Podczas symulacji zbadano również, jak na proces ewakuacji wpływa dodanie drugiego, symetrycznie położonego wyjścia z pomieszczenia (rysunek 3.4c). Przy braku paniki i średniej prędkości około 5km/h do pierwszego stratowania jednostki doszło, gdy w sali znajdowało się ponad 100 osób. W takim przypadku urazu doznała 1 osoba.

Zastosowanie jednego wyjścia dwukrotnie szerszego od przyjętego w pierwszych symulacjach (rysunek 3.4b), nie poprawiło sytuacji. Pierwsze stratowania nastąpiły już przy 50 osobach w pomieszczeniu. Może to być spowodowane umieszczeniem wyjścia z boku sali, przez co większość agentów dąży do niego z jednej strony. Natomiast przy zastosowaniu dwóch dwukrotnie większych od standardowego wyjść (rysunek 3.4d), nie dochodzi do wystąpienia ofiar nawet, gdy w pomieszczeniu znajduje się ponad 100 osób podatnych na panikę. Dzięki zastosowaniu odpowiednio dużych wyjść ewakuacja przebiega sprawnie i nie występują warunki sprzyjające wybuchowi paniki.

Badania wykazały, iż na przebieg ewakuacji znaczny wpływ ma topologia pomieszczenia i rozmieszczenie oraz wielkość wyjść awaryjnych. Na rysunku 3.5 pokazano różnicę między rozkładem ścisku w wypadku jednego i dwóch wyjść. Już dla prostego przypadku pustego pomieszczenia (wyidealizowanego, bez znajdujących się w środku przeszkód) występuje duża zmiana w przypadku zwiększenia liczby wyjść.

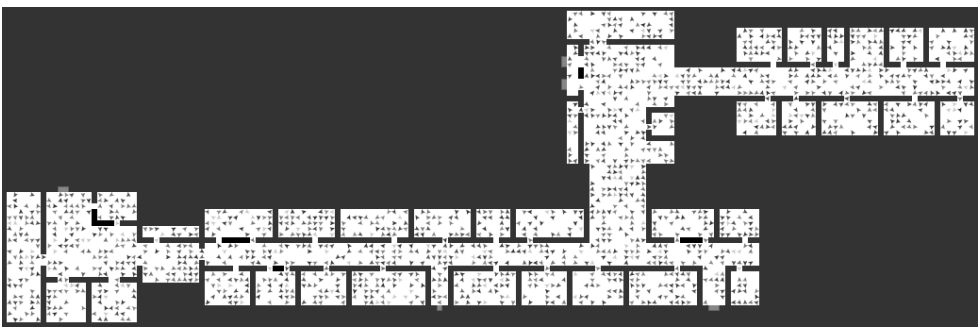


(a) Pomieszczenie z jednym wyjściem (b) Pomieszczenie z dwoma wyjściami

Rys. 3.5: Rozkład ścisku dla pomieszczeń z różną liczbą wyjść.

3.5.2. Symulacja kompleksu budynków

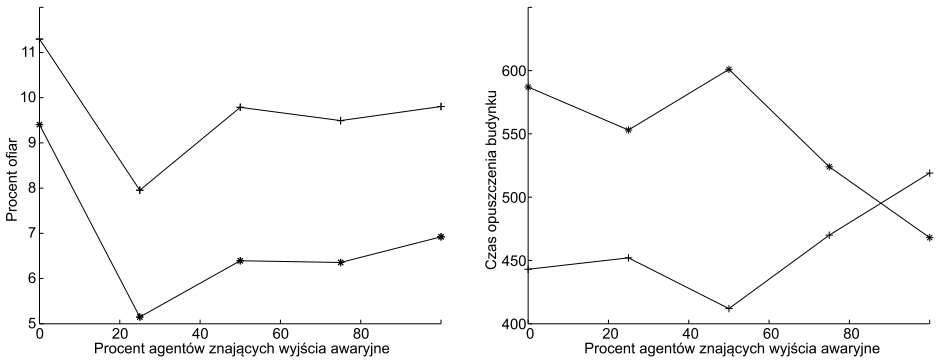
Dalsze symulacje przeprowadzono dla kompleksu budynków, o układzie pomieszczeń jak na rysunku 3.6. Kompleks budynków ma trzy wyjścia główne (z czego dwa są położone bardzo blisko siebie) oraz dwa awaryjne. Poszczególne budynki kompleksu są połączone wąskimi korytarzami, co może mieć wpływ na przebieg ewakuacji. W przeprowadzonych badaniach skupiono się między innymi na zagadnieniu znajomości wyjść awaryjnych, liczbie osób w budynku, a także zidentyfikowaniu kluczowych ciągów komunikacyjnych.



Rys. 3.6: Układ pomieszczeń kompleksu budynków wykorzystany w symulacjach.

W pierwszym przeprowadzonym eksperymencie badano wpływu znajomości wyjść ewakuacyjnych, zarówno gdy nie występuje przekazywanie informacji, jak i gdy ma ono miejsce (z 20% prawdopodobieństwem, co 5 cykli). Wartości pozostałych kluczowych parametrów były następujące: gęstość zaludnienia równa 3, 100% osób podatnych na panikę, średnia prędkość równa 5 km/h.

Oczekiwanym wynikiem było zwiększenie bezpieczeństwa ewakuacji i skrócenie jej czasu wraz ze wzrostem liczby osób znających położenie wyjść ewakuacyjnych. Agenci znający wyjścia ewakuacyjne są zdolni do wybrania najkrótszej drogi do wyjścia z budynku. Przeprowadzone symulacje dały jednak odwrotny wynik. Na rysunku 3.7 pokazano zaobserwowaną względną liczbę ofiar oraz czas ewakuacji. Analiza pierwszego z nich pozwala zauważyć, że najbezpieczniejszą ewakuację gwarantuje zaledwie 25% agentów znających położenie wyjść awaryjnych. Gdy wiedzę o ich położeniu posiadają wszyscy agenci, liczba ofiar jest tylko nieznacznie niższa niż w wypadku całkowitego braku znajomości wyjść awaryjnych. Powyższa obserwacja jest prawdziwa zarówno dla włączonego mechanizmu przekazywania informacji, jak i przy jego wyłączeniu (przy czym wyłącenie sprzyjało bezpieczniejszej ewakuacji - liczba ofiar byłą mniejsza, lecz tendencja była taka sama). Wykres obrazujący czas ewakuacji dowodzi, że mimo mniejszej śmiertelności, brak przekazywania informacji wydłuża czas opuszczania budynku przez agentów - podobne wyniki uzyskano jedynie dla 100% znajomości wyjść ewakuacyjnych, gdy przekazywanie informacji było zbędne.



Rys. 3.7: Wpływ znajomości wyjść awaryjnych na liczbę ofiar i czas ewakuacji (krzyżyki - symulacja z włączonym mechanizmem przekazywania informacji o wyjściach awaryjnych, gwiazdki - brak przekazywania informacji).

Wyniki te są tylko pozornie zaskakujące. Analizując średnie wartości ścisku i prędkości na poszczególnych polach można je prosto uzasadnić. Gdy znajomość wyjść ewakuacyjnych nie jest pełna, większość agentów podąża do szerszych wyjść głównych, zaś pozostała część do, zazwyczaj węższych, wyjść ewakuacyjnych. Ciągi komunikacyjne są bardziej równomiernie obciążone, przez co występuje mniejsze zagęszczenie ludzi i mniejszy ścisk, który mógłby skutkować powstaniem paniki i stratowaniem. Przy 100% znajomości wyjść ewakuacyjnych wielu agentów kieruje się w ich stronę, powodując tym samym mniejszy tłok

przy wyjściach głównych kosztem wyjść ewakuacyjnych, przy których pojawia się niebezpieczne zagęszczenie ludzi. Naturalnym skutkiem braku znajomości wyjść ewakuacyjnych jest jednak dłuższy czas ewakuacji. Wielu agentów kieruje się w stronę oddalonych od nich wyjść głównych, pomijając nieznane im wyjścia ewakuacyjne znajdujące się po drodze.

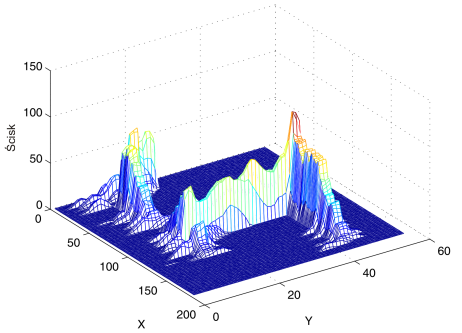
Na rysunkach 3.8a - 3.8f przedstawiono rozkład ścisku, liczby startowań i prędkości podczas symulacji w kompleksie budynków, dla 0% i 100% znajomości wyjść ewakuacyjnych. Można na nich zauważyć znaczące różnice w obciążeniu ciągów komunikacyjnych. Gdy używane są jedynie wyjścia główne, największy ścisk (oraz średnia prędkość) występuje przy wyjściach oraz na drodze do dwóch górnych wyjść głównych. Przy znajomości wyjść ewakuacyjnych przejście do górnych wyjść jest praktycznie nieużywane, a dużych ruch pojawia się w pobliżu obydwu wyjść ewakuacyjnych. Analiza stratowań pokazuje podobną prawidłowość. Przy braku znajomości awaryjnych dróg ewakuacji stratowania następują głównie na drodze do górnych wyjść ewakuacyjnych. Gdy dodatkowe wyjścia są znane, najwięcej ofiar pojawia się w ciasnym korytarzu prowadzącym do środkowego wyjścia ewakuacyjnego.

Kolejne eksperymenty dotyczyły wpływu liczby agentów (gęstości zaludnienia) na liczbę ofiar. Wartości pozostałych kluczowych parametrów były następujące: 0% osób podatnych na panikę, 100% znajomość wyjść awaryjnych, średnia prędkość równa 5 km/h. Wyniki przeprowadzonych symulacji pokazano na rysunku 3.9a. Zgodnie z oczekiwaniami, wzrost liczby ludzi w budynku powoduje zwiększenie liczby ofiar podczas ewakuacji. Ciekawą obserwacją jest charakter tej zależności. Nie jest liniowy, lecz wykładniczy - przy dużej liczbie osób w budynku liczba ofiar ewakuacji bardzo gwałtowanie rośnie.

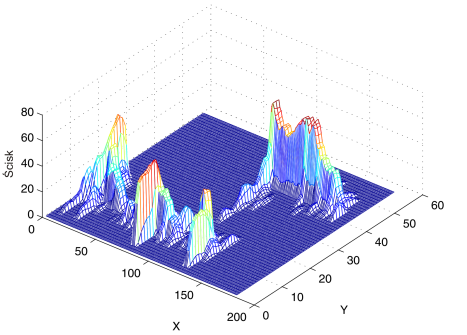
Ostatnim tematem badań był wpływ średniej prędkości z jaką chcą się poruszać agenci na liczbę ofiar podczas ewakuacji. Wartości pozostałych kluczowych parametrów były następujące: gęstość zaludnienia równa 3, 100% znajomość wyjść awaryjnych, podatność na panikę równa 0% lub 100%. Wyniki zaprezentowano na rysunku 3.9b. Jak widać, większą liczbę ofiar notuje się gdy występuje panika dla mniejszych prędkości. Gdy agenci zaczynają się poruszać z dużymi prędkościami (ewakuacja w biegu), obecność paniki zaczyna mieć drugorzędne znaczenie - liczba ofiar staje się podobna.

3.6. Podsumowanie

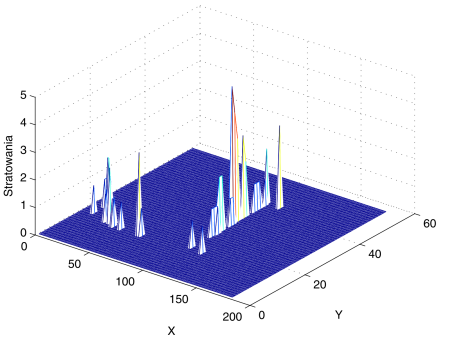
Zgodnie z przewidywaniami problem modelowania paniki okazał się skomplikowany. Ze względu na losowość zachodzących zjawisk dokładne przewidzenie przebiegu ewakuacji nie było możliwe. Mimo tego badania symylacyjne pozwoliły zidentyfikować istotne czynniki wpływające na przebieg ewakuacji. Okazało się, że zależy on od liczba wyjść. Już w najprostszym przypadku pojedynczego pomieszczenia dołączenie dodatkowego wyjścia powoduje znaczą poprawę czasu ewakuacji i zmniejsza liczbę ofiar. Rozmieszczenie drzwi w różnych częściach sali powoduje równomierne obciążenie ciągów komunikacyjnych, a co za tym idzie,



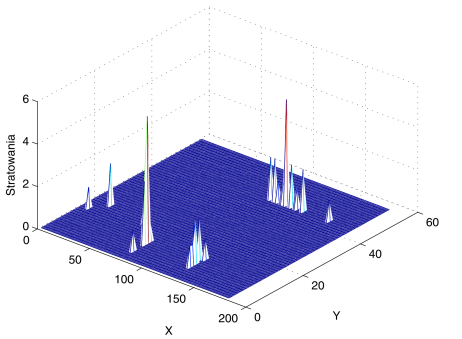
(a) Rozkład ścisku - 0% znajomość wyjść awaryjnych.



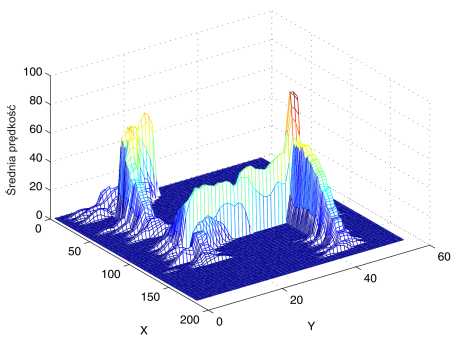
(b) Rozkład ścisku - 100% znajomość wyjść awaryjnych.



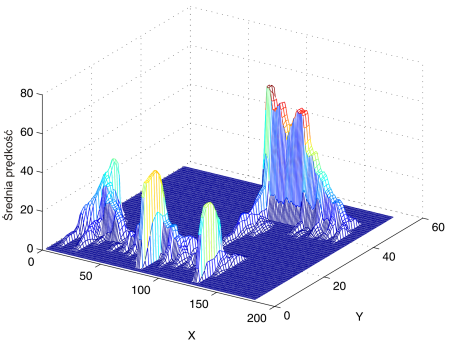
(c) Rozkład stratowań - 0% znajomość wyjść awaryjnych.



(d) Rozkład stratowań - 100% znajomość wyjść awaryjnych.



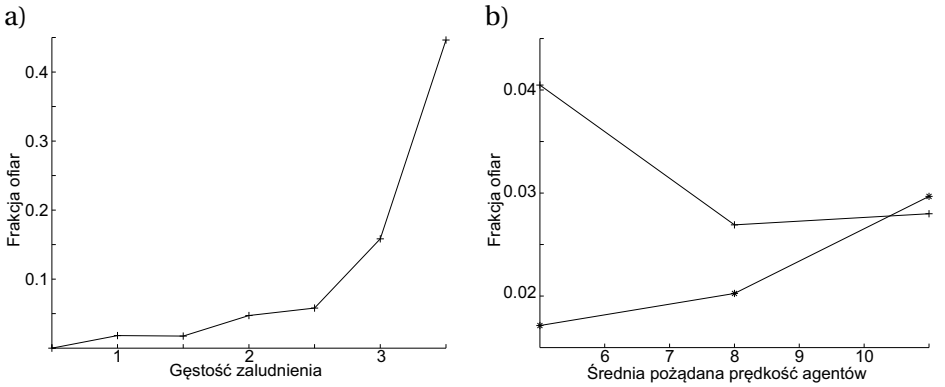
(e) Rozkład prędkości - 0% znajomość wyjść awaryjnych.



(f) Rozkład prędkości - 100% znajomość wyjść awaryjnych.

Rys. 3.8: Wpływ znajomości wyjść awaryjnych na przebieg ewakuacji (rozkład ścisku, liczby stratowań i średniej prędkości).

zmniejszenie ścisku przy drzwiach. Poszerzenie pojedynczych drzwi nie daje oczekiwanego efektu i zwiększa bezpieczeństwo jedynie w nieznacznym stopniu.



Rys. 3.9: Zależność frakcji ofiar (stosunek liczba ofiar do początkowej liczby agentów) od a) gęstości zaludnienia; b) średniej prędkości agentów (krzyżyki - 100% agentów podatnych na panikę, gwiazdki - 0% agentów podatnych na panikę).

Symulacja ewakuacji kompleksu budynków dostarczyła danych o kluczowym wpływie liczbie osób znajdujących się w budynku na liczbę ofiar. Okazało się, że zależność liczby ofiar od liczby osób w budynku jest wykładnicza. Ciekawe obserwacje dotyczą także zwiększania prędkości agentów w czasie ewakuacji. Dla normalnego chodu (5km/h) dużą liczbę ofiar powoduje dopiero wystąpienie paniki. Dla większych prędkości (11km/h - bieg truchtem) duża liczba ofiar pojawia się niezależnie od wystąpienia paniki.

Najistotniejszym wynikiem badań jest zidentyfikowanie kluczowych ciągów komunikacyjnych budynku i miejsc, w których występuje największy ścisk. Miejsca te są ściśle związane ze znajomością wyjść ewakuacyjnych. Symulacje pokazały, że ich pełna znajomość wcale nie zwiększa bezpieczeństwa ewakuacji, gdyż zbyt duża liczba agentów próbuje wtedy opuścić przez nie budynek, ignorując bardziej oddalone wyjścia główne.

Aplikacja stworzona w programie Netlogo pozwoliła skutecznie zaimplementować podejście agentowe z elementami algorytmów przepływowych i komórkowych. Część sterująca programu jest intuicyjna i pozwala na łatwe prowadzenie badań - ich zakres może być jeszcze większy niż opisane w niniejszej rozdziale. Jedyną wadą opisywanego rozwiązania jest długi czas symulacji dla dużej liczby agentów w budynku.

Literatura

[1] Gustave Le Bon. *Psychologia tłumu. Studium powszechnego umysłu.* Psychologia, 2011.

[2] G. Tarde. *Opinia i tłum, Tom IX.* Biblioteka Tygodnika Ilustrowanego, 1994.

[3] H. Blumer. *Interakcjonizm symboliczny. Perspektywa i metoda.* Zakład Wydawniczy Nomos, 2007.

[4] C. McPhail and R.T. Wohlstein. Individual and collective behaviors within gatherings, demonstrations, and riots. *Annual Review of Sociology*, 9:579–600, 1983.

[5] H. Benesch. *Atlas Psychologii, Tom II*. Prószyński i S-ka, 2005.

[6] E.L. Quarantelli. The sociology of panic. *International Encyclopedia of the Social and Behavioral Sciences*, 2001.

[7] N.J. Smelser. *Theory of Collective Behavior*. Free Press, 1963.

[8] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:223–242, 2000.

[9] W. Yu and A. Johansson. Modeling crowd turbulence by many-particle simulations. *Physical Review E*, 2007.

[10] Mehdi Moussaid, Dirk Helbing, and Guy Theraulaz. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences of the United States of America*, 108:6884–6888, apr 2011.

[11] Mehdi Moussaid and et al. Experimental study of the behavioural mechanisms underlying self-organization in human crowds. *Proceedings of the Royal Society B*, 276:2755–2762, 2009.

[12] M.E. Xiong, W.T. Cai, S.P Zhou, M. Low, F. Tian, and D. Chen. A case study of multi-resolution modeling for crowd simulation. Agent-Directed Simulation Symposium, San Diego, California, Marzec 2009.

[13] N. Pelechano and N.I. Badler. Modeling Crowd and Trained Leader Behavior during Building Evacuation. *IEEE Computer Graphics and Applications*, November 2006.

[14] G. Santos and B.E. Aguirre. A Critical Review of Emergency Evacuation Simulation Models, June 2004.

[15] D. Thalmann and S.R. Musse. *Crowd simulation*. Springer, 2007.

[16] T.M. Kisko, R.L. Francis, and C.R. Nobel. Evacnet4 user's guide. `http://www-old.ise.ufl.edu/kisko/files/evacnet/EVAC4UG.HTM`.

[17] Netlogo. `http://ccl.northwestern.edu/netlogo/`.

[18] Eligiusz Pieczyński. NetLogo – wygodne narzędzie do modelowania procesów ekologicznych. *Wiadomości ekologiczne*, LVI(3):95–138, 2010.

ROZDZIAŁ 4

REGUŁOWA WALIDACJA DANYCH W FORMACIE XML

B. Komarnicki, J. Kunert, B. Weselak

4.1. Wprowadzenie

Przez wieki gromadzenie wiedzy polegało na sporządzaniu i archiwizowaniu obszernych ksiąg. Obecnie, w dobie rozwoju technologii informacyjnych, rozległe księgozbiory zostały zastąpione portalami internetowymi (bazami danych), a same księgi – zasobami opublikowanymi na stronach tych portali (rekordami w bazach danych). Dzięki nowym rozwiązaniom sytuacja badaczy i zwykłych poszukiwaczy informacji polepszyła się diametralnie. Zniknęła bariera polegająca na konieczności fizycznego kontaktu z zasobem źródłowym. Dzięki nowym technologiom pojawiła się możliwość szybkiego i sprawnego przeszukiwania istniejących zasobów oraz publikowanie nowych, i to nawet bez konieczności opuszczania domu czy miejsca pracy.

W stosowanych obecnie rozwiązaniach dużą rolę odgrywają języki znaczników powstałe na bazie języka XML. Języki te pozwalają zapisywać dane w postaci strukturalnej w plikach tekstowych, nadając im określony kształt i charakter. Dane sformatowane zgodnie ze składnią języka XML nazywane są poprawnymi składniowo (ang. *well-formed*) dokumentami XML. Jednak poprawność dokumentu XML nie gwarantuje, że jego zawartość będzie zgodna z oczekiwaniami. Stworzono więc metody, dzięki którym można zdefiniować, a potem zweryfikować zawartość dokumentów XML. Polegają one na przeprowadzeniu automatycznej walidacji względem zadanego wzorca.

Do formułowania wzorców zawartości dokumentów XML (albo schematów, jak przyjęło się nazywać te wzorce) służą tzw. języki walidacji, z których najczęściej wykorzystywane są: definicja typu dokumentu (ang. *Document Type Definition*, DTD), schemat XML (ang. *XML Schema*), nazywany czasem schematem XSD od rozszerzenia stosowanego w nazwach plików przy zapisywaniu tego typu schematów, oraz Relax NG (ang. *REgular LAnguage for XML Next Generation*). Dokument, który pozytywnie przejdzie walidację względem zadanego schematu jest nazywany poprawnym strukturalnie lub walidowalnym (ang. *valid*) doku-

mentem XML. Dokumenty walidowalne względem jakiegoś schematu stanowią klasę zdefiniowaną tymże schematem. Stąd często stosuje się termin "instancja" w odniesieniu do walidowalnego dokumentu XML.

Formalnie rzecz biorąc DTD jest częścią specyfikacji języka XML. Natomiast XML Schema oraz Relax NG są aplikacjami XML (czyli językami zbudowanymi na bazie języka XML). Pomimo licznych korzyści wynikających ze stosowania tych języków, w praktycznych implementacjach dało się zauważyć pewne ich braki. Dlatego też opracowano schematron – kolejny język walidacji, bazujący na XML i wykorzystujący XPath.

Schematron umożliwia tworzenie asercji dotyczących obecności lub braku pewnych schematów w drzewie dokumentu XML (w informatyce asercja to pewien predykat, albo inaczej forma zdaniowa danego języka, który przyjmuje wartość prawdy lub fałszu). Pozwala również na definowanie ograniczeń oraz tworzenie wiadomości o przebiegu walidacji, na co nie pozwalają XML Schema, DTD ani Relax NG. Używając schematronu można, na przykład, definiować warunkowe wystąpienie pewnych elementów w strukturze dokumentów XML oraz określać komentarze, które zostaną przekazane użytkownikowi w przypadku wystąpienia jakichś błędów. Możliwość tworzenia komentarzy jest szczególnie przydatną cechą schematronu. Daje ona sprzężenie zwrotne do użytkownika, pozwalając mu nie tylko na dostrzeżenie błędów w walidowanych plikach XML, ale także na ich poprawę dzięki dostarczonym podpowiedziom.

W niniejszej pracy przybliżona zostanie idea schematronu. Jego możliwości zilustrowane będą przykładem aplikacji służącej do sterowania robotem. Język XML posłużył w niej do zapisu danych o sekwencjach ruchów robota, a schematron umożliwił zwalidowanie tak zapisanych sekwencji pod kątem ich fizycznej realizowalności (ze względu na fizyczne ograniczenia pewne sekwencje ruchów robota mogą okazać się niewykonalne).

4.2. Modelowanie danych

Zastosowanie języków walidacji można potraktować jako jeden ze sposobów modelowania danych. Definiując reguły i wzorce tworzy się pewien model, który nie dość, że określa strukturę danych, to daje się wykorzystać do automatycznej oceny zgodności z nim rzeczywistych danych.

Modelowanie danych jest rozległym tematem. Wiążą się z nim, między innymi, takie zagadnienia, jak: metody reprezentacji wiedzy, budowa schematu konceptualnego i aplikacyjnego, regułowa walidacja schematów aplikacyjnych i danych. Niniejszy podrozdział rozpocznie krótkie wprowadzenie do problemu budowy modelu danych i schematów aplikacyjnych. W dalszej jego części przybliżone zostaną pojęcia testowania, weryfikacji oraz walidacji.

4.2.1. Budowa modelu danych

Model danych jest uporządkowanym opisem wybranych obiektów świata rzeczywistego, które mogą być obiektami fizycznymi (jak kamień) i niefizycznymi (jak humor), relacjami pomiędzy obiektami, zdarzeniami itp. Model danych

można nazwać formalnym opisem sposobu budowy zbioru danych, obejmujący:

- strukturę danych, tj. reprezentację obiektów oraz ich związków,
- zbiór zasad określających sposób gromadzenia, przetwarzania, modyfikacji oraz udostępniania danych, a także warunki zapewniające zachowanie integralności danych w zbiorze.

Pojęcie modelu danych może być postrzegane w perspektywie tego, co model zawiera i tego, jakim celom ma on służyć. Dla przykładu programista postrzega model danych jako abstrakcyjny sposób reprezentacji obiektów świata rzeczywistego i zachodzących między nimi relacji, umożliwiający ich implementację w wybranym języku programowania. Patrzący z drugiej strony użytkownik dostrzega model danych w świetle możliwości jego wykorzystania w danej dziedzinie i w określonym celu, bez dogłębnej analizy szczegółów. Stąd budując model danych należy uwzględniać różne jego aspekty na różnych poziomach abstrakcji, w tym aspekty związane z:

- **Postrzeganą rzeczywistością** - ustalenie zakresu danych i sposobu ich organizacji w kontekście realizacji celów stawianych w projekcie. Na tym etapie projektowanie odbywa się w perspektywie użytkownika i pod kątem przeznaczenia tworzonego modelu (np. dla użytkownika interesujące mogą być różnice między produktami w magazynie sklepowym).
- **Budową modelu konceptualnego** - odniesienie do wybranych typów obiektów i zachowań ogólnego modelu, istotnych z punktu widzenia postawionych celów. Na tym etapie tworzony zostaje formalny *schemat pojęciowy* modelu, który jest niezależny od narzędzi informatycznych. Model konceptualny pozwala na określenie semantycznych związków pomiędzy danymi zapisanymi a światem rzeczywistym.
- **Budową modelu logicznego** - uporządkowanie zdefiniowanych obiektów i zachowań w strukturze danych, pozwalające na realizację postawionego zadania. Na tym etapie powinien zostać zapewniony właściwy typ obiektów, który zostanie później obsłużony przez wybrany język programowania. Pamiętać przy tym należy, że model powinien być niezależny od języków programowania, w których odbywa się implementacja obsługi danych. Na tym poziomie określane są związki topologiczne, a także ustalane są relacje.
- **Budową modelu fizycznego** - określenie przede wszystkim formy i sposobu przedstawiania poszczególnych obiektów w systemie gromadzącym dane (w postaci baz danych, tablic, plików itp.). Rozpatrywany jest tutaj także problem zasad prezentacji danych.

Ogromne znaczenie modelowania danych w informatyce zostało dostrzeżone przez konsorcjum OMG (ang. *Object Management Group*). Konsorcjum to opracowało koncepcję MDA (ang. *Model Driven Architecture*), mającą pomóc w rozwiązywaniu rozlicznych problemów związanych z integracją systemów informatycznych wykorzystujących różne technologie. Koncepcja ta obejmuje zbiór metod porządkujących proces tworzenia systemów komputerowych z naciskiem

na budowę modeli i ich transformacje na różnych poziomach abstrakcji, od meta-metamodelu (sposób (standardy) modelowania oraz zarządzania modelami określa MOF (ang. *Meta-Object Facility*)), poprzez metamodel (w języku UML), model rzeczywistych obiektów, model implementacyjny (programy), aż po dane. Z perspektywy danych stosowanie zaleceń MDA oznacza: tworzenie niezależnych od implementacji schematów aplikacyjnych na bazie modelu pojęciowego, które zostają odwzorowane na różne specyfikacje, aby ostatecznie zostać zaimplementowane i wdrożone na różnych platformach sprzętowo-programowych (wybrane definicje z tego obszaru przedstawiono w tabeli 4.1). Język XML pojawia się w tej sekwencji działań podczas odwzorowania schematów aplikacyjnych na różne specyfikacje.

Tab. 4.1: Wykaz definicji związanych z modelowaniem danych.

Termin	Opis
Model pojęciowy (ang. *conceptual model*)	Wiąże pojęcia z pewnej przestrzeni rozważań (przedmiotu zainteresowań).
Schemat pojęciowy (ang. *conceptual schema*)	Formalny opis modelu pojęciowego w określonym języku schematu pojęciowego.
Schemat aplikacyjny (ang. *application schema*)	Powstaje ze schematu pojęciowego w zawężeniu do danych z pewnego obszaru zastosowań, wykorzystywanych przez jedną lub więcej aplikacji.
Język schematu pojęciowego (ang. *conceptual schema language*)	Język formalny, dostarczający semantycznych i syntaktycznych elementów do opisu modelu pojęciowego, mogący przyjąć formę leksykalny lub graficzną.

4.2.2. Testowanie, weryfikacja i walidacja

W dziedzinie inżynierii oprogramowanie dużą wagę przykłada się do poprawnego wykonywania następujących czynności [1]: testowania (sprawdzanie oprogramowania pod kątem występowania błędów oraz spełnienia oczekiwań klienta co do jego funkcjonalności), weryfikacji (ocena zgodności wykonanego programu ze specyfikacją), walidacji (badanie, czy system działa w sposób satysfakcjonujący w rzeczywistych warunkach, dla których został zaprojektowany).

Weryfikacja i walidacja, V&V (ang. *verification and validation*) pozwalają wykrywać błędy i chronić się przed nimi w stosunkowo wczesnej fazie tworzenia oprogramowania. Proces weryfikacji, oprócz sprawdzenia zgodności oprogramowania ze specyfikacją, pozwala dodatkowo wykryć ewentualne błędy i nieścisłości w samej specyfikacji. W procesie walidacji rozważa się, czy stworzony system (model rozwiązania) dobrze funkcjonuje w rzeczywistych warunkach rozwiązuje postawione przed nim zadania.

Istnieją metody umożliwiające analizę repozytoriów kodu i rejestrów błędów, dzięki którym można zdobyć wiedzę o newralgicznych miejscach oprogramowa-

nia, gdzie błędy pojawiają się najczęściej. Metody te ułatwiają znalezienie błędów, usprawniając cały proces V&V. Ian Sommerville w [2] wyróżnił 6 grup błędów jakie można wykryć korzystając z V&V:

- błędy funkcjonalne – zła lub brakująca funkcja,
- błędy systemowe – błędne interfejsy, nieprawidłowe zarządzanie zasobami,
- błędy przetwarzania – niewłaściwe przetwarzanie danych,
- błędy danych – błędna specyfikacja, projekt, rozmieszczenie,
- błędy kodowania – niewłaściwe użycie języka oprogramowania,
- błędy dokumentacyjne – niepełna lub błędna treść dokumentu.

W trakcie planowania testów oprogramowania należy pamiętać o tym, że weryfikacja i walidacja nie zapewniają uzyskania zerowego wskaźnika wystąpienia błędów. Umożliwiają one jedynie uzyskanie pewnego poziomu pewności prawidłowego działania systemu. W praktyce stosuje się następujące tryby przeprowadzania walidacji:

- walidacja prospektywna - jest wykonywana zanim powstaną i zostaną użyte nowe elementy systemu; zapobiega użyciu w programie niepoprawnych danych, funkcji czy metod; zapewnia dodatkowo wysoki standard bezpieczeństwa aplikacji;
- walidacja retrospektywna - jest przeprowadzana już w trakcie działania systemu; jest wykonywana w opozycji do specyfikacji programu czy przewidywanych wyników; bazuje jedynie na danych historycznych; wyłączne jej stosowanie bardzo często jest kosztowne z punktu widzenia finansowego oraz nakładów pracy poniesionych w trakcie usuwania usterek;
- walidacja pełna - połączenie dwóch powyższych sposobów;
- walidacja częściowa - testowaniu są poddawane wybrane, najczęściej kluczowe elementy oprogramowania.

Języki walidacji, pomimo zbieżnej nazwy z procesem walidacji, mają nieco inny kontekst zastosowania. Zasadniczo wykorzystuje się je na etapie implementacji, gdzie pomagają w automatycznym generowaniu kodu oraz kontroli poprawności przetwarzanych danych już w trakcie działania programu.

4.3. Schematron

Schematron wymyślił Rick Jelliffe w Academia Sinica Computing Centre, Taiwan. Od opublikowania pierwszej wersji specyfikacji schematronu w 1999 przeszedł on kilka modyfikacji. W 2006 stał się częścią standardu ISO, definiującego języki definiowania schematów dokumentów: „ISO/IEC 19757 - Document Schema Definition Languages (DSDL) - Part 3: Rule-based validation - Schematron".

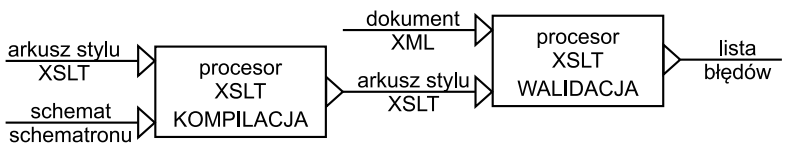
Schematron, choć bazuje na XML, różni się znacząco od innych języków schematów XML. Jest on oparty na własnościach (asercjach), a nie na gramatyce. Pozwala na kontekstową walidację, polegającą na definiowanie warunków i reguł (asercji) odnoszące się do elementów w drzewie XML wybieranych przez określe-

nie ścieżki za pomocą XPath. Używając mechanizmu rozszerzeń XPath projektanci mogą wprowadzać dodatkowe funkcje wspierające sprawdzanie warunków asercji. Dodatkowo, umożliwia generowanie komunikatów diagnostycznych, gdy jakiś warunek lub reguła nie znajdzie swojego odzwierciedlenia w walidowanym dokumencie XML. Schemat schematronu zapisywany jest w plikach z rozszerzeniem `.sch`.

W schematronie nie można opisać modelu danych dokumentu (struktury, typów danych, wartości domyślne). Ponadto wszystko, co nie zostanie jawnie zabronione w schemacie, będzie uznane za poprawne. Problem ten można rozwiązać, stosując schematron razem z innym językiem schematów XML (DTD, XML Schema). W takich przypadkach reguły kontekstowe definiuje się w języku schematronu, zaś strukturę dokumentu – w języku schematów, np. XML Schema. Schemat schematronu zanurza się w schemacie XML Schema, korzystając z elementu `annotation`, i jego podelementu `appinfo` (gdzie można umieszczać dowolne dane przeznaczone dla innych aplikacji niż procesor XML Schema). Elementy `annotation` nie są normalnie przeglądane przez procesor XML Schema. Aby je wydobyć (z pliku z rozszerzeniem `.xsd` i zapisać w pliku z rozszerzeniem `.sch`), należy posłużyć się dodatkowym przekształceniem XSLT (dostępnym na przykład pod adresem `http://www.schematron.com/resource/XSD2SCH-2010-03-11.zip`).

Schematron pozwala użytkownikom ograniczyć się do projektowania reguł biznesowych, w którym można sięgać do dowolnego elementu dokumentu XML będącego instancją schematu. Zwalnia ich od przeprowadzania samej walidacji, która odbywa się już w sposób automatyczny, z wykorzystaniem silnika walidacji, którym zazwyczaj jest jakiś wybrany procesor XSLT.

Wzorcowa implementacja schematronu działa w sposób następujący (rysunek 4.1). Na początku powstaje schemat schematronu. Jest on zapisywany w pliku z rozszerzeniem `.sch`. Następnie dla tego schematu jest generowane przekształcenie XSLT, zapisywane zazwyczaj w pliku z rozszerzeniem `.xsl`. Stosując to przekształcenie do wybranego dokumentu XML przeprowadza się ostatecznie jego walidację. Jeśli wynikiem zastosowania przekształceń jest dokument XML z pustym element głównym, oznacza to pozytywną walidację względem schematu. Walidacja negatywna kończy się wygenerowaniem pliku wynikowego zawierającego listę błędów.



Rys. 4.1: Wzorcowa implementacja schematronu.

Rzeczywiste implementacje schematronu mogą nieco odbiegać od implementacji wzorcowej. W sekwencji przekształceń mogą bowiem pojawić się dodatkowe kroki, związane np. z wydobyciem schematu zanurzonego w XML Schema

czy też z uwzględnieniem rozszerzeń. Dobry ich przegląd autorstwa Ricka Jelliffe można znaleźć pod adresem http://broadcast.oreilly.com/2009/02/running-schematron-the-evoluti.html.

W schematronie ISO istnieje tylko kilka podstawowych elementów (porównaj z listingiem 4.1):

- schema - element główny, może zawierać title, ns oraz wiele pattern,
- title - element opcjonalny, pozwalajacy nadać tytuł schematowi,
- ns - definuje zero lub więcej przestrzeni nazw i prefiksów wykorzystywanych w XPaths,
- pattern - wzorzeć, grupuje elementy rule z identyfikatorem (wykorzystywanym przez phase). Każdy węzeł testowy będzie użyty jedynie jako węzeł kontekstowy dla jednej tylko reguły wewnątrz wzorca.
- rule - reguła, grupuje wiele elementów asercji (elementy assert i report) oraz określa zbiór węzłów kontekstowych, dla których mają one być sprawdzane (w atrybucie context)
- assert - element asercji, pozwalają sprawdzić pewne warunki dla wybranych elementów, jak np. występowanie oraz wartości elementów i atrybutów. Atrybut test elementu assert jest ścieżką XPath (definicją warunku, który musi być spełniony), zaś wartość tego elementu to tekstowy opis sprawdzanego warunku.
- report - elementy asercji, pozwalają zaraportować pewne fakty. Atrybut test elementu report jest ścieżką XPath (definicją warunku, którego spełnienie oznacza błąd), zaś wartość tego elementu jest raport w postaci tekstowej.

Mogą wystapić też inne, do tworzenia bardziej złożonych schematów lub do tworzenia ładnego interfejsu użytkownika dla walidatorów.

- phase – to kolekcje elementów pattern uruchamianych razem (w jednej fazie).
- include - wywołania zewnętrznych modułów schematronu.
- let - deklaracje zmiennych.
- diagnostics - rozszerzona informacja diagnostycznych dla reguł, których dotyczy.

Listing 4.1: Główne elementy schematu schematronu zapisane w pliku.

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
 <title>
 <ns prefix="???" uri="???" />
    <pattern>,
       <rule context="???">
        <assert test="???">
        <report test="???">
```

Wszystkie elementy gramatyki schematronu wg standardu ISO/IEC 19757 są zdefiniowane w przestrzeni nazw o następującym URI: http://purl.oclc.org/dsdl/schematron. Chociaż z przestrzenią tą często jest kojarzony przedrostek sch:, to nie jest on przedrostkiem zarezerwowanym ani wymaganym.

Projektowanie schematów schematronu z założenia powinno być w miarę proste. Dla przykładu, przy wymaganiach co do zawartości dokumentu XML zdefiniowaych jak niżej:

- element kontekstowy `Osoba` musi posiadać atrybut `Tytuł`,
- element kontekstowy powinien posiadać dwa potomne elementy, `Imię` oraz `Płeć`,
- element potomny `Imię` powinien pojawiać się przed elementem *Płeć*,
- jeśli atrybut `Tytuł` ma wartość `Pan` to element potomny `Płeć` musi mieć wartość `M`,

schemat schematronu zgodnego z normą ISO/IEC 19757 może mieć postać jak na listingu 4.2.

Listing 4.2: Przykład schematu schematronu.

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
 <title>A Schematron Mini-Schema for Schematron</title>
 <ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron" />
  <pattern name="Sprawdzenie struktury">
    <rule context="Osoba">
      <assert test="@Tytuł">
        Element "Osoba" powinie mieć atrybut "Tytuł"
      </assert>
      <assert test="count(*)=2 and
                        count(Imię)=1 and count(Płeć)=1">
        Element "Person" powinien mieć elementy
        potomne "Imię" oraz "Płeć".
            </assert>
      <assert test="*[1]=Imię">
        Element "Imię" powinien wystąpić przed elementem "Płeć".
            </assert>
    </rule>
  </pattern>
  <pattern name="Sprawdzanie współwystąpienia ograniczeń">
    <rule context="Osoba">
      <assert test="(@Tytuł='Mr' and Płeć='M')
                        or @Tytuł!='Pan'">
        Jeśli tytuł ma wartość "Pan" to płeć dla osoby
                musi być "M".
            </assert>
    </rule>
  </pattern>
</schema>
```

Postępując zgodnie z wzorcową implementacją (pokazaną na rysunku 4.1), za pomocą tego schematu można walidować dokumenty XML. Wystarczy wydać dwie komendy w linii poleceń, tak, jak to przedstawiono na listingu 4.3.

Listing 4.3: Sekwencja komend wydanych w linii poleceń w celu zwalidowania dokumentu `mojPlik.xml` względem schematu `mojSchemat.sch` (ukośnik \ oznacza przeniesienie treści polecenia do następnej linii).

```
> java -cp .;saxon9he.jar;xercesImpl.jar net.sf.saxon.Transform \
-x:org.apache.xerces.parsers.SAXParser \
-o:tmp.xsl mojSchemat.sch iso_schematron_text.xsl

> java -cp .;saxon9he.jar;xercesImpl.jar net.sf.saxon.Transform \
-x:org.apache.xerces.parsers.SAXParser \
-o:raport.xml mojPlik.xml tmp.xsl
```

Pierwsza komenda służy do utworzenia pliku szablonu `tmp.xsl` ze schematu `mojScheat.sch` przy wykorzystaniu walidatora Schematron Text, zapisanego w pliku `iso_schematron_text.xsl`. Wygenerowany plik szablonu pozwoli dokonać automatycznej oceny zawartości zadanego dokumentu XML `mojPlik.xml` podczas wykonywania drugiej komendy. Ponieważ `iso_schematron_text.xsl` zależy od `iso_schematron_skeleton.xsl` do poprawnego działania walidatora konieczne jest umieszczenie w tym samym co on katalogu plików z walidatorami ISO do pobrania ze strony `http://www.schematron.com/implementation.html`.

Druga komenda generuje raport `raport.xml` z wynikami walidacji dokumentu `mojPlik.xml`, stosując przekształcenia zapisane w wygenerowanym wcześniej pliku `tmp.xsl`. Wynikami walidacji będą tekstowe opisy znalezionych błędów (negatywnych asercji lub pozytywnych raportów) zgodne z tym, co zadeklarowano w schemacie schematronu. Dla zawartości walidowanego pliku `mojPlik.xml` jak na listingu 4.4 wynik walidacji będzie miał postać jak na listingu 4.5.

Obie opisane komendy wiążą się z wykonaniem operacji na plikach XML i XSLT za pomocą metod bibliotek klas javy: saxon oraz xerces. Użyty walidator `iso_schematron_text.xsl` razem z innymi przykładowymi walidatorami można znaleźć na stronie `http://www.schematron.com/validators.html`.

Listing 4.4: Zawartość walidowanego dokumentu XML (plik `mojPlik.xml`).

```
<?xml version="1.0" encoding="UTF-8" ?>
<Osoba Tytuł="Pan">
    <Imie>Jan</Imie>
</Osoba>
```

Listing 4.5: Wynik walidacji dokumentu (plik `raport.xml`).

```
Warning: at xsl:stylesheet on line 7 column 31 of tmp.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor

        Element "Person" powinien mieć elementy
        potomne "Imię" oraz "Płeć".
            : /Osoba

        Element "Imię" powinien wystąpić przed elementem "Płeć".
            : /Osoba
```

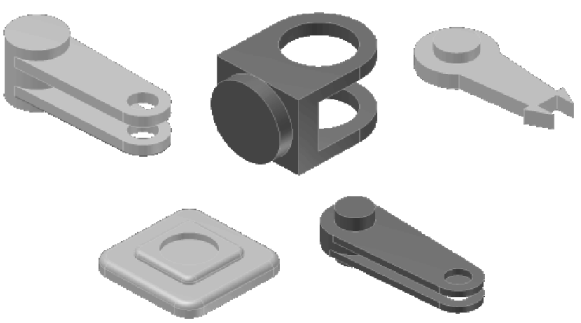
```
Jeśli tytuł ma wartość "Pan" to płeć dla osoby
        musi być "M".
    : /Osoba
```

4.4. Przykład zastosowania schematronu

W celu zademonstrowania możliwości walidacyjnych schematronu i schematu XSD powstała aplikacja do symulacji manipulatora typu 4R. W aplikacji tej zaimplementowano fizyczny model manipulatora, dla którego zakres możliwych ruchów uzależniono od jego konfiguracji. Oznacza to, że wartości kątów granicznych w przegubach manipulatora zależą od tego, w jakiej pozycji aktualnie on się znajduje. Sterowanie manipulatorem musi odbywać się z uwzględnieniem tych zmieniających się ograniczeń. Aby spełnić to wymaganie konfiguracja manipulatora jest zapisywana w plikach XML, które są poddawane walidacji pod kątem strukturalnym za pomocą schematu XSD, oraz pod kątem spełniania narzuconych warunków z wykorzystaniem schematronu. Poniżej opisano ogólny proces tworzenia aplikacji, jej strukturę oraz zasadę działania. Podano również przykłady przetwarzanych plików w celu lepszego zobrazowania omawianego zagadnienia.

4.4.1. Model fizyczny

W pierwszym etapie procesu tworzenia aplikacji wykonano trójwymiarowy model manipulatora. Wykorzystano w tym celu program Autodesk Inventor (bezpłatną wersję testową, którą można pobrać ze strony `www.autodesk.pl/inventor`). Zaprojektowany manipulator posiadał cztery przeguby obrotowe, przy czym jeden przegub miał pionową oś obrotu, a trzy pozostałe przeguby - osie poziome. Każda z części manipulatora wykonywana była osobno, poprzez łączenie prostych brył geometrycznych (przedstawiono je na rysunku 4.2). Model końcowy, którego wizualizację można zobaczyć na rysunku 4.3, powstał poprzez scalenie wszystkich części połączeniami obrotowymi.



Rys. 4.2: Części manipulatora.

4.4.2. Implementacja

W drugim etapie zaimplementowano aplikację umożliwiającej sterowanie manipulatorem, pobieranie parametrów jego konfiguracji, zapisywanie ich do pliku, a także walidację tych plików z wykorzystaniem przygotowanych schematów XSD oraz Schematronu.

W pierwszej kolejności powstało okno symulacyjne. Przy jego tworzeniu wykorzystano zestaw narzędzi programowych XNA Framework [3], pozwalający na łatwe dołączanie i animację grafiki 3D w aplikacji tworzonej w języku C#.

Aby zapisać, a następnie jednoznacznie odtworzyć konfigurację manipulatora 4R, wystarczy znać bieżące kąty obrotu w osiach jego kolejnych przegubów. Bazując na tym założeniu przygotowano schemat XSD definiujący oczekiwaną zawartość pliku XML z zapisaną konfiguracją manipulatora. Przedstawiono go na listingu 4.6.

Listing 4.6: Schemat XSD dla plików XML z zapisaną konfiguracją manipulatora.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Manipulator" xmlns=""
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:annotation>
   <xs:appinfo source="urn:schemas-microsoft-com:xml-msdatasource">
    <DataSource DefaultConnectionIndex="0"
                FunctionsComponentName="QueriesTableAdapter"
                Modifier="AutoLayout, AnsiClass, Class, Public"
                SchemaSerializationMode="IncludeSchema"
                xmlns="urn:schemas-microsoft-com:xml-msdatasource">
      <Connections />
      <Tables />
      <Sources />
    </DataSource>
   </xs:appinfo>
  </xs:annotation>
  <xs:element name="Manipulator" msdata:IsDataSet="true"
                                 msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="MoveSequence">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="BottomLink"
                          minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="XRot" type="xs:float" />
                </xs:complexType>
              </xs:element>
              <xs:element name="MidLink"
                          minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="YRot" type="xs:float" />
```

```
            </xs:complexType>
          </xs:element>
          <xs:element name="TopLink"
                      minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="YRot" type="xs:float" />
              <xs:attribute name="XCoord" type="xs:float" />
              <xs:attribute name="YCoord" type="xs:float" />
              <xs:attribute name="ZCoord" type="xs:float" />
            </xs:complexType>
          </xs:element>
          <xs:element name="Effector"
                      minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="YRot" type="xs:float" />
              <xs:attribute name="XCoord" type="xs:float" />
              <xs:attribute name="YCoord" type="xs:float" />
              <xs:attribute name="ZCoord" type="xs:float" />
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="MoveNo"
                      type="xs:int" use="required" />
      </xs:complexType>
    </xs:element>
  </xs:choice>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Dokumenty XML tworzone na jego podstawie posiadały liście o nazwach poszczególnych ogniw oraz atrybuty odpowiadający nazwie osi, wokół której następował obrót. Dodatkowo, w przypadku efektora i najwyższego ogniwa, dodane zostały współrzędne położenia czubka ogniwa w celu późniejszej walidacji w poszukiwaniu kolizji. Zawartość przykładowego pliku XML z konfiguracją manipulatora przedstawiono na listingu 4.7.

Listing 4.7: Zawartość przykładowego pliku XML z konfiguracją manipulatora.

```
<?xml version="1.0" standalone="yes"?>
<Manipulator>
  <MoveSequence MoveNo="1">
    <BottomLink XRot="0" />
    <MidLink YRot="-0.701" />
    <TopLink YRot="-1.170" XCoord="619.916"
             YCoord="-316.989" ZCoord="90.000" />
    <Effector YRot="-0.694" XCoord="1336.151"
              YCoord="-539.4" ZCoord="180.000" />
  </MoveSequence>
  <MoveSequence MoveNo="2">
    <BottomLink XRot="0" />
    <MidLink YRot="-1.74" />
```

```
    <TopLink YRot="-0.686" XCoord="938.466"
                YCoord="-1252.189" ZCoord="90.000" />
    <Effector YRot="2.237" XCoord="1438.338"
              YCoord="-1777.690" ZCoord="180.000" />
  </MoveSequence>
  <MoveSequence MoveNo="3">
    <BottomLink XRot="3.050" />
    <MidLink YRot="-0.018" />
    <TopLink YRot="1.027" XCoord="-957.415"
             YCoord="-1211.661" ZCoord="332.756" />
    <Effector YRot="5.735" XCoord="607.042"
              YCoord="309.356" ZCoord="386.072" />
  </MoveSequence>
</Manipulator>
```

Następnie, bazując na specyfikacji schematronu o numerze wersji 1.6, zaprojektowano przykładowy schemat schematronu (pokazano go na listingu 4.8). Wybrano właśnie tę wersję specyfikacji ze względu równowagę pomiędzy funkcjonalnością, a prostotą implementacji jaką ona zapewniała. Reguły (asercje) zamieszczane w schematronie podczas walidacji pliku konfiguracji mają pozwolić na rozpoznanie możliwości kolizji pomiędzy poszczególnymi przegubami manipulatora i podłożem. W przykładowym schematronie zdefiniowano tylko dwa rodzaje ograniczeń:

- ograniczenie kąta obrotu poszczególnych przegubów,
- ograniczenie przesunięcia przegubu.

Jednak nic nie stoi na przeszkodzie, aby dołączyć do aplikacji pliki schematów schematronu z bardziej wyszukanymi regułami (wynikającymi np. z obecności przeszkód w scenie). Przyjęto, że przykładowy schemat będzie schematem ładowanym domyślnie podczas startu aplikacji. Inne schematy mogą być ładowane przez użytkownika już w trakcie działania aplikacji (pod warunkiem, że zostaną opracowane).

Listing 4.8: Struktura domyślnego schematronu.

```
<schema xmlns="http://www.ascc.net/xml/schematron">
     <pattern name="Link Constraints">
          <rule context="MiddleLink">
               <assert test="YRot &gt; 0.74">
               BottomLink Y rotation out of bounds.
               </assert>
               <assert test="YRot &lt; -1.74">
               BottomLink Y rotation out of bounds.
               </assert>
          </rule>
          <rule context="TopLink">
               <assert test="YRot &gt; 2.32">
               TopLink Y rotation out of bounds.
               </assert>
               <assert test="YRot &lt; -2.32">
               TopLink Y rotation out of bounds.
```

```
                </assert>
                <assert test="YCoord &lt; -1150">
                TopLink Y transition out of bounds
                - lowest configuration possible.
                </assert>
            </rule>
        </pattern>
</schema>
```

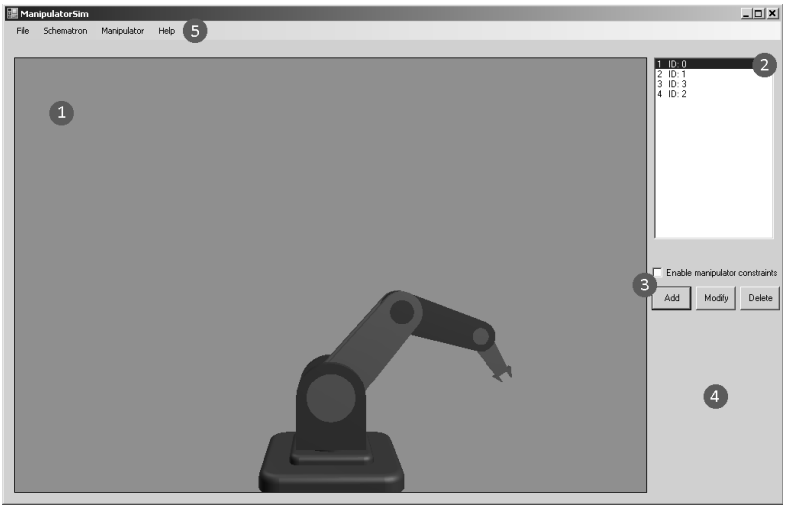
Graficzny interfejs gotowej aplikacji przedstawiono na rysunku 4.3. Jego główne komponenty to:

1. Okno symulatora - w tym oknie ujęta jest przestrzeń robocza manipulatora wraz z jego modelem.
2. Lista konfiguracji robota - prezentuje ilość zapisanych konfiguracji. Wybierając jedną z pozycji lewym klawiszem myszy można ją podejrzeć w oknie symulatora.
3. Przyciski obsługi kolejki konfiguracji:

   Add - dodaje obecną konfigurację, widoczną w oknie symulatora, do kolejki.
   Modify - modyfikuje wybraną pozycję w kolejce.
   Delete - usuwa wybraną pozycję z kolejki.
   Enable manipulator constrains - wspomaga tworzenie prawidłowej kolejki ruchów przez nałożenie na model ograniczeń obrotu na poszczególne stopnie swobody.
4. Obszar na wynik walidacji schematronem, a także instrukcja obsługi modelu robota
5. Pasek menu, z funkcjami kontroli działania programu:

   File - z opcjami: **New** (czyści kolejkę konfiguracji i przygotowuje świeżą instancję pliku XML), **Open...** (pozwala na wczytanie wcześniej zapisanej kolejki konfiguracji z pliku XML. Wczytywany plik jest poddawany walidacji schematem XSD.), **Save**, **Save As...** (pozwalają zapisać kolejkę do pliku w strukturze zgodnej ze schematem XSD).
   Schematron - z opcjami: **Load Schematron** (pozwala wczytać schematron z dysku), **Validate** (poddaje istniejącą kolejkę ruchów walidacji schematronem.).
   Manipulator - pozwala na konfigurację modelu manipulatora poprzez zmianę jego rozmiaru, czy pozycji w oknie symulatora.
   Help - wyświetla wskazówki dotyczące porzuszania manipulatorem.

Obsługa programu jest stosunkowo prosta. Do poruszania robotem służą cyfry na klawiaturze i myszka. Każdemu kolejnemu stopniowi swobody manipulatora przypisana jest kolejna cyfra. Trzymając wciśnięty przycisk należy poruszyć myszką, aby zmienić konfigurację robota. Aby poddać kolejkę konfiguracji manipulatora walidacji schematronem, należy wybrać z menu głównego programu: Schematron/Validate. Wynik walidacji zostanie wyświetlony na polu oznaczonym cyfrą 4 na rysunku 4.3. Walidacja schematem XSD następuje automatycznie podczas próby wczytania nowego pliku XML.



Rys. 4.3: Główne okno programu.

4.4.3. Podsumowanie

W przedstawionym rozwiązaniu schematron posłużył do zdefiniowania reguł pozwalających na sprawdzanie kolizyjności konfiguracji manipulatora zapisanych w plikach XML. Choć pokazany przykład miał akademicki charakter (trudno sobie wyobrazić, aby sterowanie manipulatorem w czasie rzeczywistym odbywało się poprzez zapisywanie jego konfiguracji w plikach XML, walidowanie względem schematu XSD i schematronu i następnie przesyłanie odpowiednich komend do manipulatora), to jednak doskonale zobrazował możliwości, jakie schematron otworzył w dziedzinie budowy modeli i walidacji danych. Jak pokazano, schematron może być z powodzeniem użyty do tworzenia reguł na zewnątrz wykorzystującej je aplikacji, lub, inaczej mówiąc, działanie skompilowanej aplikacji daje się sparametryzować dostarczanymi schematami. Niewątpliwą zaletą takiego podejścia jest jego elastyczność: pliki schematów mogą być generowane dynamicznie, stosownie do wymagań i potrzeb.

Literatura

[1] J. Górski. *Inżynieria oprogramowania w projekcie informatycznym.* Wydawnictwo Mikom, Warszawa, 2000.

[2] I. Sommerville. *Inżynieria oprogramowania.* Wydawnictwa Naukowo Techniczne, Warszawa, czer. 2003.

[3] D. Pośliński. Co to jest Microsoft XNA? `http://msdn.microsoft.com/pl-pl/library/co-to-jest-microsoft-xna.aspx`.

ROZDZIAŁ 5

WALIDACJA MODELI KONCEPTUALNYCH ZAPISANYCH W UML

M. Kot, M. Ogorzelec, M. Paluch

5.1. Wstęp

Rosnąca liczba coraz bardziej skomplikowanych systemów przetwarzania danych niesie za sobą potrzebę definiowania ich za pomocą elastycznych, a zarazem czytelnych reprezentacji. Budowanie złożonego systemu krok po kroku, bez szeregu założeń i możliwości szerszego spojrzenia na rozpatrywany problem często prowadzi do błędów, co przekłada się na znaczące wydłużenie procesu wdrażania i implementacji. Jednym z najczęściej stosowanych rozwiązań jest wygenerowanie modelu systemu już w fazie projektowania, co przy wykorzystaniu zaawansowanych narzędzi walidacji modelu może pomóc w diagnozowaniu trudno wykrywalnych błędy już w pierwszych cyklach życia systemu.

Do zdefiniowanego modelu systemu możliwe jest zastosowanie technik walidacyjnych, które służą do weryfikacji jego poprawności składniowej i zgodności ze specyfikacją. Jednym z popularniejszych narzędzi jest *SPIN model checker* [1], który w sposób automatyczny sprawdza poprawność modelu oprogramowania. Większość tego typu rozwiązań bazuje jednakże na konkretnych instancjach problemu, czyli warstwie M1 architektury metadanych (opisanej w podrozdziale 5.2.1).

Walidacja modeli z warstwy M2, polegająca na analizie stopnia zgodności modelu z zaproponowaną abstrakcją jest operacją również niezwykle istotna, aczkolwiek w o wiele mniejszym stopniu zbadana – głównie ze względu na duży poziom skomplikowania.

W dalszych podrozdziałach przedstawiono analizę problemu weryfikacji modelu już na poziomie konceptualnym. Zaczęto ją od wprowadzenia kilku istotnych pojęć. Opis zasad stosowania języka schematu konceptualnego przygotowano w oparciu o standard ISO/PDTR 19103 [2]. Opis sposobu tworzenia profili, ich struktury oraz metodologii badania zgodności przygotowano na bazie zaleceń z normy ISO 19106 [3]. Informacje z obszaru definiowania oraz testowania zgodności opracowano na podstawie normy ISO 19105 [4].

5.2. Standaryzacja modelowania

Problemem tworzeniem systemów w oparciu o model od dawna zajmuje się organizacja OMG (ang. *Object Management Group*). W celu ujednolicenia wszystkich specyfikacji tworzenia modeli używanych w ramach OMG, pod jej auspicjami powstała specyfikacja MOF (ang. *Meta-Object Facility*) [5].

5.2.1. Meta-Object Facility

MOF opisuje architekturę wykorzystywaną ściśle do meta-modelowania i daje możliwość budowania własnych języków modelowania dla wybranej kategorii zastosowań bądź też definiowania rozszerzeń istniejących języków modelowania. Dzięki zgodności nowych języków jak i rozszerzeń ze specyfikacją MOF, można je automatycznie dołączać do systemów współpracujących z repozytorium modeli MOF. Specyfikacja MOF stanowi najwyższą warstwę w hierarchii metamodeli OMG. Hierarchia ta posiada 4 warstwy:

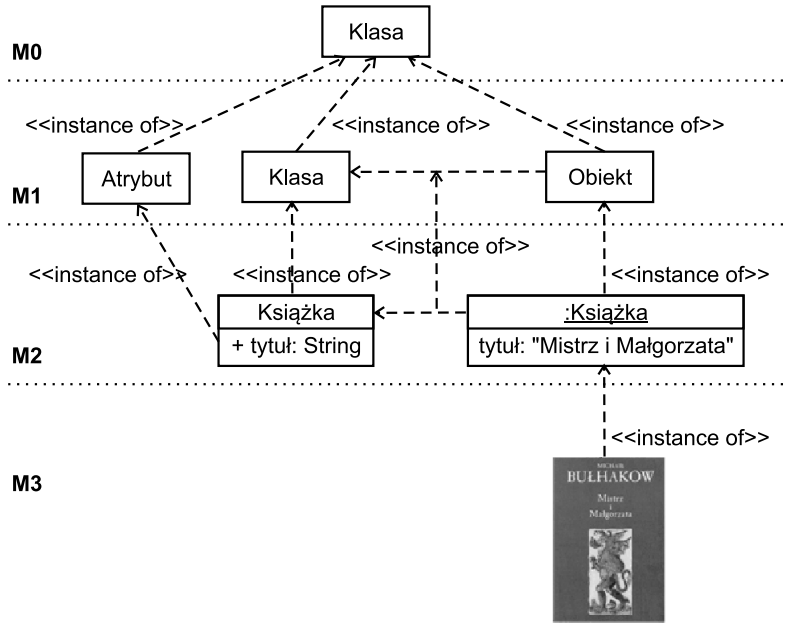
- M3 – warstwa meta-metamodelu zawierająca specyfikacja języka modelowania, np. Meta-Object Facility (MOF),
- M2 – warstwa metamodelu zawierająca model metadanych (specyfikacja języka), np. specyfikacja SQL, specyfikacja UML, specyfikacja XML,
- M1 – warstwa modelu zawierająca metadane, czyli opis formatu i znaczenia danych, np. schemat bazy danych, model klas, definicja dokumentu XML,
- M0 – rzeczywiste dane, które chcemy opisać, np. dane w bazie danych, instancje obiektów, dokument XML.

Na rysunku 5.1 przedstawiono przykładową instancję takiej hierarchii.

Na poziomie warstwy M2, zawierającej model metadanych, zostało wyspecyfikowanych kilka języków formalnych modelowania dziedziny różnego rodzaju systemów (w szczególności informatycznych). Wszystkie te języki, bazujące na warstwie M3 i zawierające Meta-Object Facility, mogą być serializowane w języku XMI (ang. *XML Metadata Interchange*). Najpopularniejsze z nich to: ORM (ang. *Object Role Modeling*), ER (ang. *Entity-relationship*), EER (ang. *Enhanced Entity-relationship*), UML (ang. *Unified Modeling Language*). Ich porównanie można znaleźć w [6].

5.2.2. Unified Modeling Language jako język schematu konceptualnego

Język UML [7] jest najczęściej stosowanym językiem reprezentacji modeli systemów informatycznych, ale i nie tylko - jest stosowany także do modelowania procesów biznesowych, inżynierii systemów i reprezentowania struktur organizacyjnych itp. UML posiada reprezentację graficzną – jego elementom przypisane są symbole, które są łączone ze sobą na diagramach. Sam język UML nie pozwala definiować formalnych ograniczeń, które mogłyby posłużyć do weryfikacji modelu na poziomie konceptualnym. Istnieje jednak jego rozszerzenie w postaci języka OCL (ang. *Object Constraint Language*), opracowanego pierwotnie przez IBM, które to umożliwia.



Rys. 5.1: Przykładowa hierarchia metamodeli OMG zbudowanych dla książki.

Schemat konceptualny jest to formalny zapis modelu konceptualnego. Do opisu języka schematu konceptualnego używa się języka UML. Modele normatywne są budowane w oparciu o diagramy klas i pakietów. Pozostałe typy diagramów mogą być użyte opcjonalnie. Modele normatywne powinny zawierać kompletne definicje atrybutów, asocjacji, operacji, a także definicje typów danych. Do najważniejszych elementów języka UML stosowanych do budowy języka schematu konceptualnego należą: Klasy, Atrybuty i Relacje.

Klasa zawiera opis zbioru obiektów, które posiadają te same atrybuty, metody, relacje, operacje czy ograniczenia. Klasa jest reprezentacją modelowanego pojęcia. Norma ISO 19100 uznaje klasy jako specyfikację, a nie implementację. Klasa reprezentowana w formie graficznej, składa się z trzech fragmentów: nazwy, atrybutów i metod.

Atrybuty klas niosą informację o obiektach danej klasy. Atrybut klasy przedstawiony jest w następującym formacie:

```
<<stereotyp>> [dostępność] nazwa [wielokrotność] [:typ]
[= wartość początkowa] [{właściwości}]
```

Typem atrybutu może być zarówno jeden z typów wbudowanych (`Float`, `Integer`, `Void` i inne), jak również klasa wyspecyfikowana w ramach modelowania.

Modyfikatory dostępu określają prawa do korzystania z elementów klasy. Można wyróżnić następujące modyfikatory dostępu:

- publiczny (+) - pełny dostęp dla wszystkich obiektów systemu,

- prywatny (-) - dostęp jedynie dla obiektów tej samej klasy,
- chroniony (#) - dostęp dla obiektów tej samej klasy i obiektów klas dziedziczących.

Relacje określają związki pomiędzy klasami. W formie graficznej reprezentowane są one przez linie łączące klasy, zakończone odpowiednimi grotami. Wyróżnia się następujące rodzajów relacji:

- asocjacja - semantyczne połączenie pomiędzy dwoma instancjami,
- generalizacja - relacja pomiędzy elementami i podelementami,
- zależność - użycie jednego elementu przez drugi,
- realizacja - dziedziczenie klasy abstrakcyjnej i jej implementacja,
- agregacja - relacja typu „całość-część", mówi z jakiego elementu składa się klasa,
- kompozycja - silniejszy typ agregacji, w którym dzieci są usuwane, jeśli usunięty jest rodzic.

5.2.3. Profile

Profile to specyfikacje składają się z kombinacji elementów zaczerpniętych z jednego lub wielu standardów bazowych, takich jak klasy, opcje, parametry, niezbędnych do realizacji określonych funkcji. Z faktu zgodności z profilem wynika zgodność ze zbiorem standardów bazowych, które ten profil realizuje. Jednakże spełnienie warunków zgodności dla standardów bazowych nie oznacza spełnienia warunków zgodności z profilem. Profile charakteryzują się tym, że:

- mogą ograniczać wybór opcji zdefiniowanych w normach bazowych do zakresu niezbędnego do osiągnięcia celu określonemu prze zprofil, przy czym opcje standardu bazowego mogą zostać zachowane jako opcje profilu;
- nie powinny wprowadzać żadnych wymagań, które powodowałyby niezgodność ze standardami bazowymi, do których się odnoszą;
- mogą zawierać bardziej szczegółowe wymagania zgodności, niż te zdefiniowane w standardach bazowych.

Ponadto każdy profil powininno cechować:

- określenie zasięgu funkcji, które zapewnia profil, wraz z wymaganiami dla użytkownika;
- określenie przypadków, w których stosowany jest profil oraz, gdy jest to konieczne, opisanie wszystkich interfejsów;
- stanowisko wspólnoty interesów społeczności, dla której adresowany jest profil;
- normatywne odniesienie do standardów bazowych i profili, zawierających dokładne rozpoznanie zastosowanych rzeczywistych treści standardów i profili, z uwzględnieniem zastosowanych poprawek, korekt technicznych, zgodności, które rozpoznane zostały jako mające potencjalny wpływ na interoperatywność i przenośność użycia profilu;

- określenie zakresów standardów bazowych czy profili odniesienia obowiązujących w profilu, uwzględniające opis wyboru klas, zgodności, wyboru opcji, zakresu wartości parametrów;
- określenie wymagań, które muszą być spełnione przez system lub zbiór danych, aby stwierdzić jego zgodność z profilem, uwzględniająca opcje zabronione;
- jeśli istotne, odniesienie do specyfikacji testów zgodności dla profilu.

Profile stosuje się z następujących powodów:

- do identyfikacji standardów bazowych, wraz z odpowiednimi klasami, opcjami i parametrami, które są niezbędne do utworzenia konkretnych funkcji, np. w celu osiągnięcia interoperacyjności;
- do zapewnienia dostępności do wybranych części grup standardów bazowych w celu implementacji komponentów rzeczywistego systemu;
- do zapewnienie uniwersalności i poprawności testów zgodności systemów zaimplementowanych zgodnie z profilem.

Wymagania zgodności z profilem można podzielić na wymagania obligatoryjne (muszą być spełnione w każdym przypadku) i wymagania opcjonalne (wybrane w zależności od implementacji). Wymagania zgodności mogą być definiowane bezwarunkowo (w tym przypadku wymagania obligatoryjne i opcjonalne nie są ograniczane żadnymi warunkami) i warunkowo (wymagania powinny być spełnione, jeśli zastosowane są odpowiednie warunki). Można również wyróżnić podział na wymagania pozytywne (co powinno być wykonane) i negatywne (czego nie powinno się robić).

5.3. Weryfikacja modeli na poziomie konceptualnym

Walidacja modelu konceptualnego polega na stwierdzeniu, czy jego podstawy mają oparcie w uznanych teoriach i twierdzeniach, oraz czy sam model jest racjonalny dla planowanego obszaru zastosowań. Walidację modelu na poziomie warstwy M2 można zdefiniować jako ocenę stopnia jego zgodności z jakąś wcześniej zdefiniowaną abstrakcją. Tworząc nowy model systemu w oparciu o wcześniej zdefiniowany, bazowy meta-model systemu (model modelu), możliwe jest określenie stopnia jego zgodności ze wzorcem, a także jego poprawność już na poziomie konceptualnym. Przykładowo, posiadając abstrakcyjny uczelni wyższych, oparty o standardową strukturę i zależności wewnętrzne, można walidować konkretne modele rodzajów uczelni (publiczne, techniczne itp.). Weryfikacja może odbywać się na bazie trzech głównych cech modeli:

- ograniczeń – przez porównywanie ograniczeń z analogicznymi w modelu abstrakcyjnym,
- struktury – przez badanie występowania konkretnych składowych modelu,
- zależności – przez sprawdzenie zależności pomiędzy poszczególnymi składowymi modelu.

Przy porównywaniu modeli w kontekście ograniczeń, określa się dwie kategorie modeli: modele ograniczające i modele rozszerzające. Pierwsza z kategorii zakłada zawieranie się ograniczeń modelu badanego w ograniczeniach abstrakcji, podczas gdy druga kategoria przyjmuje pokrycie ograniczeń modelu abstrakcji i ewentualne jego rozszerzenie o ograniczenia dodatkowe. Więcej szczegółów dotyczących klas zgodności modeli przedstawiono w podrozdziale 5.3.1, natomiast metod walidacji - w podrozdziale 5.3.2.

5.3.1. Klasy zgodności modelu

Rezultatem procesu walidacji modelu konceptualnego jest przyporządkowanie walidowanego modelu do jednej z klas zgodności. Klasa zgodności modelu niesie informację, w jakim stopniu testowany model odpowiada strukturze i ograniczeniom modelu bazowego. Dzięki temu można określić, czy spełnione zostały założenia projektowe oraz jakich elementów brakuje. Ze względu na zróżnicowanie problematyki i specyfiki modelowania, trudno jest zaproponować jedną, ogólną, zunifikowaną metodę tworzenia klas zgodności. W większości przypadków klasy zgodności definiowane są indywidualnie, odpowiednio dla rozpatrywanego zadania. Zazwyczaj wyróżnia się następujące typy klas zgodności:

- Klasa podstawowa (poziom zgodności 0) – model należący do klasy podstawowej (spełniający założenia na poziomie zgodności 0) zawiera jedynie elementy obligatoryjne oraz spełnia ograniczenia niezbędne do implementacji modelu. Gdy w procesie walidacji model nie zostanie zakwalifikowany do tej klasy, oznacza to, że jest niezgodny z modelem bazowym i konieczna jest jego korekta.
- Klasy rozszerzone (poziom zgodności 1..n-1) – modele przyporządkowywane do klas rozszerzonych posiadają zdefiniowane elementy podstawowe oraz elementy rozszerzające. Jednym ze sposobów dalszej kategoryzacji jest przyporządkowywanie modeli do klas numerowanych, o numerze określającym poziom zgodności.
- Klasa pełna (poziom zgodności n) – do klasy pełnej (o najwyższym poziomie zgodności) przyporządkowywane są te modele, które posiadają zarówno elementy niezbędnie wymagane, jak również wszystkie rozszerzenia. W szczególnym przypadku, gdy model posiada tylko jedno dostępne rozszerzenie (klasę, strukturę danych) klasa pełna staje się tożsamą z klasą rozszerzoną. Gdy model zawiera jedynie elementy obligatoryjne, wyróżnić można tylko jedną klasę, która jest zarówno klasą podstawową jak i klasą pełną.

Ze względu na fakt, że niektóre modele konceptualne zawierają elementy mogące pojawiać się wielokrotnie (np. uczelnia może posiadać wiele wydziałów, robot wiele przegubów itp.) jednoznaczne przyporządkowanie modelu do jednej z klas może okazać się niemożliwe.

5.3.2. Walidacja modelu

Proces walidacji, szczegółowo opisany w [8], składa się z kilku etapów. W ich trakcie poprzez testowanie sprawdzana jest zgodność modelu z modelem referencyjnym według zadanych kryteriów. Przez pojęcie zgodności rozumiane jest

wypełnienie określonych wymagań. Wymagania formalnie opisywane są w klauzulach zgodności (które określając warunki konieczne do tego, aby dana implementacja była zgodna ze standardem). Biorąc pod uwagę obligatoryjność, wymagania dotyczące zgodności można podzielić na trzy kategorie (podobnie jak przy ocenie zgodności z profilem):

- wymagania obowiązkowe – czyli te, które muszą wystąpić we wszystkich klasach;
- wymagania warunkowe - spełnione, jeżeli zastosowane są warunki opisane w specyfikacji oraz
- wymagania opcjonalne - mogę być wybrane, by pasować do implementacji, jeżeli wymagania użyte w stosunku do opcji są przestrzegane.

Testowanie zgodności pozwala określić stopień, w jakim dany model (implementacja) odpowiada formalnej specyfikacji. Można wyróżnić dwa typy takich testów:

- testy podstawowe – powinny być używane w celu stwierdzenia jednoznacznych przypadków niezgodności oraz jako etap wstępny, by stwierdzić czy należy wykonać dalsze testy zdolności. Testów podstawowych nie należy stosować by ostatecznie stwierdzić czy dana implementacja jest spójna z modelem. Za ich pomocą nie można również zagwarantować wykrycia przyczyn usterki.
- testy zdolności – są stosowane, żeby stwierdzić czy zdolności badanej implementacji są spójne z wymaganiami oraz w celu wykrycia przyczyn błędów i awarii. Nie powinno się ich stosować do szczegółowego testowania zachowania związanego z zaimplementowanymi zdolnościami. Nie służą one także do gwaranowania kompletności implementacji.

Ponadto do popularnych testów należą:

- testy poprawności sformułowania problemu – poprawność sformułowania problemu gwarantuje, że podczas tworzenia modelu zostały zastosowane odpowiednie opisy struktur danych oraz dodatkowych reguł i ograniczeń. Część z reguł zdefiniowanych jest w specyfikacji języka UML, natomiast pozostałe są tworzone przez twórców konkretnego systemu.
- testy poprawności składni i zgodności typów – poprawność składni potwierdza zgodność specyfikacji modelu z gramatyką języka, w której model został utworzony (UML i OCL), a zgodność typów zapewnia, że każde wyrażenie opisane jest tylko za pomocą typów istniejących w modelu.
- testy odporności – wykrywają braki lub elementy nieokreślone w modelu i sprawdzają czy system będzie działał w przypadku pominięcia pewnych części modelu. Odporność jest sprawdzana poprzez zastosowanie modelu do opisu kolejnych instancji danego problemu.
- testy adekwatności.

5.4. Przykład walidacji modelu konceptualnego

Niniejszy podrozdział poświęcono analizie problemu walidacji nieskomplikowanych modeli konceptualnych. W standardowym podejściu do procesu walidacji dokonuje się weryfikacji zgodności profili (modele rozszerzające / zwężające model bazowy) z modelem referencyjnym (model bazowy). Poniżej opisano podejście odwrotne. Występuje w nim jeden profil, który jest porównywany z co najmniej jednym modelem referencyjnym. Wynikiem tak przeprowadzonej walidacji jest informacja o stopniu zgodności profilu z danym modelem referencyjnym, co może świadczyć o posiadaniu przez walidowany model pewnej określonej cechy.

Analizę problemu walidacji przeprowadzono na przykładzie z dziedziny robotyki. Na rysunkach 5.2 i 5.3 przedstawiono dwa wykorzystywane modele referencyjne, zawierające, odpowiednio: definicję robota oraz definicję cech robota (tj. dynamikę zachowania, sterowalność i obserwowalność). Na rysunku 5.4 przedstawiono profil przeznaczony do walidacji na podstawie powyższych modeli referencyjnych. Rzecz jasna przyjęte modele nie muszą być w zupełności zgodne z rzeczywistością ani w pełni wyczerpywać rzeczywistej definicji określonego zagadnienia, gdyż reprezentują one wyłącznie koncepcję. W modelach występują przede wszystkim związki przynależności, takie jak agregacja i kompozycja, oraz związki uogólniające, takie jak generalizacja.

5.4.1. Model referencyjny - definicja robota

Jeżeli chodzi o zaproponowaną koncepcję definicji robota, z modelu (rysunek 5.2) można wywnioskować, że każdy obiekt będzie spełniał definicję robota, jeżeli będzie posiadał wyłącznie jeden korpus z przegubami (*KorpusPrzeg*) oraz, że do tego korpusu będzie należeć dowolna liczba manipulatorów (*KorpusPrzeg*) oraz dokładnie jeden układ odpowiedzialny za lokomocję. Można zauważyć, że wedle takiej definicji i wedle reguł ilościowych ograniczeń, robotem jest również obiekt nieposiadający ani manipulatorów ani układu lokomocji.

5.4.2. Model referencyjny - definicja cech robota

Model zaprezentowany na rysunku 5.3 definiuje 3 cechy robota: dynamikę zachowania, sterowalność i obserwowalność. O sterowalności decyduje fakt posiadania przez klasę `Przegub` przynajmniej jednej klasy `Aktuator`, obserwowalność w podobny sposób nierozerwalnie łączy się z klasą `Czujnik`, natomiast za cechę zachowania dynamicznego odpowiada występowanie abstrakcyjnej klasy `BrylaSztywna`, definiującej kształt oraz parametry bezwładnościowe bryły sztywnej. Podobnie jak zmienne klasy `BrylaSztywna`, również ograniczenia ilościowe mają sens fizyczny. W przypadku przegubu, może on mieć od 1 do 6 stopni swobody, a więc musi iść za tym w parze taka sama liczba aktuatorów, jak i czujników. Tylko wtedy można mówić o występowaniu cech sterowalności i obserwowalności. Co więcej, każdy łańcuch kinematyczny, który wchodzi w skład niemal każdego robota, jest w uproszczeniu naprzemiennym złożeniem

bryły sztywnej oraz przegubu. Stąd też ograniczenie co do liczebności instancji klasy `BrylaSztywna` i `Przegub` składających się na klasę `BrylaPrzeg`.

Wszystkie opisane do tej pory ograniczenia dotyczyły związków pomiędzy klasami modeli. Jednak ograniczenia można również definiować w samych klasach. Tego typu ograniczenie, czego nie widać na rysunku 5.3, nałożono na klasę `Przegub`

```
inv: Ogr[0].MaxPredkosc <= 1000 .
```

gdzie `inv` (ang. *invariant*) oznacza ograniczenie typu niezmiennik w zapisie języka OCL. Należy podkreślić, że jest to wyłącznie ograniczenie demonstracyjne i nie ma uzasadnienia fizycznego. Można je zinterpretować tak, że konkretna instancja klasy `Przegub` nie spełnia kryteriów bycia przegubem, jeżeli maksymalne ograniczenie prędkość ruchu nałożone na pierwszą współrzędną stanu przekracza wartość 1000.

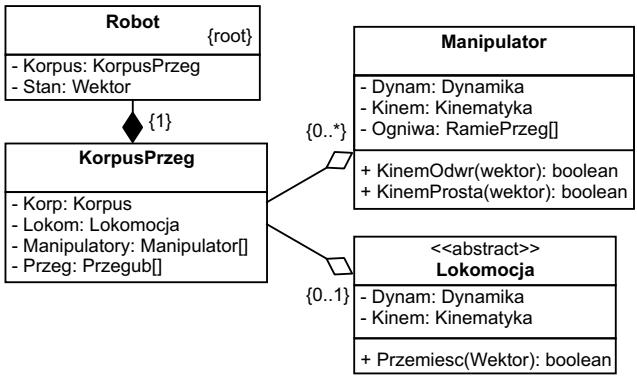
5.4.3. Model rozszerzający - robot mobilny typu Monocykl

Opisane wyżej modele są modelami referencyjnymi i w zamyśle zaproponowanego przebiegu procesu walidacji, mają one służyć do weryfikacji występowania w modelu rozszerzającym określonych właściwości. Na rysunku 5.4 zaprezentowano model konceptualny robota mobilnego typu monocykl. Jak można zauważyć, z punktu widzenia struktury klas i związków między klasami, jest to model będący zmodyfikowanym połączeniem modeli z rysunków 5.2 i 5.3. Występują w nim jednak różnice stanowiące wyzwanie dla algorytmów walidujących. Zasadniczą różnicą jest występowanie klas `UkladJezdny` i `Kolo`, dziedziczących po kasach `Lokomocja` i `BrylaSztywna`. Klasy bazowe de facto zostały zastąpione klasami pochodnymi, gdyż biorą one udział w tych samych relacjach, co pierwotnie klasy bazowe. Kolejną znaczącą różnicą jest wystąpienie w miejscu klasy `BrylaPrzeg` (rysunek 5.3) klasy o nazwie `KoloPrzeg`. Algorytm walidujący w sytuacji tego typu rozbieżności nie może polegać tylko i wyłącznie na zgodności nazw klasy. Decydująca wówczas powinna być zgodność klas znajdujących się niżej w hierarchii. W modelu rozszerzającym nie występują ponadto ograniczenia określające przedział wartości / ilości. Ograniczenia związków jak i ograniczenie występujące w klasie `Przegub` (`inv: Ogr.MaxPredkosc = 200`) mają konkretne wartości.

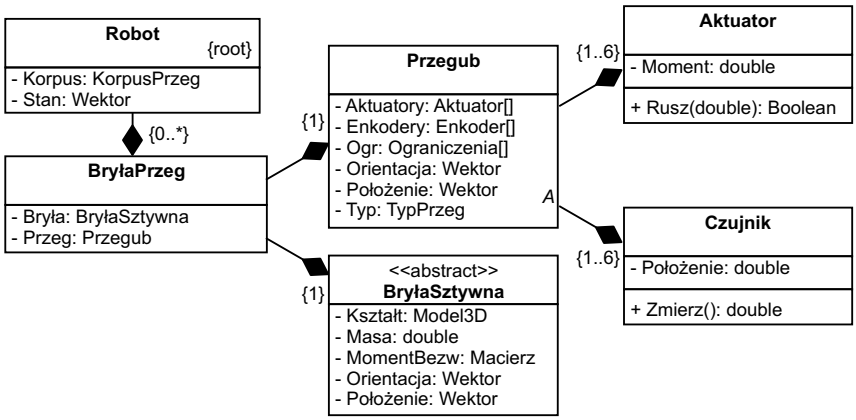
Jak widać model rozszerzający konstruowany był tak, by poprawnie działający algorytm walidujący był w stanie zaklasyfikować go jako spełniający założenia zarówno definicji robota (rysunek 5.2) jak i cech sterowalności, obserwowalności oraz dynamiki zachowań (rysunek 5.3).

5.4.4. Programowe przetwarzanie modeli UML

Praktyczne podejście do walidacji modeli na poziomie konceptualnym można sprowadzić do implementacji programu komputerowego, który zrealizuje to zadanie. Proces walidacji rozpoczyna się od wczytania porównywanych modeli zapisanych w ustandaryzowanej formie (np. w formacie XMI) do odpowiednich



Rys. 5.2: Przykładowy model konceptualny definicji robota.

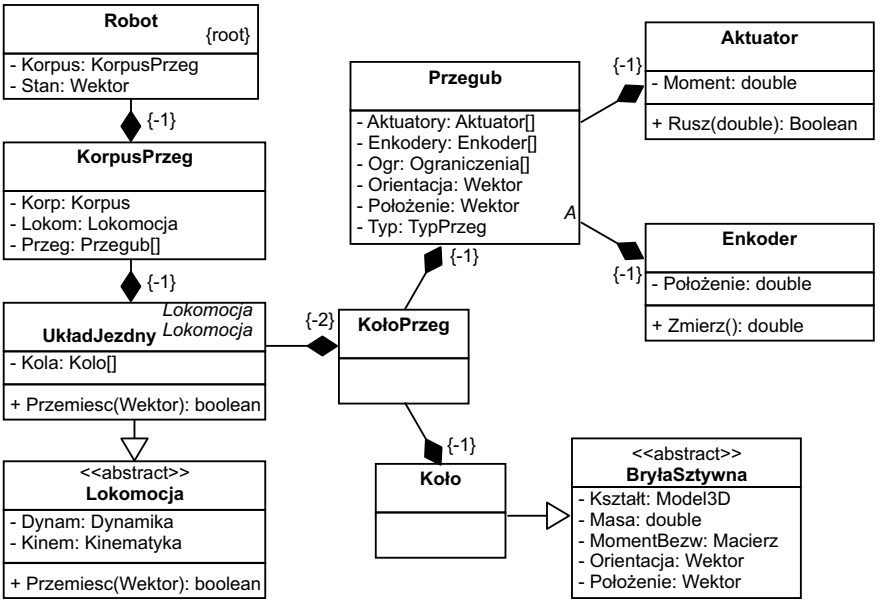


Rys. 5.3: Przykładowy model konceptualny cech robota - dynamika, sterowalność i obserwowalność.

struktur umożliwiających wygodne i szybkie przeprowadzenie testów. Przykładowe propozycje takich struktur zawiera podrozdział 5.4.5.

Weryfikacja zgodności modeli może być zrealizowana za pomocą odpowiednich funkcji, które porównują wykorzystane struktury danych pod kątem kilku różnych właściwości (podrozdział 5.4.6). Ostatni etap procesu polega na reprezentacji otrzymanych wyników, ich interpretacji, a także wyciąganiu wniosków na temat klas zgodności. Zostało to opisane w podrozdziale 5.4.7. Opisywany program został zaimplementowany w języku C++, jednakże bez trudu mógłby zostać zrealizowany w innych obiektowych językach programowania.

Jednym z założeń projektu opisywanego w dalszej części niniejszego rozdziału było zaimplementowanie mechanizmów do walidacji modeli reprezentowanych w standardzie UML. Realizacja tego założenia wymagała, w pierwszej kolejności, wykonania konwersji modelu z postaci graficznej do reprezentacji w języku XML (a dokładniej do reprezentacji w języku XMI w wersji 2.1).



Rys. 5.4: Przykładowy model konceptualny konkretnego robota mobilnego typu monocykl.

Problem dostępu do kluczowych, z punktu widzenia celów projektu, informacji wymagał zagłębienia się w hierarchiczną strukturę znaczników i ich atrybutów formatu XMI, oraz zlokalizowania tych informacji w otoczeniu ogromu informacji nieistotnych. Na samym szczycie w hierarchii pliku w formacie XMI znajdują się dwa znaczące obszary `uml:Model` i `xmi:Extension`:

```
<xmi:XMI ...>
...
<uml:Model ...>
...
</uml:Model>

<xmi:Extension ...>
...
</xmi:Extension>
</xmi:XMI>
```

gdzie `...` oznacza listę atrybutów jak i wypełnienie danego obszaru elementami niższymi w hierarchii. Jeżeli chodzi o otoczenie objęte znacznikami `<uml:Model ...> ... </uml:Model>` znajdują się tam definicje klas jak i definicje związków pomiędzy poszczególnymi klasami. Każdy z tych elementów oznaczony jest jako

```
<packagedElement xmi:type="..." xmi:id="..."
```

```
name="..." visibility="...">
```

gdzie istotne atrybuty to

- `xmi:type`, który posiada wartość np. `uml:Class` bądź `uml:Association` w przypadku definicji odpowiednio klasy oraz związku asocjacji,
- `xmi:id`, który jest identyfikatorem definiowanego obiektu obowiązującym na przestrzeni całego dokumentu,
- `name`, jeżeli definiowany obiekt posiada nazwę, np. nazwa klasy, wówczas jest ona zapisana jako wartość atrybutu `name`.

Każdy z elementów `packagedElement` zawiera elementy potomne będące definicjami np. zmiennych i metod należących do danej klasy, jednak pomimo swojej istotności z punktu widzenia samej klasy, pola te nie są przedmiotem analizy w procesie walidacji. Wszystkie istotne informacje biorące udział w tym procesie odczytywane są z obszaru objętego znacznikami `<xmi:Extension ...> ... </xmi:Extension>`, w którym elementy potomne odnoszą się do definicji z obszaru `uml:Model`.

Obszar `uml:Extension`, jak sama nazwa wskazuje, jest rozszerzeniem obszaru `uml:Model`, ponieważ z jednej strony powiela część informacji z `uml:Model`, a z drugiej strony uzupełnia ją o znaczną ilość parametrów, np. ograniczenia klas i związków oraz informacje o pozycji klasy w tworzonym przez model drzewie (atrybuty `isRoot` i `isLeaf`). A zatem odczytując z definicji klasy (obszar `uml:Model`) identyfikator `xmi:id` odszukiwany jest jego odnośnik w `uml:Extension` poprzez atrybut `xmi:idref` znacznika

```
<element xmi:idref="..." xmi:type="..." name="..." scope="...">
```

skąd natychmiastowo otrzymywana jest nazwa `name` klasy. W następnej kolejności z podobiektów znacznika `element`

```
<properties isSpecification="..." sType="..." nType="..."
            scope="..." isRoot="..." isLeaf="..."
            isAbstract="..." isActive="..."/>
```

oraz

```
<constraints>
<constraint name="..." description="..." type="..."/>
</constraints>
```

odczytywane są odpowiednio atrybuty `isRoot` (informuje czy dana klasa jest korzeniem w drzewie struktury grafowej) oraz `description` (reguła ograniczeń języka OCL). Wyżej wymienione parametry, a więc nazwa, identyfikator, reguła ograniczenia, oraz flaga `isRoot`, zostają zapisane w zaalokowanym wcześniej obiekcie klasy `AbstractObject` (rysunek 5.5).

Posiadając zbiór obiektów klasy `AbstractObject` możliwe jest utworzenie struktury grafowej. Graf reprezentowany jest przez listę sąsiadów w postaci wektora wskaźników na `AbstractObject`. Jest on generowany w oparciu o informacje dotyczące związków (połączeń) z podobszaru znacznika `element`, otoczenia

```
<links>
<Aggregation xmi:id="..." start="..." end="..."/> ,
</links> .
```

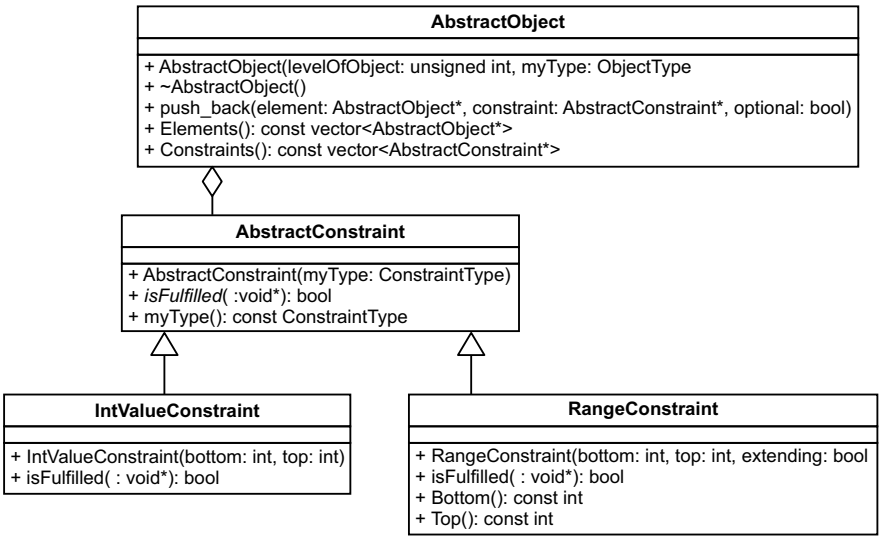
Na podstawie atrybutów `start` i `end` określana jest klasa znajdująca się po drugiej stronie związku łączącego, oraz kierunek zależności. Bezpośrednio przekłada się to na strukturę grafu tworząc w ten sposób krawędzie skierowane. Co więcej, podobnie jak w przypadku klas, możliwe jest nałożenie ograniczeń również na związki pomiędzy klasami. Informacja na temat ograniczeń związków znajduje się jako atrybut znacznika `constraints` hierarchicznie usytuowanego w

```
<xmi:Extension ...>
<connectors>
<connector xmi:idref="...">
<source xmi:idref="...">
...
<constraints constraint="..."/>
...
</source>
<target xmi:idref="...">
...
<constraints constraint="..."/>
...
</target>
</connector>
</connectors>
</xmi:Extension>
```

gdzie, jak można się przekonać, otoczenie `source` odpowiada atrybutowi `start`, natomiast `target` odpowiada atrybutowi `end` z wyżej prezentowanego otoczenia `<links>`.

5.4.5. Struktury danych do przechowywania modeli

Ze względu na różnorodność modeli UML zdefiniowanie prostej struktury do przechowywania różnego rodzaju danych jest zadaniem nietrywialnym. Zależności pomiędzy obiektami w modelu przyjmują różnorodne formy, w związku z czym nie sposób bezpośrednio zastosować je wszystkie. Podczas pracy nad programem służącym do walidacji zaproponowano wykorzystanie klasy reprezentującej abstrakcyjny obiekt, który reprezentuje wszystkie możliwe dane pojawiające się w modelu (nie jest abstrakcyjny w sensie języka C++, ponieważ tworzymy obiekty tej klasy). Każdy z obiektów posługując się typem enumerycznym definiuje swoją własną tożsamość, dzięki czemu przechowujemy informację o typie i charakterze obiektu (czy jest to pole `int`, `double` czy obiekt modelu zawierający



Rys. 5.5: Diagram klas służących do porównywania modeli.

inne obiekty). Diagram klas wykorzystanych przy implementacji testów zgodności znajduje się na rysunku 5.5.

Każda instancja klasy `AbstractObject` może zawierać wektor wskaźników na obiekty tej klasy, z którymi jest ona powiązana. Pozwala to na realizację drzewiastej struktury, która ma w założeniu przypominać wczytywany model reprezentując jego parametry i właściwości. Dla wszystkich relacji pomiędzy klasami wprowadzona została możliwość zdefiniowania połączeń opcjonalnych, które mogą jedynie pozytywnie wpłynąć na stopień zgodności modeli.

Relacje pomiędzy obiektami mogą być obwarowane ograniczeniami, w związku z czym dodatkowo zaimplementowana została abstrakcyjna klasa `AbstractConstraint`, która zawiera wirtualną metodę `IsFulfilled`. Metoda ta musi zostać zaimplementowana w klasach pochodnych (odpowiadających za konkretne ograniczenia) i służy do weryfikacji spełnienia ograniczenia dla zadanej wartości (bądź zakresu). Pojedyncze ograniczenie może odnosić się do wartości pola, zakresu ograniczeń (zwężający bądź rozszerzający) czy też ilości i relacji pod-obiektów.

5.4.6. Metody porównywania modeli

Dla przygotowanych struktur danych możliwa jest implementacja metod porównujących modele przekazane jako argumenty. Porównywanie modeli może odbywać się na kilku płaszczyznach, zarówno pod kątem samej struktury jak i zależności pomiędzy obiektami. Porównywanie modeli opiera się na powszechnie znanej metodzie przeszukiwania grafów – przeszukiwania w głąb (ang. *Depth First Search*). W algorytmie tym w pierwszej kolejności rozpatrywane są węzły leżące najgłębiej w grafie, co zostało zaimplementowane poprzez rekurencyjne formę metody porównującej.

Wszystkie przygotowane metody porównujące rekurencyjnie wywołują się na kolejnych pod-obiektach (jest to możliwe dzięki drzewiastej reprezentacji modelu), a wyniki z kolejnych poziomów są uśredniane dla konkretnego obiektu. W ten sposób otrzymywane jest swego rodzaju wartościowanie składowych modelu, od tych położnych najwyżej do tych położonych najniżej w drzewie.

Podstawową cechą wszystkich zaimplementowanych testów jest zwracanie w wyniku pełnej zgodności modeli w przypadku przekazania im jako argument dwóch identycznych modeli. Rozbieżności w modelach, w zależności od charakteru, zmniejszają otrzymany rezultat obliczeń.

Weryfikacja struktury

Weryfikacja struktury modeli jest najprostszym i najmniej precyzyjnym z porównań – sprawdzana jest jedynie zbieżność zależności obiektów w modelu, bez wnikania w ich charakter i konsekwencje. Na każdym etapie drzewa liczona jest liczba pod-obiektów jednego i drugiego modelu. W przypadku zgodności zwracana jest wartość 0, natomiast w przypadku przeciwnym wartość 1. Podobne obliczenia wykonywane są rekurencyjnie na pod-obiektach klasy, a oba wyniki uśredniane są i końcowa wartość zwracana jest wyższej warstwie drzewa.

Weryfikacja ograniczeń

Weryfikacja ograniczeń odbywa się na podobnej zasadzie co weryfikacja struktury, jednakże do wyniku końcowego nie jest brana liczba elementów struktury, a stopień zgodności kolejnych pod-obiektów modelu. Ograniczenia weryfikowane są po kolei, w związku z czym w przypadku niskiej zgodności strukturalnej modelu wynik tej weryfikacji również wypadnie dość nisko (miarodajne wyniki otrzymywane są dla dużych wartości strukturalnej zgodności modeli).

Ograniczenia w modelu uznawane są za zgodne, jeśli zarówno typy, jak i wartości ograniczeń są takie same w obu modelach. Definiuje się w tym przypadku ograniczenia zwężające i rozszerzające (opisane w rozdziale 5.3). Porównanie konkretnych ograniczeń odbywa się za pomocą metody `IsFulfilled`, jednakże przy ograniczeniach zakresu istotne jest przekazanie modelu abstrakcyjnego z modelem porównywanym w odpowiedniej kolejności.

Weryfikacja zależności

Porównanie zależności obiektów w walidowanych modelach również zakłada ich dość dobrą strukturalną zgodność. To podejście do testowania polega na dokładnej analizie rodzajów relacji występujących pomiędzy obiektami modelu. Przykładowo, w obu przypadkach konkretny pod-obiekt klasy musi być prywatny, opcjonalny i być typu `int`. Weryfikacja ta zwraca z reguły dość niski procent zgodności, jednakże być skutecznie wykorzystana do walidacji bardzo zbliżonych modeli w celu eliminacji jakichkolwiek błędów.

5.4.7. Interpretacja wyniku porównania

Trzy testy zgodności przedstawione w poprzednim rozdziale zwracają w wyniku wartość z przedziału od 0 do 1, przy czym zero oznacza kompletną niezgodność, a jeden pełną zgodność modeli. Wartości pośrednie z tego przedziału umożliwiają interpretację otrzymanych rezultatów w formie procentów, które dla złożonych modeli mogą mieć dość zróżnicowane wartości.

Procentowa reprezentacja danych prowadzi do wygodnego klasyfikowania modeli według stopnia zgodności ze zdefiniowaną abstrakcją. Zastosowanie trzech metod zwracających różne wyniki umożliwia utworzenie wielu klas zgodności, w zależności od liczby przyjętych progów dla każdego z testów. Przykładowo, definiując zgodność w najprostszej postaci (binarnie) na klasy zgodne i niezgodne, otrzymujemy:

- klasę strukturalnie zgodną i niezgodną z abstrakcją,
- klasę zgodną i niezgodną z abstrakcją w sensie ograniczeń,
- klasę zgodną i niezgodną z abstrakcją w sensie zależności.

Już dla tak prostego podziału można wyróżnić 8 klas zgodności.

5.5. Podsumowanie

W ramach projektu dokonano przeglądu literatury i usystematyzowania informacji z dziedziny modeli konceptualnych. Opisany został standard *Meta-Object Facility*, czyli sposób tworzenia systemów w oparciu o model, jak również język schematów konceptualnych - UML - wykorzystywany do opisu niższych warstw modelu MOF. Opisano także specyfikę profili, ich strukturę, właściwości i zastosowania.

Kolejnym aspektem części teoretycznej projektu było zagadnienie walidacji i badania zgodności modeli konceptualnych. Na podstawie dostępnej literatury omówiony został proces weryfikacji modeli, w szczególności kryteriów i metod testowania. Zaproponowano również własny podział klas zgodności modelu.

Jednym z elementów projektu było przygotowanie oprogramowania służącego do walidacji modeli konceptualnych i badania stopnia zgodności. Program został zaimplementowany przy użyciu języka C++ oraz biblioteki `Xerces` wykorzystanej do implementacji parsera plików `.xmi`. Do badań zgodności profilu z różnymi modelami referencyjnymi przygotowany został profil oraz dwa modele konceptualne robota, z którymi sprawdzana była jego zgodność. Modele opisane zostały według standardu XMI w wersji 2.1.

Literatura

[1] F. Schneider. UML and Model Checking. In *Proc. Fifth Langley Formal Methods Workshop*, Williamsburg, VA; United States, 1999.

[2] Technical Committee ISO/TC 211, Geographic information/Geomatics. Draft Technical Specification 19103, Geographic information – Conceptual schema language.

[3] Technical Committee ISO/TC 211, Geographic information/Geomatics. International Standard ISO 19106; Geographic information – Profiles.

[4] Technical Committee ISO/TC 211, Geographic information/Geomatics. International Standard ISO 19105, Geographic information – Conformance and testing.

[5] Computer Science Lab Wiki. `http://ai.ia.agh.edu.pl/wiki/`.

[6] C.M. Keet. A formal comparison of conceptual data modeling languages. In *13th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'08)*, volume 337, pages 25–39, Montpellier, France, 16-17 June 2008. CEUR-WS.

[7] Object M. Group. OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. `http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF`, November 2007.

[8] E. Pakalnickiene and L. Nemuraite. Checking of conceptual models with integrity constraints. *Information technology and control*, 36(3):285–294, 2007.

ROZDZIAŁ 6

PROJEKT I ZASTOSOWANIE PROSTEJ ONTOLOGII

B. Kułak, M. Wcisło, Ł. Żygadło

6.1. Wstęp

Termin ontologia wywodzi się z filozofii, a jego oryginalne znaczenie wiąże się z problemem łączenia i analizy przedmiotów o podobnych cechach czy sposobie istnienia. Obecnie jednak termin ten często jest kojarzony z inżynierią wiedzy, gdzie występuje w kontekście klasyfikacji i hierarchizacji informacji o otaczającym nas świecie w metajęzyku zrozumiałym jednocześnie dla człowieka i maszyny [1].

W niniejszym rozdziale omówiono, wzorując się na [2], zagadnienia związane z pojęciem ontologii oraz elementami i mechanizmami pozwalającymi na przeprowadzanie wnioskowania w ontologiach. Dokonano przeglądu algorytmów wnioskujących oraz opisano przykład dziedzinowej ontologii, pozwalającej na tworzenie bazy wiedzy o tematyce żywnościowej.

6.2. Ontologia

Pierwszym badaczem ontologii był Arystoteles (IV w. p.n.e.). Jednak termin ontologia w nowoczesnym znaczeniu został ukuty w 1613 r. i równolegle wykorzystany przez dwóch filozofów: Rudolpha Goclenius (Göckel) (*Lexiconphilosophicum*) i Jacoba Lorhard (*Theatrum philosophicurn*). Na początku lat osiemdziesiątych ubiegłego wieku termin ten pojawił się w dziedzinie technologii informacyjnych. W 1980 r. John McCarthy zauważył konieczność opisu otaczającego świata w formie ontologii. Podejście to przez następne zyskiwało na popularności i w 1993 r. doczekało się pierwszej próby formalizacji przez Toma Grubera.

Istnieje wiele definicji ontologii. Najbardziej trafną wydaje się być definicja Grubera [3]: „Ontologia to wyraźna specyfikacja konceptualizacji". Definicja ta została następnie rozwinięta przez Borsta [4]: „Ontologie są formalną specyfikacją współdzielonej konceptualizacji". Całość uzupełnia opis Studera i współpracowników [5]:

- **konceptualizacja** odnosi się do abstrakcyjnego modelu pewnego zjawiska z otaczającego świata zidentyfikowanego poprzez koncepcje nieodłączne dla niego,
- **wyrazistość** oznacza, że typ używanego konceptu i ograniczeń z nim związanych jest wyraźnie zdefiniowany,
- **współdzielność** oznacza, że notacja zapisu ontologii nie jest prywatna dla danego zagadnienia ale jednolita dla całej grupy.

Na początku ontologia miała być odpowiedzią na próbę znalezienia struktury klas opisującej cały wszechświat lub sprawdzenia, czy jest to możliwe. Z chwilą powstania technologii informacyjnych ontologie przyczyniły się do zmiany obiektu zainteresowań wielu inżynierów oprogramowania. Dostrzegli oni, że o wiele ważniejsze jest skupienie się na strukturze danych i ich reprezentacji niż na samych funkcjach oprogramowania. W późniejszych czasach, automatyczne wnioskowanie uznano za bardziej cenną cechę niż samo przechowywanie wiedzy i pojęcie ontologii zaczęto łączyć ze sztuczną inteligencją. Zauważono też, że oprócz budowy samych ontologii równie istotne są takie zagadnienia, jak: uczenie ontologii, ocena ontologii, ewolucja ontologii, czy łączenie ontologii [6].

Pojęcie ontologii wiąże się z inżynierią wiedzy - niedawno powstałą dziedziną zajmującą się problemami zarządzania wiedzą. Ontologie, poprzez możliwość strukturalnego opisu wiedzy, pozwoliły rozwiązać wiele problemów wynikających z ograniczeń występujących w tradycyjnym sposobie przechowywania danych w bazach danych oraz występujących w modelowaniu dziedziny za pomocą obiektowych języków programowania. Obecnie ontologie znajdują szerokie pola zastosowań. O ich dojrzałości świadczy duże zainteresowanie ze strony komercyjnych użytkowników i dużych instytucji. Wykorzystują oni ontologie jako narzędzie wspierające budowę inteligentnych interfejsów użytkownika w określonych dziedzinach, np. do: przeglądania produktów, zarządzania dokumentacją, klasyfikacji tekstów. Ciekawym przykładem jest system promowany przez rząd Wielkiej Brytanii, stworzony w celu organizacji informacji publicznej w oparciu o ontologie. Inne przykłady można znaleźć w [7].

Mówiąc o zastosowaniach ontologii należy wspomnieć o projekcie Semantic Web, którego wielkim mentorem jest Tim Berners-Lee. Semantic Web jest próbą organizacji informacji w sieci internetowej w taki sposób, aby mogła być zrozumiana nie tylko przez ludzi, ale również przez komputery. Dzięki takiej organizacji możliwe będzie zwiększenie skuteczności wyszukiwania informacji.

Główną wadą stosowanej dotychczas organizacji informacji w internecie jest jej nastawienie na człowieka, a stąd [8]:

- zbyt duża lub za mała ilości zwracanych wyników wyszukiwania przez wyszukiwarki internetowe,
- zależność wyników wyszukiwania od użytego słownictwa,
- zbyt mały kontekst wyszukiwania (użytkownik zazwyczaj poszukuje informacji zawartych na wielu stronach, podczas gdy wyszukiwarki często poprzestają na znalezieniu jednej, najbardziej trafnej).

Semantic Web ma to zmienić dzięki przedstawieniu informacji w postaci zrozumiałej dla maszyn i ludzi oraz dzięki budowie baz wiedzy w oparciu o ontologie. Co więcej, takie podejście nie wymaga zmiany istniejącej infrastruktury informatycznej, ale może być wprowadzane stopniowo. Wśród jego zalet wymienia się również [8]:

- dobrą organizację wiedzy,
- zwiększenie możliwości silników wyszukiwarek poprzez tworzenie powiązań pomiędzy istniejącymi elementami bazy wiedzy,
- wyszukiwanie zorientowane na kontekst pytania, a nie na poszczególne słowa kluczowe,
- wykorzystanie wielu źródeł informacji przy generowaniu wyniku wyszukiwania.

Istnieje kilka formalnych metod tworzenia ontologii, języków jej zapisu oraz aplikacji narzędziowych [6]. Do najbardziej popularnych języków należą języki OWL i OWL2, powstałe na bazie RDF. Jedną z bardziej znanych aplikacji narzędziowych o otwartym kodzie jest Protégé [9] - aplikacja rozwijana przez SMI (ang. *Stanford Medical Informatics*). Oferuje ona bogate możliwości modelowania struktur wiedzy opartych o ontologie i pozwala na rozszerzanie funkcjonalności środowiska narzędziowego przez mechanizm wtyczek. Protégé oferuje dwa sposoby modelowania ontologii: *frame-based* oraz zgodne z Semantic Web.

6.3. Elementy języka OWL

Język OWL, oprócz dużych możliwości opisu otaczającego świata, posiada własności pozwalające przeprowadzać wnioskowanie. Poniżej przedstawiono elementy języka OWL, na których opierają swoją pracę silniki wnioskujące.

Język OWL został stworzony w 2001 r. przez W3C. Język ten jest elementem stosu technologii rekomendowanych przez W3C do opisu stron w Semantic Web. Stanowi on rozwinięcie języka RDF [10], a jego główne cechy zapożyczono z DAML+OIL. W pierwszej wersji OWL wyróżniono trzy warstwy, o rosnącej sile ekspresji:

- OWL Lite - dostarcza prostej klasyfikacji i cech, pozwala na tworzenie prostej taksonomii opartej na relacji *is-a*, umożliwia nakładanie więzów na relacje (dopuszczalne są więzy licznościowe o wartościach 0 i 1), nie pozwala na formułowanie ekstensjonalnych definicji pojęć.
- OWL DL - zwiększa efektywność obliczeniową oraz zapewnia rozstrzygalność podczas wysnuwania wniosków, posiada ekspresję logiki opisowej, umożliwia nakładanie kilku rodzajów więzów na relacje, nie pozwala definiować relacji zachodzących pomiędzy pojęciami (klasa nie może być traktowana intensjonalnie, ani jak indywiduum, ani jak relacja, podobnie relacja nie może być traktowana ani jak indywiduum ani jak klasa).
- OWL Full - zawiera pełne słownictwo OWL oraz oferuje pełne możliwości zapewniane przez RDF, w tej warstwie klasa może być traktowana równocześnie jako zbiór jednostek lub jednostka jako taka.

W 2009 r. OWL Working Group zakończyło pracę nad specyfikacją rewizji języka OWL [11]. Język ten nazwano OWL2. Ponieważ język ten nie różni się sposobem reprezentacji danych, jest on w pełni zgodny ze swoim poprzednikiem. Do nowej wersji dodano: klucze, łańcuchy własności, bogatsze typy danych i zakresów, asymetryczne, zwrotne i rozłączne własności, szersze możliwości umieszczania notatek. Dodatkowo, poprzez usunięcie części ograniczeń związanych z OWL DL, poszerzono możliwości silników wnioskujących. OWL 2 definiuje trzy nowe profile, które mogą zostać wykorzystane w zależności od potrzeb użytkownika [12]:

- **OWL 2 EL** - profil przeznaczony dla ontologii o dużej liczbie klas i własności. Dzięki zastosowaniu wydajnych i łatwo skalowalnych algorytmów podstawowe wnioskowanie może zostać przeprowadzone ze złożonością wielomianową;
- **OWL 2 QL** - profil nastawiony na ontologie o dużej liczbie instancji, gdzie wnioskowanie w dużej mierze oparte jest o zapytania. Mimo, że modele klas UML i diagramy ER są wspierane, to potencjał reprezentacji został w dużej mierze ograniczony;
- **OWL 2 RL** - profil zapewniający największą swobodę reprezentacji wiedzy, pozwalający na przeprowadzanie wnioskowania z wykorzystaniem reguł.

Klasy Podstawowym elementem ontologii w języku OWL jest klasa. Struktura klas jest punktem wyjściowym do definiowania ontologii. Jednym z elementarnych sposobów wnioskowania jest wnioskowanie oparte na klasach i dziedziczeniu. Wszystkie klasy ontologii są podklasami klasy *Thing*. Własność dziedziczenia klas jest przechodnie, tzn. jeżeli klasa A jest podklasą klasy B, a B jest podklasą klasy C, to A jest podklasą klasy C.

Instancje Instancje są obiektami należącymi do danej klasy. Deklaracja instancji odbywa się poprzez przypisanie jej do konkretnej klasy. Granica pomiędzy klasami i instancjami czasami może być zatarta, szczególnie jeżeli dana klasa może być instancją innej klasy. Problem pojawia się także na poziomie podklas, kiedy stosunkowo łatwo pomylić listę instancji z listą podklas reprezentujących grupę obiektów. Zdarza się również, że konieczne jest traktowanie obiektów ontologii w dwojaki sposób: jako klasy i instancje. Takie podejście możliwe jest w wersji OWL FULL, co jest jedną z istotniejszych różnic przy porównaniu z wersją OWL DL.

Właściwości Oprócz taksonomii ontologie umożliwiają opisywanie obiektów poprzez nadawanie im właściwości. Podstawowym zadaniem właściwości jest łączenie klas (instancji) pomiędzy sobą lub łączenie klas (instancji) z elementami języka RDF lub typami XML. Definicja właściwości składa się z deklaracji domeny i zakresu. Pierwsza z nich definiuje klasę posiadającą daną własność, druga - klasę, z którą właściwość łączy klasę domeny. Właściwości mogą tworzyć hierarchię - właściwości niższych poziomów jednoznacznie implikują właściwości ponad nimi. Zgodnie z założeniem właściwości pozwalają na budowanie relacji między klasami i wnioskowanie na ich podstawie. Dodatkowo wykorzystując

właściwości można budować nowe klasy spełniające określone ograniczenia (restrykcje).

OWL pozwala na definiowanie typów właściwości. Poniżej przedstawiono listę typów oraz ich własności (zakładając P - właściwość, x,y,z - klasy opisywane daną właściwością):

- **właściwość przechodnia** (*TransitiveProperty*)
  Jeżeli $P(x,y)$ i $P(y,z)$ to $P(x,z)$
- **właściwość symetryczna** (*SymmetricProperty*)
  $P(x,y)$ wtedy i tylko wtedy, gdy $P(y,x)$
- **właściwość funkcyjna** (*FunctionalProperty*)
  Jeżeli $P(x,y)$ i $P(x,z)$ to $y = z$
- **inwersja** (*inverseOf*)
  Jeżeli $P1$ jest inwersją $P2$ to $P1(x,y)$ wtedy i tylko wtedy, gdy $P2(y,x)$
- **odwrócona właściwość funkcyjna** (*InverseFunctionalProperty*)
  Jeżeli $P(y,x)$ i $P(z,x)$ to $y = z$

Restrykcje Oprócz mechanizmów opisanych wcześniej OWL pozwala definiować ograniczenia (restrykcje). Poniżej przedstawiono trzy typy restrykcji:

- *allValuesFrom, someValuesFrom*
  Wymienione typy restrykcji narzucają konieczność posiadania instancji danej właściwości przez wszystkie instancje klasy opisanej tą właściwością (pierwszy przypadek) lub przez przynajmniej jedną (drugi przypadek).
- *cardinality*
  Kardynalność pozwala na dokładne określenie ilości instancji danej właściwości w klasie opisywanej przez nią. Dodatkowo istnieje możliwość definiowania minimalnej i maksymalnej liczy instancji, a także zbioru ograniczonego przez te dwie wartości.
- *hasValue*
  Ostatni typ restrykcji pozwala na definiowanie typu na podstawie jego własności. Klasa zawierająca właściwość typu *hasValue*, będzie jednocześnie klasą obiektu, którego właściwość przyjmuje wartość zdefiniowaną przez restrykcje *hasValue*.

Reguły Reguły są opisywane językiem SWRL i pozwalają poszerzyć język OWL dzięki możliwości tworzenia wyrażeń definiujących zależności między właściwościami ontologii, oraz ontologiami i predykatami z tej samej dziedziny [13]. Od strony budowy reguły są wyrażeniami typu: $poprzednik \Rightarrow konsekwencja$. Oznacza to, że jeżeli *poprzednik* jest prawdziwy to *konsekwencja* też jest prawdziwa. Dodatkowo zakłada się, że pusty *poprzednik* jest wyrażeniem zawsze prawdziwym, natomiast pusta *konsekwencja* jest wyrażeniem fałszywym [14]. Każdy element składowy reguły może składać się z podelementów zwanych atomami.

Ponieważ język SWRL jest niezależny od OWL, wnioskowanie z wykorzystaniem obu tych narzędzi stanowi pewnego rodzaju wyzwanie [13]. Protégé po-

siada odrębny edytor przeznaczony wyłącznie do obsługi języka SWRL. Wiele wbudowanych w tę aplikację mechanizmów wnioskowania wspiera reguły.

6.4. Wnioskowanie

Wnioskowanie z ontologii odbywa się za pomocą tzw. silników wnioskowania (ang. *reasoners*) [15]. Na rynku dostępnych jest dużo narzędzi tego typu, a wybór musi być dostosowany do potrzeb programisty. Część silników wnioskowania dostępnych jest na płatnej licencji. Są to między innymi: *Bossam, DLog, OntoBroker, OWLIM, RacerPro, TopSPIN, SHER*. Znacznie większą rodzinę silników wnioskowania stanowią silniki dostępne na darmowej licencji. Dzięki temu można uzyskać dokładniejsze informacje na temat sposobu ich działania i wykorzystywanych przez nie algorytmów. Przykładowymi silnikami wnioskowania dostępnymi na darmowej licencji są: *OpenCyc, FaCT++, Hoolet, KAON2, HermiT*.

Silniki wykorzystują różne algorytmy wnioskowania. Ich zestawienie przedstawiono w tabeli 6.1.

Tab. 6.1: Porównanie algorytmów wnioskowania w poszczególnych silnikach.

silnik wnioskowania	**algorytm**
OpenCYC	First-order with high-order extensions
FaCT++	Tableau
Hoolet	First-order prover
KAON2	Resolution & Datalog
HermiT	Hypertableau

6.4.1. Algorytmy

Logika predykatów pierwszego rzędu Pierwszy z omawianych algorytmów jest wykorzystywany przez silnik *OpenCYC* [16]. Bazuje on na logika predykatów pierwszego rzędu [17]. Ta logika zajmuje się rachunkiem zdań oraz predykatami i kwantyfikatorami. Predykaty są funkcjami, które zwracają wartości *prawda* lub *fałsz*. Jeśli rozważymy zdanie:

Ziemia jest planetą

to predykatem nazywamy wyrażenie:

Planeta(Ziemia)

Logika pierwszego rzędu pozwala wnioskować na podstawie własności obiektów, wykorzystując zmienne. Jeśli założymy, że *Planeta*(x) oznacza, że zmienna x jest planetą, a *CialoNiebieskie*(x) oznacza, że zmienna z jest ciałem niebieskim, to wyrażenie:

$$CialoNiebieskie(x) \rightarrow Planeta(x)$$

oznacza, że jeśli x jest ciałem niebieskim, to jest także planetą. Prawdziwość takiego wyrażenia jest zależna od tego, co podstawimy pod zmienną x i od interpretacji predykatów. Dowiedzenie prawdziwości tego wyrażenia wymaga użycia kwantyfikatorów. Wtedy wyrażenie logiki pierwszego rzędu wygląda następująco:

$$\forall x(CialoNiebieskie(x) \rightarrow Planeta(x))$$

i oznacza, że niezależnie od podstawionego x można powiedzieć, że jeśli x jest ciałem niebieskim, to jest także planetą. Wykorzystany jest tutaj kwantyfikator $\forall$, który oznacza, że twierdzenie jest prawdziwe dla wszystkich zmiennych podstawionych pod x. Aby pokazać, że wyrażenie nie jest prawdziwe, należy pokazać, że istnieją ciała niebieskie, które nie są planetami. Wtedy otrzymujemy wyrażenie:

$$\exists x(CialoNiebieskie(x) \wedge \neg Planeta(x))$$

Predykaty *CialoNiebieskie*() i *Planeta*() przyjmują tylko po jednym parametrze. *Logika predykatów pierwszego rzędu* dopuszcza użycie predykatów, które przyjmują więcej niż jeden parametr. Jako przykład rozpatrzmy zdanie: *Jest człowiek, który codziennie chodzi do sklepu*, które może być zapisane w następujący sposób:

$$\exists x(Czlowiek(x) \wedge \forall y(Dzien(y) \rightarrow ChodziDoSklepu(x, y)))$$

Predykat $Czlowiek(x)$ oznacza, że zmienna x jest człowiekiem, $Dzien(y)$ oznacza dowolny dzień w roku, natomiast $ChodziDoSklepu(x, y)$ oznacza, że człowiek x chodzi do sklepu w dzień y.

Ważnym zagadnieniem w logice predykatów pierwszego rzędu jest jej formalna składnia. W poprawnym wyrażeniu mogą znajdować się terminy określające obiekty i formuły, które reprezentują predykaty i mogą być prawdziwe lub fałszywe. W logice predykatów pierwszego rzędu terminy i formuły są zapisywane jako ciągi znaków. W składni logiki można wyróżnić symbole logiczne, których znaczenie jest zawsze takie samo. Takimi symbolami są:

- kwantyfikatory: $\forall, \exists$,
- spójniki logiczne: $\wedge$ (koniunkcja), $\vee$ (alternatywa), $\Rightarrow$ (implikacja), $\Leftrightarrow$ (równoważność), $\neg$ (negacja),
- nawiasy, klamry, znaki przystankowe,
- zmienne oznaczane małymi literami (np. x, y, z),
- znak równości.

Terminy mogą być zapisywane jako zmienne (np. x, y...) albo funkcje (wyrażenie $f(t_1, t_2, ..., t_n)$, gdzie t_i jest terminem), natomiast formuły mogą być tworzone zgodnie z zasadami:

- symbol predykatu ($P(t)$),
- równość ($t_1 = t_2$),
- negacja ($\neg t$),
- spójniki ($t_1 \Leftrightarrow t_2$),

- kwantyfikatory $(\forall\, t)$.

W logice predykatów pierwszego rzędu jest określona kolejność wykonywanych operacji:

1. negacje $(\neg)$,
2. koniunkcje i alternatywy $(\wedge, \vee)$,
3. kwantyfikatory $(\forall, \exists)$,
4. implikacje $(\Rightarrow)$.

Stosując powyższe zasady formułę

$$(\neg \forall x P(x) \Rightarrow \exists x \neg P(x))$$

można zapisać w formie z nawiasami pozwalającymi zrozumieć kolejności wykonywanych operacji:

$$(\neg[\forall x P(x)]) \Rightarrow \exists x[\neg P(x)].$$

Tableau Algorytm wnioskowania *Tableau* [18] jest stosowany w silniku *Fact++* [19]. Algorytm *Tableau* jest systemem automatycznego dowodzenia twierdzeń, którego działanie polega na dowodzeniu przez zaprzeczenie (tzn. aby dowieść prawdziwość twierdzenia algorytm zaprzecza tezę i dowodzi jej sprzeczności). Algorytm jest nazywany także *drzewem prawdy*, gdyż na początku w korzeniu umieszczana jest formuła, której sprzeczność chcemy udowodnić. Następnie zgodnie z zasadami logiki, tworzone jest drzewo:

- jeśli w jednej gałęzi jest wyrażenie $x \wedge y$ to można w tej samej gałęzi umieścić x, a po nim y,
- jeśli w jednej gałęzi jest wyrażenie $x \vee y$, to należy wstawić rozgałęzienie na x i y,
- jeśli w gałęzi jest wyrażenie $\neg\neg x$ to na końcu gałęzi można dodać x,

Celem takiego dowodzenia twierdzeń jest zamknięcie drzewa prawdy. Dzieje się tak, jeśli wszystkie gałęzie drzewa są zamknięte, co oznacza, że w każdej gałęzi muszą znajdować się sprzeczne wyrażenia (np. $\neg x$ oraz x). Zamknięcie drzewa prawdy oznacza sprzeczność formuły umieszczonej w korzeniu drzewa, a to z kolei dowodzi prawdziwości postawionej tezy.

Algorytm Tableau ma także zastosowanie w logice predykatów pierwszego rzędu (która jest stosowana we wnioskowaniu z ontologii). Dowodzenie twierdzeń jest wtedy analogiczne jak we wcześniejszym przypadku, jednakże z tą różnicą, że oprócz wyrażeń złożonych wyłącznie ze spójników logicznych w drzewie prawdy są umieszczane wyrażenia zawierające dodatkowo kwantyfikatory i predykaty.

Odmianą omawianego algorytmu jest *hyper tableau* [20], zaimplementowany w silniku *HermiT* [21]. Działanie tego algorytmu jest identyczne jak w przypadku poprzedniego, jednak tutaj dodano funkcjonalność z algorytmu *hyper resolution*, która w pierwszym kroku algorytmu rozwiązuje wszystkie negatywne literały klauzul [22]. Znacznie przyspiesza to działanie algorytmu i minimalizuje obliczenia potrzebne do wypracowania wyniku.

Różnice pomiędzy silnikami wnioskowania

Silniki wnioskowania mają różne zastosowania, a ich wybór należy wyłącznie do użytkownika i jego wymagań. W celu zobrazowania różnic pomiędzy różnymi silnikami zredagowano tabelę 6.2.

Tab. 6.2: Porównanie silników wnioskowania.

Silnik	OpenCYC	FaCT++	HermiT
Algorytm	First-order with high-order extensions	Tableau	HyperTableau
Wsparcie języka reguł	Tak (wbudowany język reguł)	Nie	Tak (SWRL)
Plugin Protégé	Nie	Tak	Tak
Licencja	Darmowa/closed-source	Open-Source	Open-Source

Oprócz licencji i algorytmów silniki wnioskowania różnią się wsparciem języka reguł. OpenCYC posiada wbudowany język reguł, przez co reguły w nim pisane nie mogą być eksportowane do innych silników. FaCT++ natomiast nie wspiera żadnego języka reguł, co stanowi ograniczenie w używaniu tego silnika. Warto także dodać, że wyniki otrzymywane przy wykorzystaniu różnych silników niczym się nie różnią.

6.5. Przykładowa ontologia

Niniejszy podrozdział posłuży opisowi przykładowej ontologii, utworzonej przez autorów w celach demonstracyjnych. Do jej stworzenia wykorzystano program Protégé (dostępny na stronie `http://protege.stanford.edu/`), z przyjaznym dla użytkownika środowiskiem pracy z ontologiami. Utworzona ontologia umożliwi zaprezentowanie wnioskowania przy użyciu wbudowanych w programie Protégé silników Fact++ oraz HermiT (OpenCYC nie można używać jako pluginu, możliwe jest jednak uruchomienie osobnej aplikacji wykorzystującej ten silnik wnioskowania).

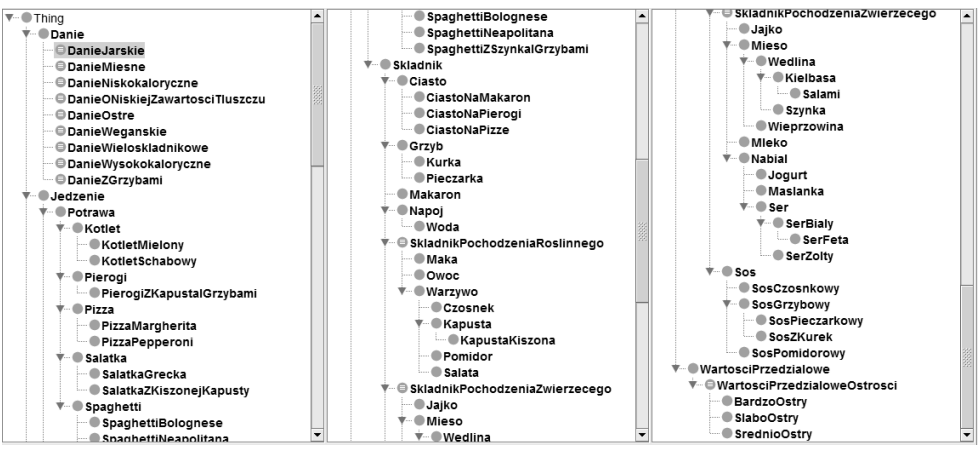
6.5.1. Opis zagadnienia

Inspiracją do opracowania opisanego dalej rozwiązania był artykuł [23], którego autorzy postanowili umożliwić dobieranie odpowiednich produktów żywieniowych dla diabetyków wykorzystując w tym celu ontologie. Podążając za tym wzorem postanowiono stworzyć prosty model, w którym na podstawie bazy składników i produktów można definiować dania spełniające odpowiednie warunki, np. dania niskokaloryczne, dania bezmięsne, dania rybne. Taka ontologia pozwoliłaby użytkownikowi (przy założeniu odpowiednio dużej i szczegółowej bazy potraw i składników) komponować odpowiednie posiłki, a tym samym

stanowiłaby pewne odwzorowanie ludzkiego spojrzenia na kategorie jedzenia. Dodatkowo, odpowiednio rozwinięta ontologia mogłaby niewątpliwie zostać wykorzystana w przyszłości np. do nauki nawyków żywieniowych człowieka przez robotycznych kucharzy .

6.5.2. Przedstawienie klas i właściwości

Na rysunku 6.1 przedstawiono strukturę klas utworzonej ontologii. Główną klasą nadrzędną jest klasa *Thing* (rzecz), co jest typowe dla każdej ontologii. Następnie wyróżnione są trzy podklasy: *Danie*, *Jedzenie* oraz *WartościPrzedziałowe*. Klasa *WartosciPrzedzialowe* służy wyłącznie wyszczególnieniu różnych poziomów ostrości dla potraw. Innymi słowy, oceniając ostrość potrawy za pomocą właściwości *jestOstry* należy użyć jednej z wartości tej klasy: *BardzoOstry*, *SrednioOstry*, *SlaboOstry*. W związku z faktem, iż te kategorie ostrości są rozłączne, potrawa nie może być jednocześnie bardzo i słabo ostra, co jest naturalne w świecie rzeczywistym. *Potrawa* jest podklasą klasy *Jedzenie*, która może zawierać w sobie wszystkie produkty żywieniowe. Potrawy składają się ze składników, czyli elementów klasy *Składnik*, co określane jest przy pomocy właściwości *zawieraSkladnik* lub *zawieraBezposredniSkladnik*, która jest właściwością pochodną tej pierwszej. Właściwość ta służy odróżnieniu tych składników, które potrawa zawiera bezpośrednio, np. ciasto w pierogach, od tych niebezpośrednich, np. mąka, będąca składową ciasta na pierogi. Właściwość *zawieraSkladnik* jest tranzytywna, więc jeżeli pizza zawiera ciasto, a ono z kolei zawiera mąkę, to można powiedzieć, że pizza zawiera mąkę (niebezpośrednio). Podział jedzenia na składniki i gotowe potrawy nie zawsze jest jednoznaczny, gdyż przykładowo ziemniaki mogą być zarówna potrawą, jak i składnikiem. Jednakże w związku z faktem, iż praktycznie niemożliwe jest stworzenie ontologii dotyczącej jedzenia, której nie dałoby się w żaden sposób podważyć, zaproponowany podział wydaje



Rys. 6.1: Struktura klas zbudowanej ontologii.

się odpowiedni. Tym bardziej, iż ontologia zazwyczaj powinna mieć charakter ogólny. Klasa *Danie* zawiera spis wszystkich potraw (podklas i instancji klasy *Potrawa*), które spełniają odpowiednie warunki i mogą być spożywane jako posiłek. Każda podklasa tej klasy jest opisana warunkami koniecznymi i wystarczającymi, które pozwalają na przyporządkowanie jej odpowiednich potraw z klasy *Potrawa*. Dzięki temu umożliwia ona wybór tych potraw, które mają odpowiednie cechy, np. zawartość tłuszczu w 100 g.

Jak już wspomniano ontologia wykorzystuje następujące właściwości: *jestOstry*, *zawieraSkladnik* oraz *zawieraBezposredniSkladnik*. Dodatkowo występują w niej właściwości odwrotne: *jestSkladnikiem* oraz *jestBezposrednimSkladnikiem* oraz właściwości danych: *zawieraKilokalorie*, *zawieraBialko*, *zawieraTluszcz* i *zawieraWeglowodany*. Każda z właściwości danych służy do określania, ile wartości odżywczych w 100 g zawiera dana klasa potraw lub instancja (czyli konkretna potrawa, np. hamburger w sieci fast foodów). Oczywiście rozszerzenie listy właściwości omawianej ontologii jest możliwe, jeśli tylko wystąpiłaby taka potrzeba.

6.5.3. Przykłady wnioskowania

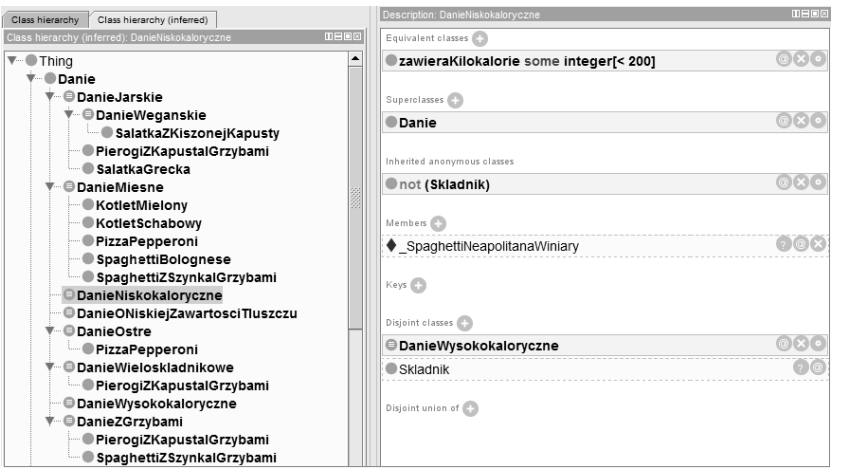
Jak już wspomniano, wnioskowanie w opisywanej ontologii będzie zachodziło głównie w stosunku do elementów klasy *Danie*, gdyż to w tym miejscu oczekiwane są logiczne informacje na temat jedzenia. Jednakże wnioskowanie może być przydatne, a nawet konieczne, w każdej części ontologii, gdyż może posłużyć np. do uproszczenia jej struktury. Widać to na przykładzie klas *SkladnikPochodzeniaRoslinnego* oraz *SkladnikPochodzeniaZwierzecego*. Wybór takich klas składowych jest uzasadniony, gdyż większość składników jedzenia spożywanego przez człowieka można zaliczyć do tych kategorii. Gdyby chcieć bezpośrednio zdefiniować elementy obu tych klas, co występuje często w przypadku składników złożonych, okazałoby się, że konieczne jest umieszczenie danego składnika w obu kategoriach. Oczywiście jest to wykonalne, ale bardziej odpowiednim sposobem jest określenie warunków wystarczających na bycie składnikiem danego typu. W tym przypadku warunki te można zdefiniować za pomocą właściwości *zawieraSkladnik*. Jeżeli bowiem dana klasa jest elementem klasy *Skladnik* oraz zawiera dowolny składnik, który jest pochodzenia zwierzęcego, to także może być zdefiniowana jako taki składnik. Podobnie jest w przypadku składników pochodzenia roślinnego. Warunek ten w programie Protégé można zapisać następująco: *Skladnik and ((zawieraSkladnik some SkladnikPochodzeniaZwierzecego) or (zawieraSkladnik only SkladnikPochodzeniaZwierzecego))*. Po utworzeniu takiego warunku i uruchomieniu silnika wnioskowania klasa *Makaron*, która zawiera *CiastoNaMakaron*, a ono z kolei zawiera zarówno mąkę, jak i jajka, została przyporządkowana do klasy *SkladnikPochodzeniaRoslinnego* oraz klasy *SkladnikPochodzeniaZwierzecego*. Na tym przykładzie widać też tranzytywność właściwości *zawieraSkladnik*, dzięki której silnik wnioskowania potrafił określić, iż Makaron zawiera mąkę i jajka, chociaż nie było to określone bezpośrednio.

Na rysunku 6.2 przedstawiono strukturę klasy *Danie* otrzymaną po uruchomieniu silnika wnioskowania Fact++. Jak widać, różni się ona od struktury pierwotnej widocznej na rysunku 6.1. Przede wszystkim można zauważyć, iż silnik

poprawnie określił danie wegańskie (potrawa nie zawierająca składników pochodzenia zwierzęcego i zawierająca wyłącznie składniki pochodzenia roślinnego lub grzyby) jako danie jarskie (potrawa nie zawierająca mięsa lub zawierająca wyłącznie składnik pochodzenia roślinnego lub grzyby). Dzięki odpowiednio zapisanym warunkom dla każdej z tych klas, *SalatkaZKiszonejKapusty* zawierająca wyłącznie warzywa, może być przyporządkowana do potraw wegańskich, a *PierogiZKapustaIGrzybami* oraz *SalatkaGrecka*, zawierające odpowiednio jajka i ser, są tylko daniami jarskimi (bezmięsnymi). Jak widać takie wnioskowanie może być np. użyteczne, gdy użytkownik systemu wykorzystującego ontologię chce przygotować wegetariański posiłek. Innymi słowy ontologia (oczywiście w odpowiednio rozwiniętej postaci) mogłaby posłużyć elektronicznemu doradcy w dobieraniu odpowiednich dań dla klientów restauracji na podstawie ich preferencji.

Pozostałe podklasy klasy *Danie* także uległy zmianom po uruchomieniu silnika wnioskowania. *DanieMiesne* zawiera teraz wszystkie klasy, które w warunkach koniecznych miały zdefiniowaną formułę: *zawieraSkladnik some Mieso*. Ponadto widać ponownie, iż wnioskowanie nie ogranicza się tylko do bezpośrednich reguł, gdyż *PizzaPepperoni* zawierająca bezpośrednio salami i ser żółty, także została zaliczona do potraw mięsnych. Stało się tak dlatego, że salami jest zdefiniowane jako kiełbasa, która z kolei jest wędliną, a wędlina jest oczywiście mięsem. Podobnie instancja klasy (konkretny przykład potrawy) *PizzaPepperoni* o nazwie *PizzaPepperoniZPizzeriZaRogiem* została zakwalifikowana do dań mięsnych ze względu na klasę, do której przynależy. Klasa *DanieMiesne* może służyć do wyboru potraw, dla osób lubiących spożywać różnego rodzaju mięsa. Podobnie klasa *DanieZGrzybami* może być używana do wyboru potrawy, którą można przyrządzić z nazbieranych akurat w lesie grzybów.

Klasy *DanieWysokokaloryczne* i *DanieNiskokaloryczne* zawierają elementy ontologii o zawartości kilokalorii odpowiednio powyżej 300 oraz poniżej 200



Rys. 6.2: Wyniki wnioskowania dla klasy Danie.

w 100 gramach potrawy. Nie mają one żadnych podklas, gdyż warunek ten dotyczy właściwości danych, czyli stosowany jest wyłącznie do instancji. Zawierają więc one instancje, które przy utworzeniu miały zdefiniowaną odpowiednią liczbę kilokalorii. Podobnie sytuacja wygląda w przypadku klasy *DanieONiskiejZawartościTluszczu*. Można się domyśleć, iż opisane klasy mogą być bardzo użyteczne przy układaniu diety, która musi zawierać odpowiednią ilość składników odżywczych.

W przypadku, gdy użytkownik chciałby przyrządzić skomplikowaną potrawę, użyteczną może być klasa *DanieWieloskladnikowe*, która zawiera potrawy o ilości bezpośrednich składników przekraczającej trzy. Oczywiście można zmodyfikować tę klasę tak, żeby zawierała potrawy o małej lub dokładnie określonej ilości składników.

Wnioskowanie na podstawie ontologii jest, jak widać, procesem, który może być niezwykle użyteczny w przypadku uczenia maszyn, urządzeń czy programów ludzkiego spojrzenia na świat. Dodatkowo ontologie mogą być środkiem w tworzeniu różnych systemów eksperckich, wspomagających ludzi w podejmowaniu decyzji, zawierających specjalistyczne informacje. Przedstawiona ontologia po odpowiednim rozbudowaniu mogłaby niewątpliwie być użyteczna w tego typu systemach. Świadczy to o dużym potencjale ontologii i wielu możliwościach ich wykorzystania.

Literatura

[1] W. Gliński. *Od informacji naukowej do technologii społeczeństwa wiedzy*, chapter Ontologie. Próba uporządkowania terminologicznego chaosu, pages 163–175. SBP, Warszawa, 2005.

[2] B. Smith and Ch. Welty. FOIS introduction: Ontology—towards a new synthesis. In *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, FOIS '01, pages 3–9, New York, NY, USA, 2001. ACM. Conference Chair-Smith, Barry and Conference Chair-Welty, Christopher.

[3] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[4] W.N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Universiteit Twente, Enschede, September 1997.

[5] R. Studer, V.R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25:161 – 197, 1998.

[6] O. Corcho, M. Fernández-López, and A.G. Pérez. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & Knowledge Engineering*, 46(1):41 – 64, 2003.

[7] C. Brewster and K. O'Hara. Knowledge representation with ontologies: the present and future. *Intelligent Systems, IEEE*, 19(1):72 – 81, Jan - Feb 2004.

[8] G. Antoniou and F.V. Harmelen. *A semantic web primer*. Cooperative information systems. MIT Press, 2004.

[9] Protégé Overview. `http://protege.stanford.edu`.

[10] D.L. Mcguinness and F. van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, February 2004.

[11] OWL Working Group. OWL 2 Web Ontology Language. Document Overview. Technical report, W3C, 2009.

[12] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language – Profiles. Technical report, W3C, 2009.

[13] Ch. Golbreich, O. Dameron, O. Bierlaire, and B. Gibaud. What reasoning support for Ontology and Rules? The brain anatomy case study. In *Workshop on OWL Experiences and Directions*, 2005.

[14] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/SWRL`, 2004.

[15] Semantic reasoners. `http://en.wikipedia.org/wiki/Semantic_reasoner`.

[16] OpenCyc homepage. `http://opencyc.org/`.

[17] First–order logic. `http://en.wikipedia.org/wiki/First-order_logic`.

[18] R. Möller and V. Haarslev. Tableau-based reasoning. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 509–528. Springer, 2009.

[19] FaCT++ homepage. `http://owl.man.ac.uk/factplusplus/`.

[20] B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *Artifical Intelligence Research*, 36:165 – 228, 2009.

[21] Hermit OWL Reasoner homepage. `http://hermit-reasoner.com/`.

[22] Jan van Eijck. Constrained hyper tableaux. In Laurent Fribourg, editor, *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2001.

[23] J. Cantais, D. Dominguez, V. Gigante, L. Laera, and V. Tamma. An example of food ontology for diabetes control. In *Proceedings of the International Semantic Web Conference 2005 workshop on Ontology Patterns for the Semantic Web*, 2005.

ROZDZIAŁ 7

ZWIĘKSZANIE JAKOŚCI DANYCH SENSORYCZNYCH

T. Mazurkiewicz, A. Pyka

7.1. Wstęp

Podczas przetwarzania danych w różnych zastosowaniach istnieje problem związany z ich niedokładnością. Występuje on w szczególności w aplikacjach wymagających wysokiej precyzji danych wejściowych, np. w nawigacji czy lokalizacji. Niezmiernie ważna jest też prawidłowa interpretacja danych uzyskanych z odczytów sensorycznych dla robotów mobilnych, których położenie na scenie jest funkcją wysoce zmienną w czasie.

W niniejszym rozdziale przedstawiono autorską metodę zwiększania jakości danych sensorycznych dla robota mobilnego klasy *Line-Follower*. Do klasy *Line-Follower* zaliczane są roboty typu (2,0), czyli roboty mobilne o dwóch sztywnych kołach napędowych, których prędkość można zmieniać oraz bez kół skrętnych. Ich zadaniem jest śledzenie szerokiej na 2cm czarnej linii, namalowanej bądź wyklejonej czarną taśmą na białej powierzchni podłoża. Prawidłowe odwzorowanie zadanej linią trasy przez robota zależy w znaczącym stopniu od odczytów sensorycznych [1, 2, 3, 4]. Sensorami dla robotów klasy *Line-Follower* są najczęściej zestawy refleksyjnych czujników optycznych, opartych o podczerwoną diodę IR LED oraz fototranzystor. Niestety, odczyt z sensorów bywa silnie zakłócany przez oświetlenie zewnętrzne, zawiera sporo składowej podczerwonej. Prezentowany algorytm pozwala na znaczącą poprawę jakości danych sensoryczych, a w efekcie znaczącą poprawę sposobu śledzenia zadanej ścieżki przez robota.

7.2. Zadanie zwiększania jakości danych

Rozwój różnorodnych technologii sensorycznych zwiększa ilość danych możliwych do wydobycia z otaczającego środowiska. Dostępne są dziesiątki sposobów akwizycji i przechowywania zdobytych informacji. Niestety, mnogość możliwości nie zawsze idzie w parze z jakością: dane mogą być niedokładne, niepełne lub po prostu błędne. Problem jest o tyle poważny, iż celem gromadzenia

większości danych jest ich wykorzystanie do podejmowania decyzji. Tym samym niska ich jakość może prowadzić do podejmowania decyzji nieoptymalnych, co rzutuje na ocenę jakości działania całego systemu.

Do głównych cech decydujące o jakości danych zalicza się: dokładność, powtarzalność, aktualność, przydatność, spójność, kompletność, wiarygodność. Aby chronić system (informatyczny, przemysłowy, robotyczny) przed danymi niskiej jakości stosuje się działania mające na celu ich poprawą lub przynajmniej uniknięcie ich pogorszenia. Biorąc pod uwagę technikę działania metody stosowane w tym celu można podzielić na następujące grupy: standaryzacja formatu danych, kontrola poprawności przepływu danych, filtracja danych, metody statystyczne poprawy jakości danych.

7.2.1. Standaryzacja formatu danych

Każda architektura, platforma czy język programowania oferuje do wykorzystania wiele struktur danych o zróżnicowanej dokładności i złożoności. Często można nimi opisać nawet nietrywialnych przypadki. Jednym z podstawowych sposób poprawy jakości danych jest zadbanie o ich dokładność, powtarzalność, aktualność i spójność poprzez wprowadzenie standardów dotyczących struktur danych, sposobu i dokładności ich reprezentacji. Przykładem dobrze obrazującym ten problem jest sposób przechowywania danych adresowych. Przez dane adresowe, w dużym uproszczeniu, można rozumieć nie tylko sam kod pocztowy lub kraj, ale również pozostałe fragmenty danych, jak: miasto, ulicę, numer domu, mieszkania. Adres może być przechowywanych w bazie danych na jednym polu typu tekstowego, ale może też zajmować osobne pola dla każdego fragmentu danych. Dane mogą być serializowane i zapisywane w zewnętrznych plikach, npplikach XML. Oprócz samego sposobu zapisu rozbieżności mogą dotyczyć rónież notacji kodu pocztowego (z myślnikiem lub bez) czy wpisywania numeru mieszkania (gdy brak, to pozostawić puste czy uzupełnić w określony sposób). Adres to rzecz, która może zmieniać się dość często, stąd warto przechowywać dodatkową informację o czasowym zakresie jego ważności.

W przykładzie robota klasy *Line-Follower* problemem był dobór właściwej struktury danych do przechowywania danych sensorycznych oraz ich numerycznego formatu wpływającego na dokładność i szybkość przetwarzania (obliczenia zmiennoprzecinkowe, długość słowa procesora). Również ważnym zagadnieniem był odpowiedni dobór wielkości bufora danych sensorycznych (przy zbyt dużym buforze pojawiało się niebezbieczeństwo podejmowania decyzje w oparciu o nieaktualne dane).

7.2.2. Kontrola poprawności przepływu danych

Częstym źródłem pojawiania się niepełnych czy błędnych danych w systemie jest źle dobrany nośnik danych lub sposób ich przepływu. Może to być źródłem całego spektrum potencjalnych błędów - od najbardziej prozaicznych (jak gubione pakiety podczas bezprzewodowego przesyłu danych), przez błędy ludzkie (npbłędne przepisanie danych z formularza papierowego do systemu elektro-

nicznego) aż po ograniczenia technologiczne i projektowe (npżbyt wolne przetwarzanie danych, przez co część danych oczekujących na przetworzenie jest nadpisywana przez nowe nadchodzące dane).

Możliwości kontroli poprawności przepływu danych jest równie dużo jak źródeł błędów. Pozwalają one na zachowanie kompletności i spójności. W opracowanym przykładzie najważniejszym działaniem z tego zakresu było stosowanie sum kontrolnych podczas przesyłu danych.

7.2.3. Filtracja danych

Opisane wyżej metody pozwalają zachować w dużej mierze spójność, kompletność, powtarzalność czy aktualność danych. Jednak w małym stopniu wpływają na przydatność gromadzonych danych czy na ich dokładność. Odpowiednia filtracja danych pozwala zwiększyć jakość gromadzonych danych właśnie w tych obszarach. W pierwszej kolejności można odrzucić te informacje, które na podstawie wiedzy apriorycznej uznać można za niepoprawne. Dodatkowo, można wydobyć z gąszczu informacji te, które są rzeczywiście potrzebne, odrzucając informacje zbędne i redundantne, które zwiększają koszt przetwarzania a nie wpływają na jego wynik.

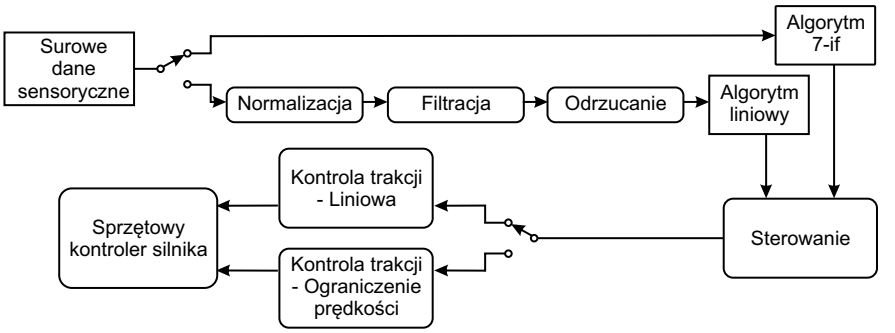
W przypadku robota typu *Line-Follower* filtracja odbywa się głównie na analogowym sygnale, mając na celu usunięcie z niego zakłóceń, a przez to zwiększenie dokładności i wiarygodności danych sensorycznych.

7.2.4. Metody statystyczne poprawy jakości danych

Najobszerniejszą ale i zarazem najbardziej podstawową i powszechną techniką poprawy jakości danych jest stosowanie metod statystycznych. Zaczynając od tych najbardziej podstawowych, jak stosowanie wielokrotnych pomiarów, ich uśrednianie i obliczania odchyleń standardowych, przez metody takie jak aproksymacja czy linearyzacja dla danych niepełnych, aż po metody bardziej zaawansowane, wchodzące w dziedzinę rachunku prawdopodobieństwa. Tego rodzaju techniki stosuje się przede wszystkim w celu zwiększenia wiarygodności, dokładności i powtarzalności danych. Wspomniana aproksymacja czy linearyzacja może pozwolić na zwiększenie spójności i kompletności posiadanych informacji poprzez oszacowanie wartości nieznanych czy przyszłych. Również aktualność i przydatność danych może zostać poprawiona dzięki działaniom z pogranicza statystyki i rachunku prawdopodobieństwa. Jak widać metody statystyczne w odróżnienie od wcześniej omawianych, mogą mieć duży wpływ na jakość danych w każdym z omawianych obszarów.

7.3. Implementacja zadania

W celu zilustrowania działania opisanych metod w praktyce, zaimplementowano algorytm sterowania na platformie mobilnej typu *Line-Follower*. Przepływ danych w robocie odbywa się według schematu z rysunku 7.1.



Rys. 7.1: Przepływ danych w robocie *Line-Follower*.

7.3.1. Mechanika robota

Elementem nośnym robota jest płyta z elektroniką. W przedniej części zostały wytrawione ścieżki, natomiast tylna część laminatu została odpowiednio docięta i stanowi mocowanie dla silników napędowych. Jako napęd wykorzystane zostały odpowiednio przerobione mikroserwa TowerPro MG90S. Są to serwomechanizmy z połową metalowych trybów, dzięki czemu są wystarczająco odporne na przeciążenia mechaniczne. Z serwomechanizmów została usunięta blokada pełnego obrotu oraz wyjęta została elektronika sterująca. W jej miejsce zostały wlutowane tranzystory MOSFET IRLZ44N, które mogą być sterowane bezpośrednio sygnałami TTL. Utracono wprawdzie możliwość obrotu kół w obu kierunkach, niemniej w algorytmie nie była planowana konieczność wstecznej jazdy robota. Sterowanie silnikami odbywa się poprzez 10-bitowy sygnał o zmiennej długości wypełnienia, PWM. Bezpośrednio na osie serwomechanizmów zamontowane zostały neoprenowe koła średnicy 45mm. Serwomechanizmy zostały przykręcone do ramy nośnej przy użyciu kawałka laminatu, który tworzy obejmę mocującą.

7.3.2. Elektronika robota

Sercem układu elektronicznego jest mikrokontroler Atmel AtMega8. Jest to ośmiobitowa jednostka w architekturze RISC, o pojemności pamięci flash 8kB oraz 1kB pamięci SRAM, zawierająca w swojej strukturze m.in. 8-kanałowy przetwornik analogowo-cyfrowy (w zastosowanej wersji DIP wyprowadzonych jest jedynie 6 kanałów), który jest wykorzystywany do pomiarów sygnałów z czujników linii. W roli czujników linii zastosowano refleksyjne sensory podczerwieni Vishay CNY70 w układzie dzielnika napięciowego. Otrzymywany z nich sygnał napięciowy z zakresu 0..5V, zależny od strumienia świetlnego padającego na detektor (fototranzystor), jest digitalizowany przez 10-bitowy przetwornik analogowo-cyfrowy [5, 6].

Białe tło bardzo dobrze odbija światło, natomiast czarne w znaczny stopniu je pochłania, zatem strumień światła padający na fotodetektor determinuje odcień podłoża, przy założeniu braku oświetlenia zewnętrznego. Niestety, w rzeczywistym przypadku, aby uniezależnić się od warunków środowiskowych, należy za-

stosować pomiar różnicowy. Wykonuje się serię pomiarów z załączonym oświetleniem dodatkowym (doświetlaczem) oraz bez oświetlenia dodatkowego. Bezwzględna różnica wyników jest proporcjonalna do strumienia świetlnego odbitego od badanej powierzchni a zatem niejako determinuje odcień podłoża (czarny/biały) [7, 8, 9].

7.3.3. Kodowanie algorytmu

Algorytmy przetwarzania danych sensorycznych

Zadanie kodowania algorytmu rozpoczęto od zapisania metod inicjalizacji przetwornika ADC oraz odczytu wartości napięć na poszczególnych kanałach przetwornika w pętli głównej programu:

```
MEASURES := 3 //liczbę pomiarów dla jednego kanału
                    // dla odczytu uśrednionego

for j = 1 to 5 do //odczytaj 5 kanałów dla LEDs OFF
   ADC_SETCHANNEL := j //przełącz kanał
   adcdata[j] := 0 //wyczyść bufor na wynik
   v := MEASURES //liczba pomiarów dla jednego kanału
   while v > 0
    v := v - 1
    ADC_START := true //rozpocznij konwersję
    while ADC_READY = true //czekaj, dopóki nie koniec
    endwhile
    adcdata[j] := adcdata[j] + ADC_RESULT //zapisz w tabeli
   endwhile
endfor

poweronAllLed() //włącz LED i czekaj na rozgrzanie
delay_us(100)

for j = 1 to 5 do //odczytaj 5 kanałów dla LEDs ON
   ADC_SETCHANNEL := j //przełącz kanał
   v := MEASURES //liczba pomiaró dla jednego kanału
   _adc := adcdata[j] //zapisz wcześniejsze rezultaty
   adcdata[j] := 0 //i wyczyść bufor wyników
   while v > 0
    v := v - 1
    ADC_START := true //rozpocznij konwersję
    while ADC_READY = true //czekaj, dopóki nie koniec
    endwhile
    adcdata[j] := adcdata[j] + ADC_RESULT //zapisz w tabeli
   endwhile
   //zapamiętaj różnicę odczytów jasne-ciemne
   adcdata[j] := adcdata[j] - _adc
   //uśrednij
   adcdata[j] := adcdata[j] /= MEASURES
   if adcdata[j] < 0 then
     adcdata[j] := 0
   endif
```

```
endfor

clearLed() //wyłącz LED i czekaj na zgaszenie
_delay_us(200)
```

Algorytm przetwarzania surowych danych uzyskanych w opisanym powyżej procesie dygitalizacji wartości wyjściowych analogowych czujników koloru linii składa się z trzech etapów – kalibracji, rozciągania (normalizacji) oraz odrzucania względem średniej. Dane kalibracyjne zapisywane są w nieulotnej pamięci EEPROM mikrokontrolera sterownika robota, przez co po ponownym załączeniu zasilania robota pamiętane są nastawy algorytmu.

Obsługa przekształcania danych w pętli głównej programu przedstawia się następująco (pominięto opis procedur zapisu danych kalibracyjnych):

```
//etap pierwszy - próba poprawy minimów i maksimów kalibracyjnych
nmax := 0 //wartość maksymalna w serii danych
nmin := MM_INF //wartość minimalna w serii danych
for j = 0 to 5 do
        if mins[j] > adcdata[j]
                mins[j] := adcdata[j]
        endif
        if maxs[j] < adcdata[j]
                maxs[j] := adcdata[j]
        endif
endfor

//etap drugi - normalizacja skalująca (rozciąganie)
prog := 388 //próg wyznaczony doświadczalnie
avg := 0 //średnia z wartości znormalizowanych
for j = 0 to 5 do
        normal[j] := (maxs[j] - adcdata[j]) * 1024 / maxs[j]
        if nmax < normal[j]
                nmax := normal[j]
        endif
        if nmin > normal[j]
                nmin = normal[j]
        endif
        avg := avg + normal[j]
endfor
avg := avg / 6 //obliczona średnia z wartości znormalizowanych

//etap trzeci - odrzucanie znormalizowanych danych,
//które są niższe niż średnia z wyników
rozrzut := nmax - nmin //rozrzut na normalizowanych danych
for j = 0 to 5 //wycinamy wartości niższe od średniej
        if normal[j] < avg
                normal[j] := 0
        endif
endfor
for j = 0 to 5 //rozciągamy liniowo do pełnego zakresu (0..1023)
        if normal[j] > 0
                normal[j] = normal[j] + (1024 - nmax)
```

```
        endif
endfor
```

Ostatecznym etapem przetwarzania wejściowych danych pomiarowych jest algorytm obliczający funkcję wyniku, którego pseudokod przedstawiono na listingu poniżej:

```
wynik := 0
c_top := 15
c_bot := 10
mx1 := 0
mx2 := -1
lastPosFactor := 15

if rozrzut > prog
  mx2 := -1
  lastSet := true
  for i := 0 to 2 do
    if normal[i] > normal[5-i] AND
      (normal[i] * c_bot > c_top * mx2 OR
      (normal[i] > mx2 AND lastSet))
      mx2 := normal[i]
      mx1 := i
      lastSet := true
    elseif normal[i] < normal[5-i] AND
          (normal[5-i] * c_bot > c_top * mx2 OR
          (normal[5-i] > mx2 AND lastSet))
          mx2 := normal[5-i]
          mx1 := 5-i
          lastSet := true
    else
        lastSet := false
    endif
  endfor
  wynik := 2048 * mx1 - 5120
  if mx1 > 0
    wynik := wynik - normal[mx1-1]
  endif
  if mx1 < 5
    wynik := wynik + normal[mx1+1]
  endif
  lastPos := ((lastPos * lastPosFactor) +
  wynik * (10000 - lastPosFactor)) / 10000
else
  if lastPos > 0
    wynik := 5120
  elseif
   wynik := -5120
  endif
endif
```

Algorytmy trakcyjne robota

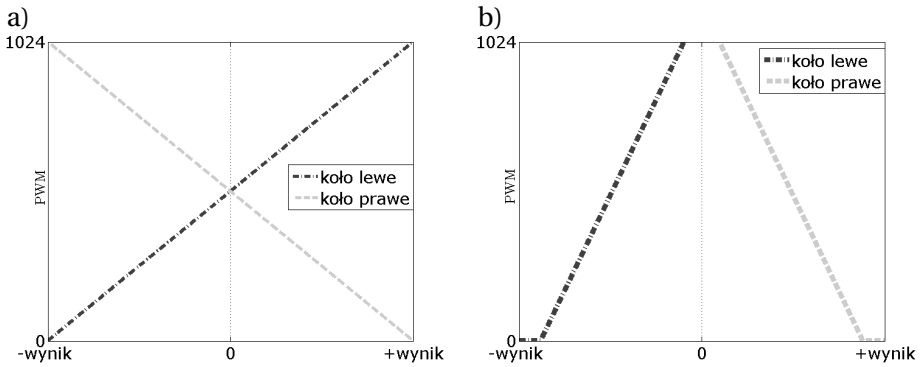
W celu przetestowania jakości trakcyjnej robota jeżdżącego w oparciu o czyste dane w stosunku do zaproponowanego algorytmu opisanego w podrozdziale 7.3.3, zaimplementowano tradycyjny algorytm binarny nazywany `7-if` od występujących w nim warunków `jeżeli`. Jego działanie opiera się o dzielenie wejściowej funkcji estymaty położenia linii na przedziały o zmiennej szerokości. Następnie, w zależności od przedziału, przyjmowane są konkretne z góry ustalone wartości sterowania podawanego na silniki. Wejściowa funkcja estymaty, której wartości przechowuje zmienna `wynik`, mogą być pobierane zarówno z czystych przeskalowanych odczytów z przetwornika ADC jak też z algorytmu opisanego w podrozdziale 7.3.3. Pozwala to na przeprowadzenie eksperymentów które udzielą odpowiedzi na pytanie o poprawę jakości trakcyjnych robota po zastosowaniu proponowanego algorytmu. Poniższy listing zawiera pseudokod algorytmu `7-if`.

```
if wynik = 0 //line at center point of robot
  setRightMotor(minimalSpeed + (desiredSpeed * 3) /8)
  setLeftMotor(minimalSpeed + (desiredSpeed * 3) /8)
  output := 0

elseif wynik > 0 //linia za dużo na prawo
       if wynik >= 2048
        if wynik >= 4096 //strefa 5
         setRightMotor(minimalSpeed)
         setLeftMotor(minimalSpeed + (desiredSpeed * 3) /5)
        else //wynik<4096 strefa 4
         setRightMotor(minimalSpeed)
         setLeftMotor(minimalSpeed + (desiredSpeed * 4) /5)
        endif
        else //wynik < 2048 strefa 3
         setRightMotor(minimalSpeed + (desiredSpeed * 1) /3)
         setLeftMotor(minimalSpeed + (desiredSpeed * 2) /3)
        endif
else //linia za dużo na lewo
      if wynik < -2048
       if wynik < -4096 strefa 0
        setRightMotor(minimalSpeed + (desiredSpeed * 3) /5)
        setLeftMotor(minimalSpeed)
       else //wynik >= -4096 strefa 1
        setRightMotor(minimalSpeed + (desiredSpeed * 4) /5)
        setLeftMotor(minimalSpeed)
       endif
      else //wynik >= -2048 strefa 2
        setRightMotor(minimalSpeed + (desiredSpeed * 2) /3);
        setLeftMotor(minimalSpeed + (desiredSpeed * 1) /3);
      endif
endif
```

Oprócz powyższego, w robocie zaimplementowano algorytmy PID oraz algorytm liniowy, działające w oparciu o funkcję estymaty pobieranej z proponowanego algorytmu opisanego w podrozdziale 7.3.3 (ich opis pominięto).

Do sterowania robotem niezbędny jest również niskopoziomowy algorytm trakcji silników, regulujący wypełnienie sygnału PWM kluczującego stopnie mocy zasilające silniki napędowe robota. Przedstawiono dwie propozycje – algorytm liniowy oraz algorytm z limitowaniem prędkości silników. Algorytm liniowy, jak sama nazwa wskazuje, zmienia liniowo wypełnienie sygnału PWM kluczującego stopnie mocy w zależności od funkcji wejściowej, co przedstawiono na rysunku 7.2a. Algorytm z limitowaniem prędkości silników ogranicza pośrednio prędkość jednego z kół poprzez zmianę wypełnienia sygnału PWM, gdy funkcja wejściowa znajduje się w odpowiedniej połówce charakterystyki przedstawionej na rysunku 7.2b.



Rys. 7.2: Algorytmy trakcyjne robota: a) algorytm liniowy, b) algorytm z limitowaniem prędkości.

7.3.4. Programy narzędziowe

W celu łatwiejszej obsługi robota oraz wizualizacji stanu sensorycznego i trakcyjnego robota napisana została konsola diagnostyczna, której screen przedstawiono na rysunku 7.3.

Aby możliwy był rozwój algorytmu zaproponowanego w podrozdziale 7.3.3, stworzono oprogramowanie wizualizujące poszczególne fazy działania algorytmu. Screen z działania programu przedstawiono na rysunku 7.4, na którym prezentowana jest reakcja algorytmu na światło zewnętrzne – zbliżenie piętnastowatowej świetlówki na odległość 5mm od czujników wpływało nie więcej niż 3% na wynik algorytmu.

7.4. Testy

W przypadku problemu zaprezentowanego w niniejszym rozdziale, trudno znaleźć czy zgromadzić dane referencyjne mogące posłużyć do obiektywnej

oceny zaproponowanego rozwiązania. Problematyczne jest również wybranie wskaźników bezpośrednio opisujących optymalizowane zagadnienie. Z tego powodu zdecydowano się na zastosowanie wskaźników pośrednich, przedstawiających problem z punktu widzenia celu działania robota typu *Line-Follower* oraz optymalizacji wyniku sterowania.

7.4.1. Wskaźniki

Wybrano dwa wskaźniki do pośredniej oceny rezultatów pracy algorytmu: czas przejazdu oraz łączną rozbieżność trasy przejazdu od trasy wyznaczonej.

Pierwsze kryterium oceny jest naturalnym wyborem z punktu widzenia regulaminu zawodów robotów typu *Line-Follower*. Ich celem jest osiągnięcie jak najlepszego czasu, więc oczywistym celem (a zarazem kryterium oceny) poprawy jakości działania algorytmu jest poprawa czasu przejazdu.

Drugie kryterium jest ważne z punktu widzenia jakości sterowania, gdzie najważniejsze jest jak najwierniejsze odwzorowanie trasy przejazdu. Rozbieżność między trasą przejazdu, a trasą zadania będzie liczona z wykorzystaniem zmiennej `wynik`, która w zaproponowanym algorytmie reprezentuje odległość środka robota od zadanej linii.

W ramach testów zdecydowano się na analizę porównawczą wyników przejazdów robota bez algorytmu poprawy jakości danych oraz z nim.

7.4.2. Warunki przeprowadzenia testów

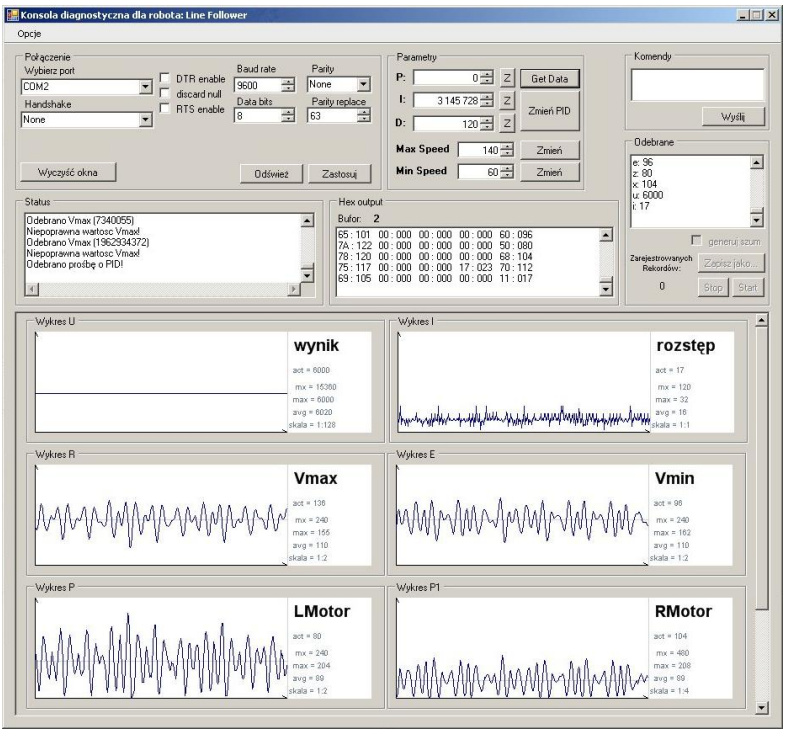
Do przeprowadzenia testów przygotowane zostało stanowisko testowe przedstawione na zdjęciach 7.5 oraz 7.6. Stanowisko składało się z:

- zestawu 3 tras testowych,
- zestawu barier podczerwieni start/meta wraz ze sterownikiem,
- komputera rejestrującego i wyświetlającego czas przejazdu,
- kamery cyfrowej o osi optycznej prostopadłej do powierzchni torów.

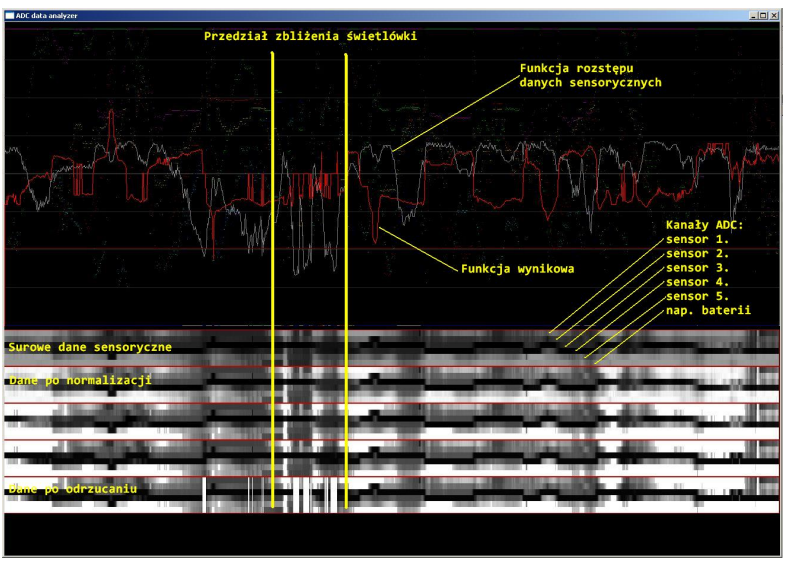
7.4.3. Trasy

Trasy wykorzystane w eksperymencie zostały zaprojektowane tak, aby przetestować możliwości robota w obecności różnych utrudnień. Na zdjęciu 7.7 przedstawiony został widok z góry na poszczególne trasy.

Pierwsza z nich bada przede wszystkim skuteczność wejścia w zakręt przy dużej prędkości, wyjścia z zakrętu na prostą oraz zdolność osiągania maksymalnych osiągów na prostej. Długość trasy wynosi 1430mm. Na drugiej z tras zrezygnowano z prostych, a skupiono się na zdolności na efektywnego pokonywania ostrych zakrętów po łuku. Długość trasy wynosi 1650mm. Trzecia trasa podobnie jak druga, nie posiada długich prostych, ale znajduje się na niej duże zagęszczenie zakrętów pod kątem 90 stopni. Długość trasy wynosi 1460mm.

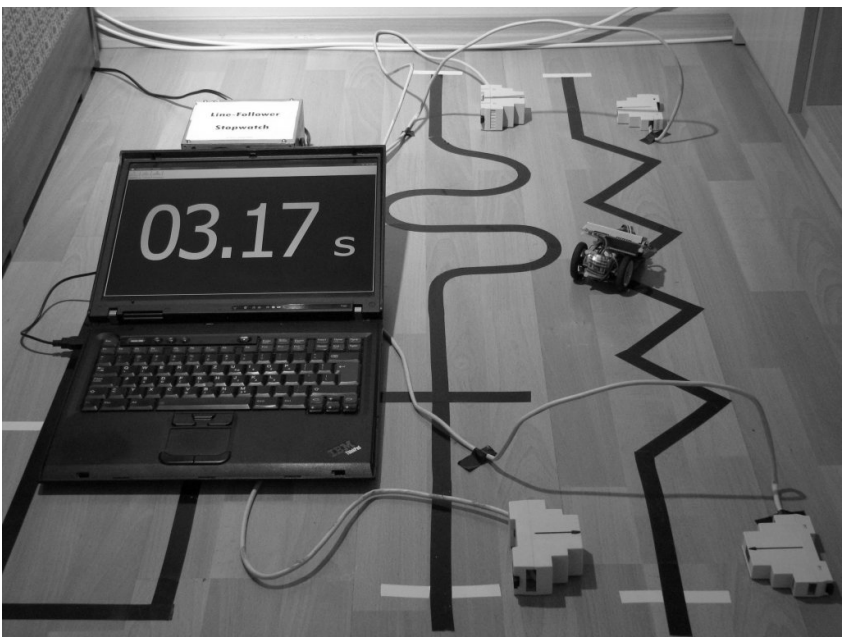


Rys. 7.3: Program diagnostyczny.



Rys. 7.4: Program wizualizujący fazy algorytmu obliczającego funkcję pozycji linii.

Rys. 7.5: Widok na stanowisko pomiarowe.



Rys. 7.6: Robot podczas pomiarów.

Rys. 7.7: Trasy użyte do przeprowadzenia testów porównawczych.

7.4.4. Wyniki testów

Na każdej z tras i dla każdego algorytmu przeprowadzono 30 kompletnych przejazdów. W tabeli 7.1 przedstawiono uśrednione rezultaty czasu przejazdu oraz funkcji wyniku. Funkcja wyniku jest miarą odległości środka robota od linii w czasie i została wyliczona na podstawie danych pomiarowych z czujników robota. Dodatkowo na podstawie uśrednionego czasu przejazdu oraz znając długość toru, wyliczone zostały średnie prędkości przejazdu.

Tab. 7.1: Uśrednione wyniki eksperymentów.

	algorytm liniowy			algorytm `7-if`		
numer trasy	czas [s]	wynik	prędkość [m/s]	czas [s]	wynik	prędkość [m/s]
1	5,64	23917	0,253	7,67	53181	0,186
2	6,12	43559	0,270	8,10	46491	0,204
3	6,13	38344	0,238	7,20	41687	0,203

Natomiast w tabeli 7.2 przedstawiono procentową poprawę wskaźników jakości dla algorytmu liniowego względem algorytmu bazowego `7-if`.

Tab. 7.2: Wyniki porównawcze poprawy jakości danych.

numer trasy	Δ czas [%]	Δ wynik [%]
1	26,41	54,92
2	24,48	6,33
3	14,81	8,02

7.5. Wnioski

Najlepszą poprawę zarówno czasu przejazd, jak i wartości funkcji wyniku, uzyskano na pierwszym torze. Przyczyną takiej sytuacji jest przede wszystkim długa prosta poprzedzająca ostry zakręt. Robot z algorytmem `7-if` miał problem, aby utrzymać się w zakręcie i w większości przypadków w całości wyjeżdżał poza trasę. Problemem było również rozwinięcie maksymalnej prędkości na prostych, ponieważ w algorytmie `7-if` mamy do czynienia z częstymi przeregulowaniami i ustabilizowanie toru jazdy idealnie wzdłuż wyznaczonych prostych odcinków trasy zdarzało się sporadycznie.

Wartość funkcji wyniku dla algorytmu `7-if` wypadała dużo lepiej na trasach 2 i 3. Trasy, choć kręte, pozbawione były długich prostych, co zmniejszało prędkość wejścia w zakręt, a tym samym ryzyko wypadnięcia z toru. Duże natężenie zakrętów niwelowało również występujące przeregulowania. Mimo tego algorytm liniowy na obu tych trasach okazał się lepszy, pod kątem obu wskaźników. Choć tutaj w przeciwieństwie do trasy nr 1 większą poprawę odnotowano dla czasu przejazdu niż dla odwzorowania. Duża gęstość i ilość zakrętów wpływa negatywnie na jakość odwzorowania trasy.

Widoczne jest, że największy zysk można uzyskać na odcinkach prostych i wymagających dużych zmian prędkości, zaś dużo mniejszy zysk jest na samych zakrętach. Jednak na każdej z tras, które wspólnie skupiają najtrudniejsze fragmenty spotykane na torach dla robotów klasy *Line-Follower*, zaimplementowany algorytm okazał się skuteczniejszy. Potwierdza to poprawności i skuteczności obranych działań i pokazuje, że w tej kategorii robotów można za sprawą samego oprogramowania znacząco poprawić działania robota.

Literatura

[1] I. Colak. Evolving a line following robot to use in shopping centers for entertainment. In *IECON '09. 35th Annual Conference of IEEE*. Industrial Electronics, 2009.

[2] V. Girbes. On generating continuous-curvature paths for line following problem with curvature and sharpness constraints. In *Robotics and Automation (ICRA), IEEE International Conference*, 2011.

[3] M. A. Morton. Traction control study for a scaled automated robotic car. *Buletin of the Faculty of the Virginia Polytechnic Institute and State University*, 2004.

[4] Laszlo Mero. An optimal line following algorithm. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 3(5):593–598, May 1981.
[5] Atmel AVR127: Understanding ADC parameters. Technical report, Atmel, nov. 2011.
[6] Avr120: Characterization and calibration of the adc on an avr. Technical report, Atmel, feb. 2006.
[7] P. Hadam. *Projektowanie systemów mikroprocesorowych*. Wydawnictwo BTC, Warszawa, 2004.
[8] R. G. Lyons. *Wprowadzenie do cyfrowego przetwarzania sygnałów*. Wydawnictwa Komunikacji i Łączności, Warszawa, 2010.
[9] T. Zieliński. *Cyfrowe przetwarzanie sygnałów. Od teorii do zastosowań*. Wydawnictwa Komunikacji i Łączności, Warszawa, 2009.

ROZDZIAŁ 8

ODNAJDOWANIE TEKSTU NA SKANACH MAP

Z. Pietrowska, M. Puchalska, T. Płatek

8.1. Wprowadzenie

Od zarania dziejów ludzkości wiedza o otaczającym nas świecie była przekazywana z pokolenia na pokolenie w postaci mówionej, obrazkowej, a po wynalezieniu pisma – również w formie pisanej. Wraz z postępem cywilizacyjnym do przechowywania tych przekazów, obok pamięci ludzkiej, zaczęto wykorzystywać różnorodne materiały pozwalające na ich gromadzenie oraz utrwalanie. Mimo, że dawno temu namalowane na skałach, wyryte w kamieniu, wypalone na skórze, wykreślone na pergaminie lub wydrukowane na papierze przekazy te wciąż stanowią bezcenne źródło informacji dla wielu badaczy.

Obecnie, w dobie komputerów i komunikacji bezprzewodowej, wspomnianą spuściznę wielu pokoleń poddaje się istotnej transformacji. Transformacja ta polega na zamianie formy analogowej jej zapisu w formę cyfrową. Dzięki niej możliwym staje się współdzielenie zasobów, automatyczne ich wyszukiwanie i analizowanie, nie mówiąc już o możliwości multimedialnej prezentacji.

Zamiana na formę cyfrową zazwyczaj rozpoczyna się od skanowania materiałów, po którym następuje dalsze przetwarzanie. Samo skanowanie materiałów jest procesem dość prostym. Jego wynikiem jest zbiór punktów obrazujący obiekt zainteresowania (wynikiem jest dwuwymiarowy obraz lub trójwymiarowa chmura punktów). Dużo trudniejszym w realizacji jest dalsze przetwarzanie tak uzyskanych danych. W zależności od postawionego celu przetwarzanie to może przyjąć różną postać i kończyć się różnymi wynikami.

Szczególnym materiałem historycznym, podlegającym opisanej transformacji, są dawne mapy. Mapy, które powstały jeszcze przed opracowaniem nowoczesnych metod geodezyjnych i kartograficznych, były tworzone bez zachowania skali ani odwzorowania. Dają one ogólny pogląd na to, jak postrzegano wtedy przestrzeń, oraz dostarczają informacji o tej przestrzeni poprzez wyróżnienie istotnych jej elementów. Mapy późniejsze, a więc zachowujące skalę i odwzorowanie (przynajmniej w jakimś zakresie), pełnią podobną funkcję, przy czym dokładniej oddają informacje o przestrzeni. Ze względu na to, że mapy dawne

były tworzone na różne sposoby, przez różnych autorów, ważne jest opracowanie metod, które pozwalają na pozyskiwanie z nich informacji w sposób efektywny.

Mapy publikowane są często z podaniem ich zakresu przestrzennego, tzw. „Bounding Box" [1]. Dzięki niemu można je przestrzennie wyszukiwać z zasobach bibliotecznych, a po georektyfikacji – również umieszczać na osobnych warstwach w serwisach mapowych. Co więcej, mapy niosą ze sobą dane w postaci opisów tekstowych na nich umieszczanych. Na ich podstawie można pozyskać wiedzę historyczną, przykładowo na podstawie nazw miejscowości z map różnego okresu można stwierdzić, jak się one zmieniały na przestrzeni dziejów.

W dalszej części rozdziału opisano sposoby pozyskiwania danych z opisów tekstowych zamieszczonych na dawnych mapach. Scharakteryzowano szczegółowo jeden z algorytmów oraz przedstawiono efekty jego zastosowania na przykładzie przetwarzania wybranej mapy.

8.2. Dawne mapy w Internecie

Mapa jest to uogólniony obraz powierzchni Ziemi lub jej części, wykonywany na płaszczyźnie, w skali, według zasad odwzorowania kartograficznego, przy użyciu umownych znaków graficznych. Mapa może stanowić narzędzie badań historycznych oraz do prezentacji wyników. Mapy dawne różnią się jednak znacząco od map teraźniejszych. W ich przypadku trudno jest mówić o jakimś konkretnym odwzorowaniu kartograficznym, zaś sposób reprezentacji danych i ich wykonania zależy od tego, w jakim okresie i przez kogo zostały stworzone. W ogólnym zarysie mapy można podzielić ze względu na ich treść na następujące kategorie:

- ogólnogeograficzne: topograficzne, przeglądowo-topograficzne, przeglądowe;
- tematyczne: społeczno-gospodarcze, przyrodnicze, fizyczno-geograficzne, polityczno-administracyjne.

Kryterium podziału map może stanowić również ich skala. Stosując to kryterium można wyróżnić:

- mapy wielkoskalowe (skala od 1:100 do 1:10 000);
- mapy średnioskalowe (od 1:20 000 do 1:300 000);
- mapy małoskalowe (poniżej 1:500 000).

Obecnie powstaje wiele portali udostępniających dane przestrzenne, w tym dawne mapy i materiały historyczne. Przykładem jest DIGMAP - serwis, którego celem jest promowanie dziedzictwa kulturowego i naukowego [2]. W serwisie tym udostępniane są mapy dawne, jak również informacje z nich pozyskane. Innym przykładem jest MapRank - zaawansowana wyszukiwarka pozwalająca użytkownikom na skutecznie i intuicyjne przeprowadzenie analiz (wyszukiwanie) dokumentów z obszernych bibliotek cyfrowych lub geograficznych baz danych [3].

Pozyskiwanie danych z map dawnych, a w tym nazw obiektów fizjograficznych, granic podziału administracyjnego, nazw miejscowości jest szczególnie interesujące, gdy można dokonać ich analizy porównawczej ze stanem obecnym. Dobrym źródłem referencyjnym do tych badań są oficjalne gazetery, jak np. Pań-

stowy Rejestr Nazw Geograficznych. Dzięki danym pozyskanym z map dawnych oraz gazeterom możliwe byłoby np. zbadanie przebiegu zmian nazw miejscowości na przestrzeni dziejów.

Gazeter w infrastrukturze informacji przestrzennej pełni rolę serwisu kojarzącego nazwy obiektów geograficznych z ich przestrzenną lokalizacją. Każdy gazeter ma swoje szczególne przeznaczenie oraz dotyczy wybranych obiektów [4]. Gazeter to inaczej słownik zawierający listę nazw obiektów (np. miejscowości) i ich geograficzne położenie (wyrażone w wybranym układzie odniesień przestrzennych). Może on zawierać również dodatkowe dane, jak wykaz źródeł związanych z obiektem, opis otoczenia itp. Dostęp do danych gazetera można uzyskać w tradycyjny sposób (pobierając dane w postaci opublikowanej listy) lub też drogą elektroniczną (poprzez interfejs sieciowy, którym zazwyczaj jest interfejs usługi WFS).

8.3. Automatyczne wykrywanie tekstów na mapach

Odnajdowanie i rozpoznawanie tekstów to dwa często wyróżniane etapy automatycznego wykrywania tekstów. Celem odnajdowania tekstu jest wyznaczenie obszaru na obrazie, na którym znajduje się tekst. Podczas wykrywania i rozpoznawania tekstu obszar ten jest analizowany, a wynikiem tej analizy jest zbiór znaków odpowiadający znalezionemu i rozpoznanemu tekstowi. Rozpoznawanie znaków i całych tekstów w sposób automatyczny stało się ważną dziedziną badań. Opracowany w ich ramach zestaw technik umożliwiający rozpoznawanie znaków w pliku graficznym o postaci rastrowej nazwano technikami OCR (ang. *Optical Character Recognition*). OCR jest obecnie ważną i powszechnie stosowaną technologią [5].

Mimo, iż w ostatnich latach zaproponowano wiele metod automatycznego odnajdowania i rozpoznawania tekstów, efektywne wykrywanie tekstów jest nadal wyzwaniem. Spowodowane jest to różnymi warunkami przeprowadzania tej czynności, np. rozmiarami czcionki, stylami pisma oraz kolorami zarówno tła jak i pisma. Zgodnie z [6] metody wykrywanie tekstu można podzielić na dwie główne grupy: oparte na wykorzystaniu tekstur oraz oparte na analizie elementów połączonych. Znaczna część tych metod pozwala na bardzo dobre wykrywanie tekstu o różnych rozmiarach, czcionkach, przy pewnych warunkach upraszczających problem (jak np. równoległe ułożenie wierszy pisma).

W przypadku map istnieją pewne zalecenia dotyczące sposobu redakcji konkretnych ich zbiorów, w tym zalecenia co do parametrów technicznych tekstu (przykład przedstawiono w tabeli 8.1). Mimo to trudno jest wskazać na ogólnie obowiązujący standard, nie mówiąc już o tym, że w przypadku map dawnych taki standard po prostu nie istnieje.

Odnajdywanie tekstu na mapach wymaga pokonania licznych trudności. Należą do nich np. różne kierunki oraz o różne kroje i kolory napisów. Co więcej, podczas wykrywania tekstów na mapach pojawiają się problemy dodatkowe, związane z treścią prezentowaną na mapie, a więc: siecią dróg, hydrografią, symbolami obiektów topograficznych itp. W związku z dużą różnorodnością tych da-

nych podczas opracowywania metod ich przetwarzania wprowadza się zazwyczaj pewne ograniczenia co do analizowanego zakresu tematycznego.

Tab. 8.1: Przykładowe parametry pisma występującego na mapach.

Parametr	Rodzaje	Przykład	Kategorie
Rodzaj pisma	Wersalik	**POLSKA, WARSZAWA**	Państwa, regiony, stolice większe miasta
	tekst	Gdańsk, Wyspa Sobieszewska	Wszystko pozostałe
Krój	bezszeryfowe		Państwa, regiony, miasta, przylądki
	szeryfowe		Morza, Oceany, rzeki, zatoki
pochylenia	proste	Półwysep Skandynawski	Państwa, regiony, miasta, przylądki, wyspy
	pochyłe	*Zatoka Pucka*	cieśniny, rzeki, zatoki
grubość	cieńkie	Wieżyca	szczyty
	pogrubione	**Województwo pomorskie**	Państwa, jednostki administracyjne.
	zwykłe	Gdańsk	Wszystko pozostałe
kolor	niebieski	*Wisła*	obiekty wodne
	zielony	*Trójmiejski Park Krajobrazowy*	Obiekty związane z lasami.
	sepia	*140*	Poziomice.
	zwykłe	Gdańsk	Wszystko pozostałe

Przykłady stosunkowo prostych algorytmów wykrywania tekstów opisano w artykułach [7],[8] i [6] . Metody przedstawione w artykułach [9] , [10] oraz [11] pozwalają na odnajdowanie tekstu o różnej orientacji, ale są o wiele trudniejsze pod względem implementacyjnym od wcześniej wymienionych.

Jedną z metod wykrywania tekstów jest odporny system do dokładnego wykrywania i lokalizacji tekstu na zdjęciach środowiska miejskiego, opisany w artykule [10]. W metodzie tej można wyodrębnić dwa etapy: wykrywanie tekstu oraz lokalizację tekstu.

Wykrywanie tekstu składa się z dwóch kroków: wstępnego przetwarzania oraz analizy regionu. We wstępnym przetwarzaniu następuje transformacja obrazu RGB na obraz w skali szarości, następnie jest tworzona piramida obrazów poprzez interpolacje metodą najbliższego sąsiada (ponieważ na danym obrazie mogą występować testy o różnych rozmiarach). Natomiast na etapie analizy dla każdej kolejnej próbki wychwytywane oraz klasyfikowane są cechy charakterystyczne. Przy wykrywaniu cech można posłużyć się filtrami Gaussa, a do kla-

syfikacji próbek należy zbudować kaskadowy klasyfikator AdaBoost (stosowany z powodzeniem w wielu dziedzinach, np. przy wykrywaniu twarzy).

Lokalizacja tekstu również składa się z dwóch kroków: generacji linii tekstu oraz ekstrakcji tekstu. Generacja linii tekstu polega na grupowaniu okien w miejscach podejrzanych o występowanie tekstu i, w zależności od procentowej ilości powierzchni wspólnej, zaklasyfikowania obszaru jako tekst bądź jego odrzucenia. Ekstrakcja tekstu ma przygotować obszary do rozpoznawania znaków. Składają się na nią: lokalna binaryzacja oraz analiza połączonych obiektów składowy.

Metoda wyszukiwania tekstu na obrazach poprzez lokalne progowanie (ang. *finding text in images via local thresholding*) została zaproponowana w artykule [6]. Jest ona wzorowana na algorytmie wykrywania scen w filmach. Pozwala na lokalizowanie napisów o różnej czcionce i wielkości oraz w otoczeniu złożonego tła, lecz działa jedynie dla poziomych tekstów oraz nie wykrywa bardzo małych znaków. Pierwszym jej krokiem jest przekonwertowanie obrazu wejściowego do modelu barw YUV. W dalszych etapach analizowana jest jedynie składowa Y (luminacja). Wykrywanie krawędzie, odbywające się w następnej kolejności, polega na wyznaczeniu różnicy pomiędzy wartością luminancji dla danego piksela a jej wartością dla jego trzech sąsiadów (lewym, górnym i przystającym do lewego-górnego wierzchołku). Jako wartość krawędzi przyjmowana jest największa z tych wartości. Następnie obraz jest dzielony na poziome linie. Dla każdej z tych linii wyznaczany jest histogram, pozwalający na odczytanie ilości krawędzi o wysokim kontraście. Kolejnym krokiem jest próbkowanie w dół (ang. *downsampling*), przeprowadzone zgodnie ze wzorami:

$$(D(H, I/N) = H(I + N) - H(i),$$

$$D'(H, I/N) = ABS(D(H, I/N)),$$

gdzie I oznacza numer linii, natomiast N dla każdego I spełnia warunek $ImodN = 0$. Wartości szczytowe na wykresie po przeprowadzonym procesie wskazują na miejsca początku i końca wyrazów. Ostatecznie, obraz poddawany jest przetwarzaniu końcowemu polegającemu na uzupełnieniu luk w postaci pojedynczych pikseli.

Na szczególną uwagę zasługuje metoda wykrywania tekstu w nagraniach wideo, opisana w [9]. Autorzy [9] proponują rozpoczęcie analizy obrazu od zastosowania w dziedzinie częstotliwości najpierw idealnego filtru dolnoprzepustowego [12], a następnie filtru Laplace'a [12]. Pozwala to na wstępne wyszczególnienie fragmentów, które z uwagi na fakt występowania na odpowiednich obszarach znacznych rozbieżności wartości między pikselami potencjalnie mogą być tekstem. Po utworzeniu mapy maksymalnych rozbieżności można podzielić mapę na dwa zbiory wykorzystując algorytm centroidów [13]. Morfologiczna operacja otwarcia przygotowuje obraz do dalszej obróbki, czyli klasyfikacji komponentów składowych na komponenty proste i złożone przez wykorzystanie szkieletu obszarów wstępnie zaklasyfikowanych jako zawierające tekst. Po podziale komponentów złożonych na proste z wykorzystaniem punktów intersekcji szkieletów odpowiednich komponentów analizowana jest ich graniczna gęstość oraz parametr zwany liniowością (co pozwala na eliminację elementów, które w pierwszej

fazie nie zostały odrzucone a nie stanowią napisów). Na potrzeby niniejszego rozdziału metodę tę nazwano *Algorytmem opartym o filtr Laplace'a*.

8.4. Algorytm oparty o filtr Laplace'a

Z uwagi na potrzebę efektywnego wykrywania tekstu o różnej orientacji, czcionce oraz często finezyjnym kształcie zdecydowano się na implementację algorytmu opartego na filtrze Laplace'a [9]. Algorytm ten składa się z czterech głównych etapów, opisanych w kolejnych podrozdziałach:

1. wykrywania tekstu,
2. klasyfikacji połączonych komponentów,
3. segmentacji połączonych komponentów,
4. eliminacji błędnie zaklasyfikowanych obszarów.

8.4.1. Wykrywanie tekstu

Celem pierwszego kroku analizy obrazu jest zidentyfikowanie obszarów, które mogą, aczkolwiek nie muszą być tekstem. Główna idea polega na tym, że lepiej przyjąć słabe kryterium klasyfikacji niż od razu na początku odrzucić zbyt wiele regionów.

Współrzędne obrazu w dziedzinie przestrzennej oznaczane są przez (x, y), w dziedzinie częstotliwości przez (u, v). Sam obraz w dziedzinie przestrzennej reprezentowany jest symbolem g. Aby wygładzić szum, najpierw obraz transformowany jest do dziedziny częstotliwości i stosowany jest idealny filtr dolnoprzepustowy

$$H_1(u,v) = \begin{cases} 1 & \text{jeżeli } \sqrt{(u)^2 + (v)^2} <= D_0 \\ 0 & \text{w przeciwnym wypadku} \end{cases} . \tag{8.1}$$

przy czym D_0 jest to wybrana eksperymentalnie wartością. Następnie używany jest filtr Laplace'a , również w dziedzinie częstotliwości

$$H_2(u,v) = -(u^2 + v^2).$$

Użycie każdego z filtrów w dziedzinie częstotliwości oznacza pomnożenie odpowiednich składowych macierzy filtru przez odpowiednie składowe transformaty Fouriera obrazu g. W efekcie, w dziedzinie przestrzennej w obszarach zawierających napisy piksele mają znacznie odbiegające od siebie skrajne wartości, podczas gdy obszary z małą amplitudą zmian z całą pewnością nie zawierają tekstu. Zazwyczaj piksele o wartości zero odpowiadają przejściom między tekstem a tłem. Zasada ta przestaje obowiązywać, gdy kontrast obrazu jest niewielki lub faktura bardzo skomplikowana. Miarą, która daje się zaadaptować do tego typu przypadków, jest maksymalna rozbieżność (dalej oznaczana skrótem MD). MD jest transformacją lokalną i jest zdefiniowana jako różnica pomiędzy maksy-

malną i minimalną wartością pikseli zawierających się w oknie rozmiaru $1 \times N$.

$$\begin{aligned} Max(x,y) &= \max_{\forall t \in [-\frac{N}{2}, \frac{N}{2}]} g(x, y-t) \\ Min(x,y) &= \min_{\forall t \in [-\frac{N}{2}, \frac{N}{2}]} g(x, y-t) \\ MD(x,y) &= Max(x,y) - Min(x,y). \end{aligned} \tag{8.2}$$

Mapa MD jest uzyskiwana przez przesuwanie okna po całym obrazie. Jak wspomniano wcześniej, wartości pikseli mapy MD są znacznie większe tam, gdzie występuje tekst, w związku z tym wyróżniamy dwa klastry danych: C_1 – tekst, C_2 – nie tekst. Podział danych na klastry realizowany jest algorytmem centroidów, korzystającym z metryki euklidesowej odległości MD. Istotne jest, że klaster C_1 posiada większą wartość średnią niż klaster C_2. Aby usunąć pojedyncze, nie pasujące piksele na uzyskanym w ten sposób obrazie binarnym, można zastosować operacje morfologicznego otwarcia.

8.4.2. Klasyfikacja połączonych komponentów

Dalszej obróbce podlega klaster C_1, czyli tekst. Przed przystąpieniem do omawiania drugiego etapu klasyfikacji konieczne jest przytoczenie kilku definicji:

Prosty połączony komponent - to pojedynczy łańcuch tekstowy lub w całości błędnie zaklasyfikowany jako tekst element tła.

Złożony połączony komponent - to obiekt złożony z wielu łańcuchów tekstowych oraz elementów tła, które zostały błędnie zaklasyfikowane.

Klaster C_1 składa się z wielu komponentów połączonych, zarówno prostych jak i złożonych. Celem kroku drugiego algorytmu opartego o filtr Laplace'a jest określenie, które komponenty są proste, a które złożone. Do tego celu wykorzystywany jest szkielet danego komponentu (czyli obraz złożony z minimalnej liczby pikseli zapewniającej spójność obszaru). Komponent klasyfikowany jest jako złożony, gdy liczba intersekcji (czyli punktów przecięć linii tworzących szkielet) jest różna od 0, w przeciwnym razie komponent jest klasyfikowany jako prosty.

Szkieletyzacja jest operacją pozwalającą wyodrębnić osiowe punkty (szkielety) w analizowanym obrazie. Szkielet figury jest zbiorem wszystkich punktów, które są równoodległe od co najmniej dwóch punktów należących do brzegu. Szkielet figury jest znacznie od niej mniejszy, a odzwierciedla w pełni jej topologiczne własności [14].

8.4.3. Segmentacja połączonych komponentów

Do wyświetlenia ostatecznego wyniku wyszukiwania tekstu na obrazie mogą posłużyć proste elementy. Zanim będzie to jednak możliwe, należy rozdzielić złożone komponenty na komponenty proste oraz wyeliminować komponenty stanowiące elementy tła. Krok trzeci algorytmu ma na celu podział złożonych komponentów. W tym celu wyodrębnia się segmenty szkieletu, czyli jego odcinki mię-

dzy punktami intersekcji lub punktami krańcowymi (np. przez usunięcie punktów intersekcji). Na koniec należy wyekstrahować składowe komponentu odpowiadające poszczególnym segmentom szkieletu i dodać je jako osobne elementy do zbioru komponentów prostych, uzyskanego w poprzednim etapie klasyfikacji.

8.4.4. Eliminacja błędnie zaklasyfikowanych obszarów

Zbiór komponentów prostych, którym można dysponować na tym etapie, zawiera zarówno znaleziony tekst jak i elementy tła. Aby rozróżnić komponenty zawierające tekst wystarczy przebadać liniowość komponentu

$$S = \frac{\textit{Długość szkieletu}}{\textit{Odległość między krańcami szkieletu}} \tag{8.3}$$

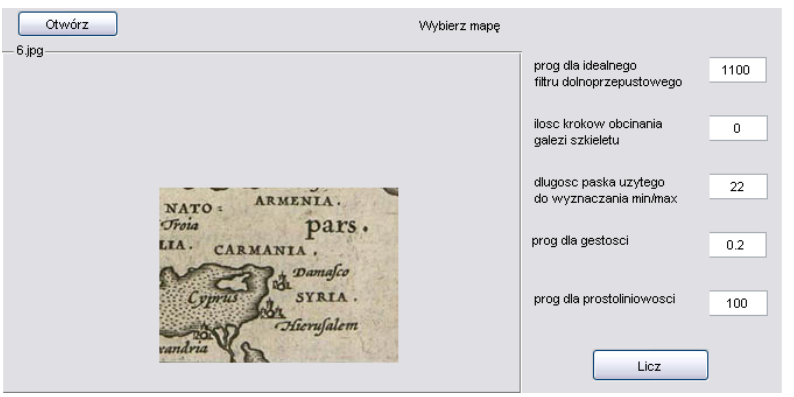
oraz jego gęstość

$$E = \frac{\textit{Długość krawędzi}}{\textit{Objętość komponentu}}. \tag{8.4}$$

Połączony komponent zawiera tekst, jeżeli spełnione są warunki $S < S_0$ oraz $E \geq E_0$, gdzie S_0 i E_0 to wartości dobrane empirycznie.

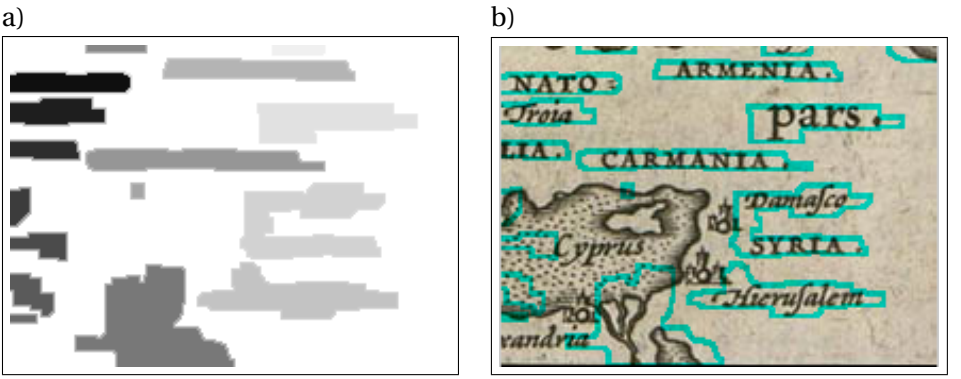
8.5. Opis aplikacji służącej odnajdowaniu tekstu na mapach.

Aplikacja została wykonana w środowisku MATLAB w wersji 2010b. W celu łatwej obsługi wyposażona została w graficzny interfejs użytkownika. Przedstawione na rysunku 8.1 okno aplikacji składa się z następujących elementów: obszaru zawierającego wczytaną mapę oraz panelu zawierającego parametry algorytmu.

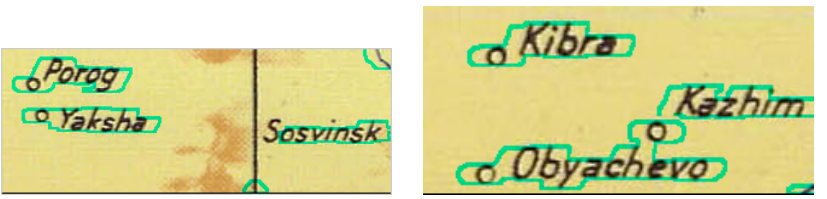


Rys. 8.1: Podstawowe okno aplikacji.

Mapę można wczytać używając przycisk Otwórz, a uruchomienie algorytmu następuje po naciśnięciu przycisku Licz. Po zakończeniu działania algorytmu pojawiają się dwa nowe okna z wygenerowanymi obrazami (zobacz rysunek 8.2), które można zapisać celem dalszej obróbki.



Rys. 8.2: Obrazy wygenerowane jako wynik działania algorytmu: a) znalezione obszary b) granice obszarów na podkładzie mapy.



Rys. 8.3: Wynik działania algorytmu dla wartości parametrów: 1100, 0, 22, 0.2, 100.

Działanie zaimplementowanego algorytmu można dostrajać za pomocą zestawu następujących parametrów:

1. **próg dla idealnego filtru dolnoprzepustowego** - wartość współczynnika D_0 stosowanego we wzorze (8.1) jako wartość progu dla idealnego filtru dolnoprzepustowego.

2. **ilość kroków obcinania gałęzi szkieletu** - przy generowaniu szkieletu obiektów złożonych pojawiają się pewne nieregularności, które nie niosą ze sobą żadnej informacji, a mogą powodować podział obszaru tekstu na kilka podobszarów. W niektórych przypadkach korzystne jest wykonanie kilku iteracji obcinania gałęzi szkieletu, czyli jak gdyby odejmowania „wystających" punktów . Ilość kroków obcinania gałęzi szkieletu jest dobierana empirycznie.

3. **długość paska użytego do wyznaczania min/max** - wartość ta wykorzystywana jest do policzenia dwóch macierzy wartości minimalnych i maksymal-

nych w otoczeniu punktu,a na ich podstawie jest liczona macierz MD (8.2). Parametr ten musi mieć wartość parzystą.

4. **próg dla gęstości** - wartość współczynnika E_0, połączony komponent zawiera tekst, jeżeli spełniony jest warunek $E \geq E_0$, gdzie E jest obliczane ze wzoru (8.4). Wartość ta jest dobierana empirycznie.

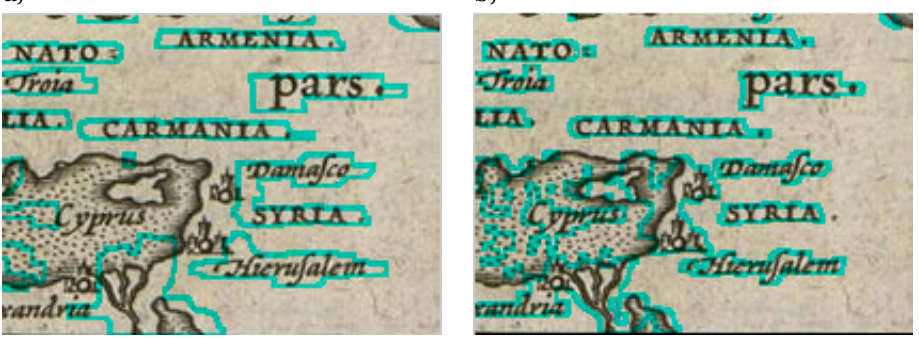
5. **próg dla prostoliniowości** - wartość współczynnika S_0, połączony komponent zawiera tekst, jeżeli spełniony jest warunek $S < S_0$, gdzie S jest wyznaczane za pomocą wzoru (8.3). Wartość ta jest dobierana empirycznie.

Przykładowe wyniki działania algorytmu odnajdowania tekstu na mapach zostały zaprezentowane poniżej. Na rysunku 8.3 widać, że aplikacja bardzo dobrze sobie radzi z odnajdowaniem tekstu na mapach. Na rysunku 8.4a został przedstawiony wynik działania aplikacji dla mapy z dodatkowymi oznaczeniami w formie rysunków, które, nota bene, również niosą ze sobą pewną informację. Niestety, nie wszystkie napisy zostały odszukane. Po zmniejszeniu wartości trzeciego parametru, tj. długości paska użytego do wyznaczania min/max do wartości 6, algorytm odnalazł wszystkie napisy (rysunek 8.4b), jednakże kosztem zaklasyfikowania większej części rysunków również jako tekst.

8.6. Podsumowanie

W niniejszym rozdziale przedstawiono sposoby uzyskiwania informacji z opisów tekstowych na mapach. Następnie streszczono działanie wybranych metod, jak również szczegółowo opisano algorytm oparty o filtr Laplace'a. W oparciu o wspomniany algorytm stworzono aplikację do odnajdywania tekstu na mapach. Jako, że wyniki działania algorytmu są bardzo dobre, można wykorzystać je do dalszej obróbki za pomocą OCR'a. W ten sposób informacje, które niosą ze sobą mapy dawne, można zebrać w formie cyfrowej i wykorzystać np. do analiz historycznych.

a) b)



Rys. 8.4: Wyniki działania algorytmu dla wartości parametrów: a) 1100, 0, 22, 0.2, 100, b) 1100, 0, 6, 0.2, 100.

Literatura

[1] OpenGIS Web Map Service (WMS) Implementation Specification 1.3.0. http://portal.opengeospatial.org/files/?artifact_id=14416.

[2] Digimap. Discovering our past world with digitized maps. http://portal.digmap.eu, 2012.

[3] Maprank search. http://www.klokantech.com/mapranksearch/, 2012.

[4] K.K. Kemp, editor. *Encyclopedia of geographic information science*. SAGE Publications, Inc., 2008.

[5] S. Mori, H. Nishida, and H. Yamada, editors. *Optical Character Recognition*. John Wiley & Sons, Inc., 1999.

[6] J. Gllavata, R. Ewerth, and B. Freisleben. Finding text in images via local thresholding. In *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology*, 2003.

[7] R. Minetto, N. Thome, M. Cord, N. Leite, and J. Stolfi. Snoopertrack: Text detection and tracking for outdoor videos. In B. Macq and P. Schelkens, editors, *ICIP*, pages 505–508. IEEE, 2011.

[8] V. Wu, R. Manmatha, and E.M. Riseman. Finding text in images. In *ACM DL*, pages 3–12. ACM, 1997.

[9] P. Shivakumara, T.Q. Phan, and Ch.L. Tan. A laplacian approach to multi-oriented text detection in video. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 33(2):412–419, February 2011.

[10] Y. Pan, X. Hou, and Ch.-L. Liu. A robust system to detect and localize texts in natural scene images. In *The Eighth IAPR International Workshop on Document Analysis Systems, DAS '08*, pages 35 –42, sept 2008.

[11] W. Pan, T.D. Bui, and C.Y. Suen. Text detection from natural scene images using topographic maps and sparse representations. In *International Conference on Image Processing, ICIP09*, pages 2021–2024, 2009.

[12] J. Izydorczyk, G. Płonka, and G. Tyma. *Teoria sygnałów. Wstęp. Wydanie II poprawione i uzupełnione*. Wydawnictwo Helion, 2008.

[13] A. Drozdek. *Wprowadzenie do kompresji danych*. Wydawnictwa Naukowo Techniczne, 2007.

[14] Przetwarzanie i analiza obrazów komputerowych - algorytmy, idee i praktyczne przykłady. Szkieletyzacja. http://www.analizaobrazu.za.pl/teoria.html#szkieletyzacja, 2012.

ROZDZIAŁ 9

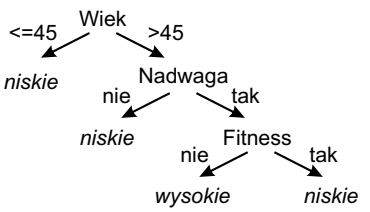
ROZMYTE DRZEWA DECYZYJNE

D. Powązka, P. Synak

Współczesne techniki sztucznej inteligencji muszą sobie radzić z coraz większą ilością danych. Dlatego od metod maszynowego uczenia się wymaga się swoistej odporności na duże zbiory danych wejściowych (lub uczących). Jedną z takich metod sztucznej inteligencji używaną w problemach klasyfikacji jest budowa drzewa decyzyjnego. Metoda ta, niestety, ma pewne wady i niedoskonałości. W celu zwiększenia jej efektywności zaproponowano w niej pewne usprawnienia. Niniejszy rozdział poświęcony jest przedstawieniu tych właśnie usprawnień.

9.1. Drzewa decyzyjne

Drzewa decyzyjne (ang. *decision trees, DTs*) są najczęściej wykorzystywane w procesie wyboru strategii działania bądź też przy zadaniach związanych z klasyfikacją obiektów. Ogólna idea ich działania może być łatwo przedstawiona za pomocą grafu skierowanego. Niech będzie dany obiekt wejściowy X, opisany wektorem cech (tzn. składowe owego wektora mają reprezentować pewne cechy obiektu). Poszczególne węzły grafu odpowiadają warunkom narzuconym na daną cechę (X_j). Wędrówka po grafie rozpoczyna się od jego korzenia. W zależności od wartości danej cechy obiekty wejściowego następuje przejście do odpowiedniego potomka danego węzła. I tak dzieje się, aż do przejścia do węzła końcowego (liścia), reprezentującego rozwiązanie. Przykładowe drzewo decyzyjne przedstawiono na rysunku 9.1. Jak widać, różnicowanie ze względu na war-



Rys. 9.1: Drzewo decyzyjne do oceny prawdopodobieństwa wystąpienia zawału.

tość danej cechy następuje w węzłach grafu, na krawędziach zaś znajdują się poszczególne wartości rozgraniczające. Każde z rozwinięć drzewa powinno kończyć się przyporządkowaniem odpowiedniej strategii/klasy obiektowi wejściowemu. W przypadku przedstawionym na rysunku może być to zakwalifikowanie osoby o określonym wieku, wadze i aktywności sportowej do grupy osób mocno lub słabo zagrożonych zawałem (z wysoki lub niskim prawdopodobieństwem wystąpienia zawału).

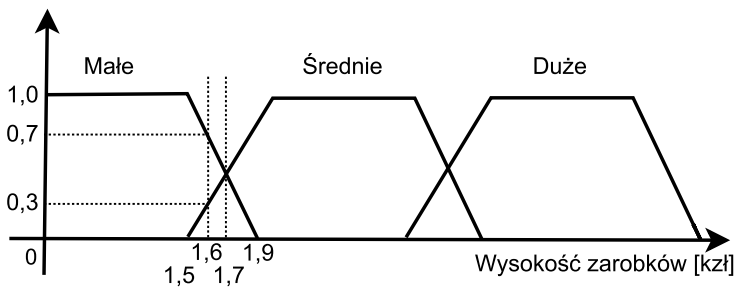
9.2. Logika rozmyta (*fuzzy sets*)

Logika rozmyta jest bardzo przydatna w przypadku próby opisu zjawisk, cech bądź też obiektów, których jednoznaczna klasyfikacja jest trudna lub nawet niemożliwa. W odróżnieniu od klasycznej algebry zbiorów, gdzie przynależność elementu do zbioru wartościowana jest binarnie (coś należy lub nie do danego zbioru), przynależność do zbiorów rozmytych definiowana jest przez następujące przekształcenie:

$$u_A(x) : X \rightarrow [0,1]. \tag{9.1}$$

W ten sposób element X może przynależeć do danego zbioru A w stopniu zdefiniowanym jako wartość z przedziału $[0,1]$. Pozwala to lepiej odzwierciedlić rzeczywistość przy opisie niektórych cech. Dobrym przykładem jest mogą być tutaj wartości zarobków. Bardzo trudno jest wskazać ostrą"granicę pomiędzy zarobkami niskimi a średnimi. Jeśli za X przyjąć zarobki danej osoby w wysokości 1600 zł, za N - zbiór wartości odpowiadających zarobkom niskim, zaś za S zbiór wartości odpowiadający zarobkom średnim, to w przypadku klasycznym należałoby zdefiniować ostrą granicę, np. $y = 1700$ zł, jako kryterium klasyfikacji. Skutkowałoby to tym, że $u_N(X) = 1$ oraz $u_S(X) = 0$.

Bardzo trudno jest wyznaczyć taką arbitralną i stałą (globalną) granicę. O wiele wygodniej byłoby w tym wypadku zastosować logikę rozmytą. W takim wypadku funkcje przynależności mogłyby zwracać następujące wartości: $u_N(X) = 0.7$ oraz $u_S(X) = 0.3$ zgodnie z funkcjami przynależności przedstawionymi na rysunku 9.2. Dzięki temu przyjęta reprezentacja kategorii zbiorów zarob-



Rys. 9.2: Wykres funkcji przynależności do kategorii *Małe, Średnie, Duże* w zależności od wysokości zarobków, wg [1].

ków będzie bardziej zbliżona do rzeczywistości niż kategorie przyjęte dla przypadku logiki klasycznej (np. w przypadku, gdy $X1 = 1699$ a $X2 = 1701$ otrzymuje się skrajnie różne przyporządkowania, mimo bardzo zbliżonej wartości cechy rozróżniającej).

Operacje na zbiorach rozmytych różnią się od operacji zdefiniowanych w klasycznej algebrze zbiorów. W przypadku logiki rozmytej i zbiorów rozmytych zdefiniowano operacje:

Suma $u_{A\cup B}MAX_{x\in X}\{u_A(x), u_B x\}$
Różnica $u_{A\cap B}MIN_{x\in X}\{u_A(x), u_B x\}$
Dopełnienie $u'_A(x) = 1 - u_A(x)$
Równość $A = B \Leftrightarrow u_A(x) = u_B(x), x \in X$
Zawieranie $A \in B \Leftrightarrow u_A(x) \geq u_B(x), x \in X$

Dodatkowo w logice rozmytej można zdefiniować następujące elementy:

nośnik $Support(A) = x|u_A(x) > 0$,
rdzeń $Core(A) = x|u_A(x) = 1$,
wysokość $h = supu_A(x)|x \in X$,

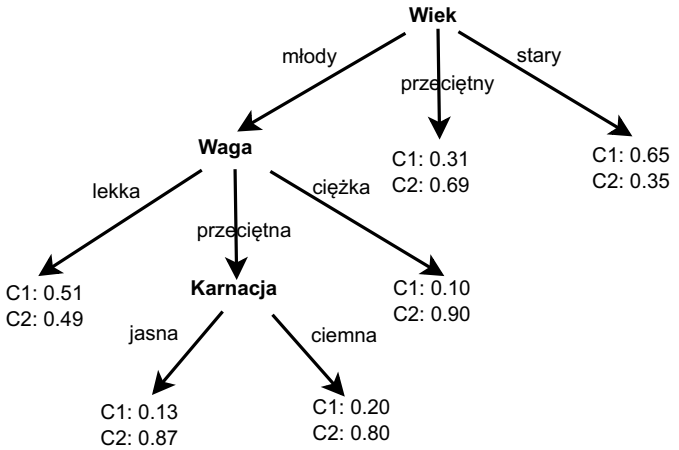
które będą potrzebne w przypadku opisu generowania rozmytego drzewa decyzyjnego.

9.3. Rozmyte drzewa decyzyjne

Rozmyte drzewa decyzyjne (ang. *Fuzzy Decision Trees*, FDTs) to modyfikacja metody drzew decyzyjnych, polegająca na wykorzystaniu logiki i zbiorów rozmytych. Główną zaletą takiego podejścia do reprezentacji procesu decyzyjnego jest zmniejszenie wrażliwości algorytmów generujących drzewa decyzyjne na specyficzne dane serii uczącej. Granica podziału (wartość podana na krawędziach drzewa, patrz 9.1) jest dzięki temu mniej ostra, a przez to można uzyskać większy stopień generalizacji reguł. Zmniejsza to zagrożenie przeuczenia (ang. *overfitting*) lub nadmiernej szczegółowości. Przykładowe drzewo decyzyjne pokazano na rysunku 9.3. Atrybuty uwzględniane w tym drzewie, takie jak: jasna, ciemna i podobne, są wyrażeniami rozmytymi (ang. *fuzzy terms*). Dane początkowe zawierały (przed fuzyfikacją) dla atrybutu karnacja następujące wartości: *biały*, *metys*, *afroamerykanin*. Fuzyfikacja (przekształcenie wartości $\{0,1\}$ na przedział wartości $[0,1]$) w przypadku atrybutu kolor włosów była następująca

- u_{jasna}(biały) = 0.8, u_{jasna}(afroamerykanin) = 0, u_{jasna}(metys) = 0.2
- u_{ciemna}(biały) = 0, u_{ciemna}(afroamerykanin) = .8, u_{ciemna}(metys) = 0.2

Kolejną istotną różnicą pomiędzy zwykłym drzewem decyzyjnym, a rozmytym drzewem decyzyjnym jest fakt, iż w przypadku DT liście (węzły bez potomków) jednoznacznie przypisują kategorię danemu na wejściu obiektowi. FDT działają trochę inaczej, każdy liść posiada wektor wszystkich możliwych klas (do których obiekt może być zakwalifikowany) wraz z rozmytą wartością funkcji przynależno-



Rys. 9.3: Przykładowe rozmyte drzewo decyzyjne, wg [2].

ści. W przypadku, gdy obiekt wejściowy ma następujące parametry (dla drzewa przedstawionego na rysunku 9.3):

- wiek: *młody*
- waga: *przeciętna*
- karnacja: *jasna*

to przynależy w 0.13 do klasy $C1$ i w 0.87 do klasy $C2$. Dzięki takiej reprezentacji można bardzo łatwo wyszczególnić przypadki problematyczne (występujące, gdy wartości przynależności do poszczególnych klas są zbliżone do siebie, tzn. $C1 \approx C2$). Takim przypadkiem problematycznym mógłby być obiekt z atrybutem waga:*lekki*. W takim przypadku wartości funkcji przynależności do klas $C1$ i $C2$ są zbliżone do siebie (odpowiednio 0.51 i 0.49). Zwykłe drzewo decyzyjne zwróciłoby wartości $C1 = 1$ i $C2 = 0$. Wartość funkcji przynależności (w przypadku, gdy jest znormalizowana, tzn. wysokość równa 1) może być interpretowana jako prawdopodobieństwo tego, że obiekt wejściowy X należy do klasy $C1$.

9.4. Budowa rozmytego drzewa decyzyjnego

W przypadku zwykłego drzewa decyzyjnego techniką wykorzystywaną do jego budowy (tzn. dobrania atrybutów do poszczególnych węzłów drzewa oraz dokładnej wartości parametru rozróżniającego) jest metoda minimalizacji entropii. Atrybut w danym węźle wybierany jest w taki sposób, by maksymalnie rozdzielić zbiory klas na podstawie danego atrybutu. Niech atrybut X_j tak dzieli zbiór próbek uczących, że w jednym węźle potomnym są wszystkie elementy klasy A, a w drugim węźle potomnym wszystkie elementy klasy B. Wtedy entropia wynosi 0 (najlepszy przypadek). Jeśli zaś atrybut X_j dzieli serię uczącą tak, że w jednym węźle znajdują się wszystkie próbki klas A i B, entropia przyjmuje wtedy wartość

maksymalną (równa 1). Można to wyrazić następującym wzorem

$$Ent_b = \sum_c \frac{n_{bc}}{n_b}(-log_2 \frac{n_{bc}}{n_b}), \tag{9.2}$$

gdzie n_{bc} jest liczbą próbek c w gałęzi b, n_b to całkowita liczba próbek w gałęzi b, natomiast n_t jest to całkowita liczba próbek we wszystkich gałęziach. W przypadku rozmytych drzew decyzyjnych użytkownik musi rozwiązać dwa problemy:

1. Dobrać optymalny algorytm tworzenia drzewa.
2. Heurystycznie dobrać funkcje odpowiedzialne za fuzzyfikację.

Pierwszy problem może być rozwiązany w analityczny i jednoznaczny sposób za pomocą jednego z dostępnych algorytmów tworzenia drzew decyzyjnych. Drugi problem jest o wiele bardziej skomplikowany. Nie ma w tym wypadku gotowych rozwiązań, możliwych do przyjęcia bez znacznych modyfikacji.

9.5. Algorytm ID3 - wersja rozmyta

W przypadku FDT jednym z bardziej popularnych algorytmów tworzenia drzewa jest rozmyta wersja algorytmu ID3. Idea tego algorytmu, maksymalizacja zysku informacji, jest bardzo zbliżona do metody minimalizacji entropii (entropia jest odwrotnie proporcjonalna do zysku informacji).

Rozmyty algorytm ID3, przedstawiony w publikacji [2], operuje na rozmytych zbiorach i generuje rozmytą wersję drzewa decyzyjnego. Drzewo to powstaje ze wszystkich atrybutów zdefiniowanych przez użytkownika. Składa się z wierzchołków, które reprezentują testowane atrybuty, gałęzi dla wartości atrybutów oraz liści, które oznaczają przynależność do klasy.

Algorytm ID3 w wersji rozmytej jest bardzo podobny do jego standardowej wersji. Różnicą jest to, że atrybut jest wybierany na podstawie wzmocnienia informacyjnego, które jest wyliczane na bazie prawdopodobieństwa przynależności do danego atrybutu. Rozważmy zbiór danych D, gdzie każda dana ma l numerycznych wartości dla każdego z atrybutu $\{A_1, A_2, \ldots, A_l\}$ oraz przynależność do klasy $C = \{C_1, C_2, \ldots, C_n\}$. Dodatkowo podane są zbiory rozmyte $F_{i1}, F_{i2}, \ldots, F_{im}$ dla każdego z atrybutu A_i. Niech D_k^C będzie rozmytym podzbiorem D, gdzie wszystkie dane należą do klasy C_k. Oznaczmy również przez $|D|$ sumę wszystkich funkcji przynależności zbioru D. Algorytm, który utworzy rozmyte drzewo decyzyjne składa się z poniższych kroków:

1. Utwórz korzeń drzewa w którym znajdują się wszystkie dane.
2. Jeśli wierzchołek t, z rozmytym zbiorem D, spełnia poniższe kryteria:
   a) proporcja danych w zbiorze dla klasy C_k jest większa lub równa ustalonemu progowi θ_r,

$$\frac{|D_k^C|}{|D|} \geq \theta_r, \tag{9.3}$$

b) liczba danych w zbiorze jest mniejsza od ustalonego progu θ_n,

$$|D| \leq \theta_n, \tag{9.4}$$

c) nie ma więcej atrybutów do klasyfikacji,

to dany wierzchołek jest liściem z przynależnością do danej klasy C_k.

3. Jeśli nie wierzchołek nie spełnia powyższych kryteriów to wykonaj poniższe czynności:
   a) Dla każdego atrybutu A_i, gdzie $i = 1,2,\ldots,l$, oblicz wzmocnienie informacyjne $G(A_i,D)$, które jest dane wzorem 9.5. Wybierz atrybut A_{max}, który maksymalizuje wzmocnienie.
   b) Podziel zbiór D na podzbiory $D_1,D_2,\ldots,D_m$ ze względu na A_{max}. Wartość funkcji przynależności podzbioru D_j jest równa iloczynowi wartości funkcji przynależności do zbioru D oraz wartości funkcji przynależności $F_{max,j}$, dla atrybutu A_{max}.
   c) Wygeneruj nowe wierzchołki $t_1,t_2,\ldots,t_m$ dla podzbiorów $D_1,D_2,\ldots,D_m$ oraz określ zbiór $F_{max,j}$ do krawędzi, które łączą t i t_j.
   d) Zastąp D przez D_j, gdzie $j = 1,2,\ldots,m$, i powtórz procedurę od kroku 2 rekursywnie.

Wzmocnienie informacyjne $G(A_i,D)$ dla atrybutu A_i dane jest wzorem

$$G(A_i,D) = I(D) - E(A_i,D), \tag{9.5}$$

gdzie

$$I(D) = -\sum_{k=1}^{n}(p_k \log_2 p_k), \tag{9.6}$$

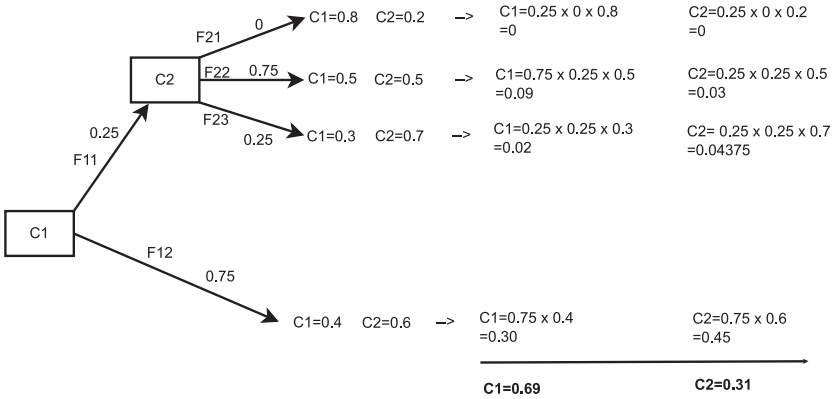
$$E(A_i,D) = \sum_{j=1}^{m}(p_{ij} I(D_{F_{ij}}), \tag{9.7}$$

$$p_k = \frac{|D^{C_k}|}{|D|}, \tag{9.8}$$

$$p_{ij} = \frac{|D_{F_{ij}}|}{\sum_{j=1}^{m}|D_{F_{ij}}|}. \tag{9.9}$$

Do wyboru klasy dla danego liścia zaproponowano trzy rozwiązania:

1. Wierzchołek jest przypisany do klasy o największej wartości funkcji przynależności.
2. Jeśli zachodzi warunek (a) w 2 kroku algorytmu postąp tak samo jak w kroku (a). Jeśli nie to wierzchołek jest rozważany jako pusty.
3. Wierzchołek jest określany przez wszystkie klasy i ich funkcje przynależności.



Rys. 9.4: Wnioskowanie poprzez rozmyte drzewo decyzyjne, wg [2].

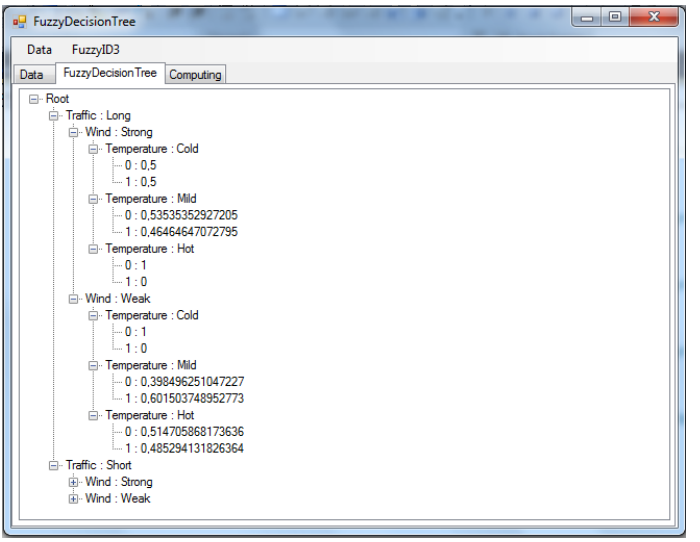
9.6. Wnioskowanie poprzez drzewo decyzyjne

Wnioskowanie w standardowym drzewie decyzyjnym jest wykonywane poprzez schodzenie od korzenia wgłąb drzewa testując każdy atrybut i wybierając odpowiednią gałąź. Procedura ta jest powtarzana aż do momentu dojścia do liścia. Klasa związana z liściem jest wynikiem klasyfikacji.

W rozmytych drzewach decyzyjnych procedura wnioskowania jest bardziej skomplikowana. W tym przypadku zejście wgłąb drzewa zachodzi po wszystkich gałęziach z pewnym prawdopodobieństwem. Obrazuje to przykład przedstawiony na rysunku 9.4. A_i jest atrybutem, który będzie testowany. Natomiast F_{ij} na gałęziach określa zbiór rozmyty z numeryczną wartością funkcji przynależności. Teraz należy określić trzy operacje. Po pierwsze, trzeba przydzielić wartości funkcji przynależności dla każdej ścieżki poprzez iloczyn alternatyw. Po drugie, musi być wyliczona wartości funkcji przynależności całkowitej do danej klasy przez iloczyn prawdopodobieństwa wystąpienia klasy C_k i wartości wyliczonej w kroku pierwszym. Po trzecie, należy dokonać sumy wartości z wszystkich gałęzi otrzymując całkowite prawdopodobieństwo przynależności do danej klasy. Jeśli prawdopodobieństwo przynależności do każdej z klas nie sumuje się do jedności, wartości te należy znormalizować. Dla przykładu z rysunku 9.4 prawdopodobieństwo przynależności do klasy C_1 wynosi 0.69, a do klasy C_2 równe jest 0.31.

9.7. Implementacja rozmytego drzewa decyzyjnego

Program tworzący i korzystający z rozmytych drzew decyzyjnych zaimplementowano w języku C#, korzystającego z platformy .Net. Aplikacja pozwala na generowania drzewa posługując się omówionym wcześniej rozmytym algorytmem ID3 9.5. Dodatkowo dzięki graficznemu interfejsowi użytkownika można edytować aktualnie używanego zbioru uczącego oraz wnioskować za pomocą FDT (rysunek 9.5).



Rys. 9.5: Aplikacja do budowy drzew decyzyjnych (z wyświetlonym przykładowym drzewem).

Do zamodelowania logiki rozmytej w programie posłużono się biblioteką AForge [3]. Biblioteka ta dostarcza gotowy zestaw narzędzi do tworzenia i wnioskowania na zbiorach rozmytych. Takie rozwiązanie w znacznym stopniu przyspieszyło prace implementacyjne.

Literatura

[1] C.Z. Janikow. Fuzzy decision trees: issues and methods. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(1):1 –14, feb 1998.

[2] M. Umanol, H. Okamoto, I. Hatono, H. Tamura, F. Kawachi, S. Umedzu, and J. Kinoshita. Fuzzy decision trees by fuzzy ID3 algorithm and its application to diagnosis systems. In *Proceedings of the Third IEEE Conference on Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence.*, volume 3, pages 2113 –2118, jun 1994.

[3] A. Kirillov. Aforge.net framework. `http://www.aforgenet.com/framework/`.

Od redaktora i wydawcy

Wdrażanie innowacyjnych rozwiązań, w tym technologii informacyjnych, to nie tylko narzędzie, ale również środek do pobudzania gospodarki i odkrywania coraz to nowych obszarów dla naukowej eksploracji. Fakt ten zauważyli nie tylko badacze, ale również politycy, którzy w pakiecie swoich haseł zaczęli zamieszczać takie terminy, jak: społeczeństwo informacyjne czy też społeczeństwo oparte na wiedzy. Wnikliwy obserwator może zauważyć w tym pewien paradoks: z jednej strony zastosowanie nowoczesnych technologii pozwoliło rozwiązać problemy do tej pory nierozwiązywalne, z drugiej jednak strony zrodziło kolejne problemy, być może jeszcze trudniejsze do rozwiązania.

Choć wprowadzenia komputerów i technologii mobilnych są uznawane za milowe kroki w historii rozwoju ludzkiej cywilizacji, można je w pewnym sensie porównać do otwarcia puszki Pandory. Być może nie jest to najlepsze porównanie, oddaje jednak istotę zjawiska polegającego na uzależnieniu się od komputerów całych społeczeństw, lawinowym przyroście zasobów cyfrowych oraz powstaniu chaosu informacyjnego. Ale człowiek nie byłby człowiekiem, gdyby nie podjął działań zaradczych. Stąd w jednostkach naukowych i ośrodkach badawczych od lat są prowadzone prace nad wykorzystaniem komputerów nie tylko do przechowywania danych, ale również do automatycznego ich przetwarzania i rozwiązywania złożonych problemów, zdefiniowanych na różnych poziomach abstrakcji. Działania te wykraczają poza samą implementację algorytmów ekstrahujących wartości parametrów otaczającego nas świata. Materializują się na pograniczu sztucznej inteligencji i inteligencji istot żywych, tworząc pomost pomiędzy czymś, co jesteśmy w stanie sami przeanalizować, a czymś, co umyka naszym zmysłom z powodu wielkiej złożoności albo szczegółowości. Ich dziedzinę można nazwać „komputerowym przetwarzaniem wiedzy".

W niniejszej książce zebrano opracowania studentów II roku studiów magisterskich Wydziału Elektroniki Politechniki Wrocławskiej, kierunku Automatyka i robotyka, specjalności Robotyka, wykonane w ramach prowadzonego przeze mnie kursu ***Komputerowe przetwarzanie wiedzy***, w semestrze letnim roku akademickiego 2011/2012. Prace te obrazują wybrane aspekty wykorzystania komputerów do przetwarzania danych i informacji oraz pozyskiwania wiedzy. Dokładniej mówiąc, są one relacją z wykonanych projektów, zawierającą opis wybranych problemów oraz sposobów ich rozwiązania. Zakres tematyczny opracowań można zawrzeć w następującym zestawieniu:

- –Modelowanie (sztucznej komórki, tłumu)
- –Przetwarzanie języka naturalnego (na potrzeby interfejsu wyszukiwania)
- –Walidacja danych i modeli (w kontekście generowania profili modeli informacyjnych)
- –Ontologie (w podejściu praktycznym)
- –Analiza danych sensorycznych (niwelacja ich niepewności i niepełności)
- –Detekcja tekstu (pozyskiwanie wiedzy z materiałów historycznych)
- –Drzewa decyzyjne (w powiązaniu z logiką rozmytą)

Mam nadzieję, że lektura tych opracowań okaże się interesująca dla czytelnika.

Tomasz Kubik
Wrocław, październik 2012

ISBN 978-83-930823-3-9
9 788393 082339