KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2013/2014
pod redakcją Tomasza Kubika

KOMPUTEROWE PRZETWARZANIE WIEDZY

Kolekcja prac 2013/2014
pod redakcją Tomasza Kubika

Skład komputerowy, projekt okładki
Tomasz Kubik

ISBN 978-83-930823-6-0

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

Piąty tom z cyklu *Komputerowe przetwarzanie wiedzy, Kolekcja prac 2013/2014* zawiera wyniki projektów wykonanych pod kierunkiem dra inż. Tomasza Kubika w ramach kursu *Komputerowe przetwarzanie wiedzy* przez studentów studiów magisterskich Wydziału Elektroniki Politechniki Wrocławskiej, kierunku Automatyka i robotyka, specjalności Robotyka. Projekty zrealizowano w semestrze letnim roku akademickiego 2013/2014. Podobnie jak w latach ubiegłych, celem *Projektu* było przedstawienie wybranych metod i narzędzi do przetwarzania wiedzy oraz ich implementacji programowych służących do rozwiązania konkretnych zadań praktycznych. Niniejsza *Kolekcja* zawiera 128 stron druku i składa się z jedenastu rozdziałów, których tytuły zostały wymienione poniżej:

1. P. Bogner, M. Dmochowski: *Zarządzanie danymi nieustrukturyzowanymi*
2. M. Nowak, G. Maj: *Audyt w systemach informatycznych*
3. J. Bihun, A. Aniszczyk: *Regułowe wsparcie semantycznego wnioskowania*
4. B. Kochanowski, M. Krupop: *Rozproszone bazy wiedzy*
5. A. Klama, M. Kotyla: *Interfejsy komunikacji człowiek – komputer*
6. M. Niestrój, Ł. Pawlik: *Gry analogowe z komputerem*
7. M. Szymański, V. Tsiselskyi: *Środowisko do testowania interfejsu haptycznego*
8. W. Domski, W. Górniak, D. Kwaśnik: *Wykorzystanie algorytmu genetycznego w rozwiązywaniu problemu spełniania ograniczeń*
9. M. Kolasa, M. Patro: *Projekt języka dziedzinowego zanurzonego w składni LaTeX*
10. F. Sobczak, M. Błędowski: *Wykorzystanie logiki temporalnej do weryfikacji modelu*
11. D. Rolla, R. Tomaszewski: *Wizualizacja zjawisk temporalnych*

Dla scharakteryzowania zawartości tomu posłużymy się metodą *pars pro toto* ograniczając się do rozdziałów 6, 8 i 9.

6. *Gry analogowe z komputerem*: Termin ten oznacza grę z komputerem prowadzoną w świecie rzeczywistym, nie wirtualnym. W rozdziale podano przegląd narzędzi informatycznych wykorzystywanych przy tego rodzaju grze, a następnie przedstawiono własną koncepcję gry w warcaby na rzeczywistej planszy. Do zadań komputera należy śledzenie przebiegu gry, podejmowanie decyzji

co do kolejnych ruchów, a także pilnowanie przestrzegania reguł gry. Z tego względu, w przeglądzie szczególną uwagę poświęcono algorytmom przetwarzania obrazów wizyjnych, metodom sztucznej inteligencji odnoszącym się do gier logicznych i interfejsom komputer-człowiek. Opracowano aplikację, która umożliwia grę w warcaby z komputerem przy wykorzystaniu wzroku i słuchu człowieka.

8. *Wykorzystanie algorytmu genetycznego w rozwiązywaniu problemu spełniania ograniczeń*: Problem spełnienia ograniczeń jest problemem decyzyjnym, w którym wynikowa decyzja uwzględnia ograniczenia nałożone na strukturę przestrzeni decyzyjnej. Rozdział dotyczy zadania zagospodarowania przestrzeni wystawowej polegającego na rozmieszczeniu elementów typu pawilony handlowe w dopuszczalnym obszarze przestrzeni wystawowej. Jako narzędzie do rozwiązania zadania zaproponowano algorytm genetycznego i przedstawiono jego implementację komputerową.
9. *Projekt języka dziedzinowego zanurzonego w składni LaTeX*: Języki dziedzinowe stanowią rodzaj języków programowania wyspecjalizowanych w konkretnej dziedzinie. Rozdział omawia zagadnienia związane z tworzeniem języków dziedzinowych, a jego kulminację stanowi własny projekt języka dziedzinowego opartego na składni LaTeXa i jego prekompilatora, przeznaczonego do wspomagania tworzenia grafiki robotów.

Niniejsza *Kolekcja* wyróżnia się bogatą treścią, przemyślaną koncepcją i staranną redakcją. Z pewnością zasługuje na zainteresowanie Czytelników pragnących uzyskać rozeznanie w zakresie zadań, metod i zastosowań komputerowego przetwarzania wiedzy.

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

ROZDZIAŁ 1

ZARZĄDZANIE DANYMI NIEUSTRUKTURYZOWANYMI

P. Bogner, M. Dmochowski

1.1. Wprowadzenie

Z terminem „dane nieustrukturyzowane" chyba najczęściej można spotkać się przy opisach systemów informatycznych. Służy on do określania tej części zasobów, których postać nie podlega jakimś szczególnym regułom ułatwiającym ich automatyczne przetwarzanie. Zalicza się do nich treści wyrażone w języku naturalnym, obrazy i dźwięki itp. – a więc wszelkie przekazy zawierające dane lub informacje w postaci niejawnej bądź ukrytej w jakimś trudnym do zinterpretowania przez komputery kontekście. Dane nieustrukturyzowane leżą na przeciwnym biegunie do danych strukturalnych, które posiadają sformalizowaną postać, dobrze zdefiniowaną składnię oraz znaczenie zrozumiałe dla maszyn.

Zarządzanie danymi nieustrukturyzowanymi jest ważnym i złożonym zagadnieniem. Wyróżnia się w nim indeksowanie oraz kategoryzację – czynności, pozwalające na zbudowanie takiego zbioru danych, który będzie łatwiejszy do przetworzenia przez komputerowy niż zbiór danych oryginalnych. Do zarządzania danymi nieustrukturyzowanymi dość często są wykorzystywane metody bazujące na użyciu klasyfikatorów oraz sztucznych sieci neuronowych. Jedną z klasycznych metod z tego obszaru jest naiwny klasyfikator bayesowski, stosowany z powodzeniem do segregowania tekstów. Do popularnych jego zastosowań należy odfiltrowywanie niepożądanych wiadomości w systemach antyspamowych. Klasyfikator ten może być wykorzystany w innych celach, na przykład do kategoryzacji tematycznej zeskanowanych i rozpoznanych (metodami OCR) tekstów podczas cyfryzacji księgozbiorów.

W niniejszym rozdziale opisano rozwiązanie, w którym naiwny klasyfikator bayesowski posłużył jako element systemu klasyfikującego teksty według odgórnie ustalonego zestawu kategorii. Po przygotowaniu zestawu uczącego (liczba tekstów w zbiorze uczącym kształtowała się na poziomie 900) skuteczność klasyfikacji badano na próbie artykułów zawierających aktualne informacje publikowane na stronach portali internetowych za pośrednictwem kanałów RSS.

1.2. Podstawy teoretyczne

1.2.1. Problem klasyfikacji

Definicja 1 (Klasyfikacja [1]) *Klasyfikacja to problem estymacji nieznanej funkcji celu* $\Phi : \mathscr{D} \times \mathscr{C} \rightarrow \{0,1\}$*, gdzie* $\mathscr{D}$ *jest zbiorem klasyfikowanych obiektów, a* $\mathscr{C} = \{c_1, \ldots, c_{|\mathscr{C}|}\}$ *jest ustalonym zbiorem klas.*

Upraszczając powyższą definicję można powiedzieć, że klasyfikacja obiektu polega na określeniu, czy dany element należy do podanej klasy, czy nie. Definicja ta jest ogólna i dopuszcza przypadki, gdy element należy do więcej niż jednej klasy lub nie należy do żadnej z klas.

Ogólny problem klasyfikacji można sprowadzić do problemu klasyfikacji tekstu, w którym zbiór klasyfikowanych obiektów zdefiniowany jest jako zbiór dokumentów tekstowych o cechach zależnych od ich zawartości (zależnych od występowania konkretnych wyrazów). Ponadto można przyjąć, co uczyniono w niniejszym rozdziale, że cechy te są niezależne od pozycji danego słowa w tekście oraz że jeden dokument może należeć dokładnie do jednej klasy. Przy tych założeniach definicja 1 przyjmuje postać

Definicja 2 (Klasyfikacja dokumentów tekstowych) *Klasyfikacja to problem estymacji nieznanej funkcji celu* $\Phi : \mathscr{D} \rightarrow \mathscr{C}$*, gdzie* $\mathscr{D}$ *jest zbiorem klasyfikowanych dokumentów, a* $\mathscr{C} = \{c_1, \ldots, c_{|\mathscr{C}|}\}$ *jest ustalonym zbiorem klas.*

1.2.2. Naiwny klasyfikator Bayesa

Przy użyciu naiwnego klasyfikatora Bayesa klasa c_{out}, do której jest zaklasyfikowany dokument d, jest wyznaczana w następujący sposób [2]:

$$c_{out} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c).$$

Reprezentując dokument tekstowy d jako zbiór cech $\{x_1, x_2, \ldots, x_m\}$ (zbiór słów znajdujące się w jego treści)

$$c_{out} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_m|c)P(c) \tag{1.1}$$

oraz zakładając, że:

- pozycja wyrazu w tekście nie ma znaczenia,
- prawdopodobieństwo wystąpienia danego słowa nie zależy od klasy, do której należy dokument [3],

równanie (1.1) przyjmuje postać:

$$c_{out} = \underset{c \in C}{\operatorname{argmax}} \left[P(x_1|c)P(x_2|c)\cdots P(x_m|c)\right] P(c) \tag{1.2}$$

1.2.3. Metoda klasyfikacji

Aby obliczyć elementy równania (1.2) można posłużyć się następującymi estymatorami:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N}, \qquad \hat{P}(w_i|c_j) = \frac{N_{w_i,c_j}}{\sum_{w \in V} N_{w,c_j}}, \tag{1.3}$$

gdzie N –liczba wszystkich tekstów w zbiorze uczącym, N_{c_j} – liczba tekstów klasy c_j w zbiorze uczącym, N_{w_i,c_j} – liczba wystąpień słowa w_i w tekstach klasy c_j, $\sum_{w \in V} N_{w,c_j}$ – liczba wyrazów we wszystkich tekstach z klasy c_j, V – słownik. Innymi słowy, $\hat{P}(w_i|c_j)$ można obliczyć poprzez obliczenie częstotliwości wystąpień słowa w_i we wszystkich tekstach klasy c_j.

Widać jednak, że taki sposób estymacji częstości występowania słów zwraca 0 dla wyrazów, które nie znalazły się w zbiorze uczącym. W takim wypadku klasa nigdy nie zostanie wybrana jako maksymalizująca wyrażenie w równaniu (1.2), ponieważ iloczyn $\prod_i P(x_i|c)$ będzie równy 0. Aby temu zaradzić można zastosować wygładzanie Laplace'a [2]:

$$\hat{P}(w_i|c_j) = \frac{N_{w_i,c_j} + 1}{\sum_{w \in V} N_{w,c_j} + |V|}. \tag{1.4}$$

Dzięki temu zabiegowi żaden z czynników w równaniu (1.2) nie będzie zerowy.

Kolejnym problemem może być obliczenie $\hat{P}(w_u|c_j)$ gdy w_u jest słowem niewystępującym w słowniku V. W takiej sytuacji prawdopodobieństwo to można otrzymać poprzez sztuczne dodanie do słownika jednego słowa, które ma liczność wystąpień 0. Na podstawie równania (1.4) otrzymujemy:

$$\hat{P}(w_u|c_j) = \frac{1}{\sum_{w \in V} N_{w,c_j} + |V + 1|}.$$

Algorytm uczenia

1. Ze zbioru tekstów uczących wyodrębnij słownik.
2. Oblicz $\hat{P}(c_j)$ dla każdej z klas według równania (1.3).
3. Oblicz $\hat{P}(w_i|c_j)$ dla każdego słowa w słowniku według równań (1.3) i (1.4).

1.2.4. Przetwarzanie tekstu

W języku polskim istnieje bogata fleksja. Stąd wyrazy o tym samym znaczeniu mogą występować pod wieloma postaciami. Jest to sytuacja, która utrudnia automatyczne przetwarzanie języka naturalnego. Można temu zaradzić poddając tekst procedurze hasłowania. Procedura ta składa się z dwóch etapów: analizy morfologicznej i ujednoznaczniania morfologicznego.

W pierwszym etapie dąży się do określenia wszystkich form podstawowych, od których może pochodzić przetwarzane słowo. W tym miejscu nie jest brany pod uwagę kontekst wystąpienia wyrazu. Oprócz tego analiza morfologiczna

obejmuje rozpoznanie postaci fleksyjnej badanego słowa, jednak informacja ta nie będzie wykorzystywana w opisywanym rozwiązaniu. Można zatem powiedzieć, że będziemy mieli do czynienia z ograniczoną analizą morfologiczną.

Etap drugi, czyli ujednoznaczniania, polega na wybraniu formy podstawowej, która jest reprezentowana przez badane słowo na podstawie kontekstu jego wystąpienia. Po takiej operacji otrzymuje się tekst złożony z podstawowych form wyrazów pierwotnie w nim występujących, który nadaje się do dalszego przetwarzania.

1.3. Implementacja klasyfikatora

1.3.1. Architektura rozwiązania

Program spełniający rolę klasyfikatora napisano w języku C++, dzieląc go na kilka istotnych części odpowiednio do realizowanego w nich przetwarzania.

Wczytanie bazy wiedzy Pierwszą czynnością niezbędną do wykonania w trakcie działania programu jest wczytanie uprzednio przygotowanej bazy wiedzy. W obecnej implementacji baza ta składa się z pliku zawierającego informację o liczbie kategorii i ich nazwach oraz z folderów zawierających pliki tekstowe, których zawartość stanowią ciągi form podstawowych słów, będących podstawą klasyfikacji. Domyślna baza wiedzy zawiera pięć kategorii: świat, Polska, sport, muzyka i film oraz technologie. Liczba folderów powinna być równa liczbie kategorii. Wykorzystany jako element bazy wiedzy przykładowy ciąg słów może być uzyskany jako rezultat działania pozostałych modułów programu, to jest pobrany z Internetu i przetworzony do ciągu form podstawowych. Po wczytaniu wszystkich plików klasyfikator przydziela poszczególne formy podstawowe do odpowiedniej kategorii.
Dzięki zaimplementowaniu wczytywania bazy wiedzy z plików tekstowych można utworzyć własną bazę wiedzy z wybraną liczbą kategorii.

Pozyskiwanie tekstów do sklasyfikowania Klasyfikowane teksty pozyskiwano z serwisów informacyjnych. Moduł programu odpowiedzialny za pobieranie tekstów z Internetu za pośrednictwem kanałów RSS to FeedReader (`https://code.google.com/p/feed-reader-lib/`), którego autorem jest Yoav Aviram. Jest to biblioteka napisana w języku C++ z możliwością interpretacji formatów RSS 1.0, RSS 2.0, ATOM 0.3, ATOM 1.0 oraz RDF. Istnieje także możliwość rozszerzenia tej puli o własne formaty, gdyż FeedReader przechowuje dane o każdym obsługiwanym formacie w postaci pliku XSL (*eXtensible Stylesheet Language*) definiującym postać wczytywanego źródła XML. Treści uzyskane za pośrednictwem FeedReader są w postaci przygotowanej do przetwarzania w następnym kroku działania programu.

Wstępne przetworzenie tekstu W języku polskim istnieje wiele form fleksyjnych, dlatego też niezbędne jest wstępne przetworzenie tekstu przed rozpoczęciem procedury klasyfikacji. W tym celu wykorzystywana jest biblioteka programu TaKIPI [4], który ustala dla wyrazów w zadanym tekście ich

opis morfosyntaktyczny, a następnie określa właściwą interpretację poszczególnych wyrazów w zależności od kontekstu ich wystąpienia. Po dokonaniu tych czynności program sprawdza, jaka jest forma podstawowa dla poszczególnych słów, co umożliwia przeprowadzenie klasyfikacji tekstu z wykorzystaniem bazy słów kluczowych zawierającej wyłącznie formy podstawowe słów. Rezultat działania modułu TaKIPI to plik XML, który zawiera informacje o dokonanej interpretacji poszczególnych słów tekstu.

Przetworzenie pliku XML Wśród zawartych w uzyskanym pliku XML danych znajdują się informacje o najbardziej prawdopodobnych formach podstawowych wszystkich słów z zadanego tekstu. Można je łatwo wyodrębnić ponieważ zawierający je znacznik XML posiada dodatkowy atrybut, nieobecny w przypadku znaczników innych zgadywanych form podstawowych. Czynność wyodrębnienia dokonuje parser XML utworzony przy pomocy biblioteki Xerces-C++. Rezultatem działania parsera jest lista słów stanowiących najbardziej prawdopodobne formy podstawowe słów z zadanego tekstu.

Klasyfikacja tekstu Uzyskana w poprzednim kroku lista form podstawowych jest przekazywana do klasyfikacji. Opis działania klasyfikatora przedstawiono w podrozdziale 1.2. Rezultatem jego działania jest wskazanie kategorii, do której rozważany tekst prawdopodobnie należy.

Poszerzanie bazy wiedzy Czynnością wykonywaną poza normalnym tokiem działania programu jest poszerzanie bazy wiedzy. Proces ten jest całkowicie zautomatyzowany dla domyślnej bazy wiedzy. Z predefiniowanych źródeł wiadomości w formacie RSS pobierane są teksty, które już wcześniej zostały zaklasyfikowane do odpowiedniej kategorii przez ich twórców.

1.3.2. Wykorzystane narzędzia i biblioteki

W projekcie wykorzystano liczne zewnętrzne biblioteki i narzędzia. Poniżej przedstawiono pełną ich listę.

- Biblioteka do pobierania tekstów ze źródeł w formacie RSS – FeedReader. Koszysta ona z biblioteki cURL (`http://curl.haxx.se/`), obsługującej transfer danych poprzez rozmaite protokoły internetowe; Xalan-C++ (`http://xml.apache.org/xalan-c/`), odpowiadającej za obsługę plików w formacie XSL; a także Xerces-C++ (`http://xerces.apache.org/xerces-c/`), odpowiadającej za interpretację plików XML.
- Biblioteka do uzyskiwania form podstawowych wyrazów w zadanym tekście – TaKIPI [4]. Korzysta ona z lekko zmodyfikowanej bazy danych analizatora morfologicznego Morfeusz Polimorf (`http://sgjp.pl/morfeusz/index.html`) oraz z bibliotek ANTLR (`http://www.antlr.org/`) (*ANother Tool for Language Recognition*) i ICU (`http://site.icu-project.org/`) (*International Components for Unicode*).
- Biblioteki dla języka C++ Boost (`http://www.boost.org/`) (podstawowe oraz biblioteka regex) w wersji 1.55.

- Biblioteki Qt (`http://qt-project.org/`) w wersji 5.3.0 oraz (przy kompilacji) program `qmake` w wersji 3.0.

Wszystkie wymienione biblioteki z wyjątkiem Boost oraz Qt zostały załączone do archiwum z kodem źródłowym, co znacznie zwiększa przenośność aplikacji. Ich licencje pozwalają na dokonanie takiej czynności.

1.3.3. Wyniki klasyfikacji

Do testów użyto bazy wiedzy składającej się z 893 dokumentów pochodzących z popularnych polskich portali internetowych. Każdy z dokumentów należał do jednej z pięciu kategorii: *świat* (155 dokumentów), *Polska* (179 dokumentów), *sport* (215 dokumentów), *muzyka i film* (165 dokumentów), *technologie* (179 dokumentów). Działanie zaimplementowanego klasyfikatora zbadano poddając klasyfikacji łącznie 335 wiadomości. Ich rozkład na kategorie był następujący: *świat* (73 dokumenty), *Polska* (72 dokumenty), *sport* (85 dokumentów), *muzyka i film* (52 dokumenty), *technologie* (50 dokumentów). Wyniki klasyfikacji zebrano w tabeli 1.1.

Tab. 1.1: Macierz wyników klasyfikacji.

wynik \kat. rzeczywista	*świat*	*Polska*	*sport*	*muzyka i film*	*technologie*
świat	44	0	0	0	0
Polska	21	54	0	3	0
sport	3	5	86	3	1
muzyka i film	1	0	0	43	0
technologie	4	13	2	3	49

Jak można zauważyć, spośród wszystkich testowanych wiadomości 276 zostało poprawnie rozpoznanych. Analizując zebrane wyniki można określić kilka wskaźników poprawności działania algorytmu. Oznaczając elementy macierzy klasyfikacji przez a_{ij}, gdzie indeksy i oraz j oznaczają, odpowiednio, indeks wiersza wynikowej klasyfikacji oraz indeks kolumny odpowiadającej rzeczywistej klasie dokumentu możemy zdefiniować:

- ogólny odsetek poprawnych klasyfikacji, obliczany jako stosunek liczby poprawnie zaklasyfikowanych dokumentów do liczby wszystkich klasyfikowanych dokumentów:
$$\frac{\sum_{i=1}^{n} a_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}} \text{ oraz} \tag{1.5}$$
- odsetek poprawnych klasyfikacji w obrębie danej kategorii, obliczany analogicznie jak wyżej, przy czym wszystkie rozważane dokumenty muszą pochodzić z ustalonej klasy k:
$$\frac{a_{kk}}{\sum_{i=1}^{n} a_{ik}}. \tag{1.6}$$

Wskaźniki te osiągnęły następujące wartości:

- ogólny odsetek poprawnych klasyfikacji: 82%;
- odsetek poprawnych klasyfikacji dla kategorii *świat*: 60%;
- odsetek poprawnych klasyfikacji dla kategorii *Polska*: 75%;
- odsetek poprawnych klasyfikacji dla kategorii *sport*: 98%;
- odsetek poprawnych klasyfikacji dla kategorii *muzyka i film*: 83%;
- odsetek poprawnych klasyfikacji dla kategorii *technologie*: 98%.

Najlepiej rozpoznawanymi kategoriami były: *sport* i *technologie*, natomiast kategorią najgorzej rozpoznawaną był *świat*. Co ciekawe, *sport* był mylony jedynie z *technologiami*, a *technologie* ze *sportem*. Dla żadnych pozostałych dwóch kategorii nie zachodzi taka relacja. Jednocześnie były to najlepiej rozpoznawane kategorie.

Największe problemy pojawiły się przy dwóch kategoriach: *świat* i *Polska*. Ponieważ *Sport*, *muzyka i film* oraz *technologie* dotyczą wąskiego grona tematów, często pojawia się w nich specyficzne słownictwo. W przypadku wiadomości ze świata lub kraju tematyka może być zróżnicowana: polityka, gospodarka, relacje międzynarodowe, wypadki itd. W związku z tym rozróżnienie tych dwóch kategorii może być niejasne, gdyż głównie opiera się ono na miejscu wystąpienia opisywanej sytuacji.

1.4. Wnioski

Na podstawie przeprowadzonych badań wysnuto następujące konkluzje.

- Naiwny klasyfikator Bayesowski jest jednym z najprostszych narzędzi do klasyfikacji. Jego główną zaletą jest to, że nie wymaga on nastawy żadnych parametrów, w związku z czym może być użyty bez żadnej wstępnej wiedzy o klasyfikowanych obiektach, co więcej, może posłużyć do pozyskania takich informacji.
- Klasyfikator ten może być użyty do każdego rodzaju tekstu, niezależnie, czy słowa, które się w nim pojawiają występują w używanym słowniku, czy nie.
- W przypadku języka polskiego, ze względu na mnogość form fleksyjnych, konieczne jest wstępne przetworzenie tekstu w celu uzyskania podstawowych form wyrazów. Taka transformacja nie wprowadza zmian pod względem semantycznym, natomiast pozwala zredukować objętość słownika oraz rozpoznawać różne formy fleksyjne tego samego wyrazu jako jedną pozycję bazy wiedzy, co jest pożądanym efektem.
- Działanie klasyfikatora znacząco różniło się w zależności od kategorii, która była rozpoznawana. Poprawność klasyfikacji wahała się od 60% do 98%. Związek z tym miała tematyka kategorii. Kategorie wąskie (w których często pojawia się specyficzne słownictwo) były klasyfikowane niemal bezbłędnie. Kategorie ogólne, zawierające informacje na wiele różnych tematów, były rozpoznawane gorzej, jednak w dalszym ciągu uzyskiwane wyniki są lepsze, niż przy losowym przypisywaniu kategorii według rozkładu jednostajnego.

Literatura

[1] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[2] W. Dai, G.-R. Xue, Q. Yang, Y. Yu. Transferring naive bayes classifiers for text classification. *Proceedings of the National Conference on Artificial Intelligence. Vol. 22*, number 1, strony 540–545. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press, 2007.

[3] B. Pang, L. Lee, S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, wolumen 10, strony 79–86. Association for Computational Linguistics, 2002.

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

ROZDZIAŁ 2

AUDYT W SYSTEMACH INFORMATYCZNYCH

M. Nowak, G. Maj

Do istotniejszych obszarów zarządzania w różnego rodzaju projektach informatycznych należy zarządzanie jakością. O jakości mówi się podczas oceny sposobu wykonania produktów oraz sprawdzaniu ich zgodności ze specyfikacją. Z terminem tym wiąże się również przebieg procesu produkcyjnego oraz inne aspekty związane z wytwarzaniem dzieł i artefaktów. Zapewnienie odpowiedniej jakości to prawdziwe wyzwanie dla dużych firm o zasięgu globalnym, zatrudniających tysiące programistów z całego świata.

W niniejszym rozdziale podjęto próbę opracowania metody oceny jakości i efektywności pracy programistów. Temat ten jest dość złożony i wymaga dobrej znajomości specyfiki zachodzących procesów oraz cyklu życia powstających produktów. Aby przybliżyć czytelnikowi związane z nim zagadnienia opisano dostępne na rynku rozwiązania oraz przedstawiono własną propozycję prostego narzędzia do kontroli efektywności pracy programistów.

2.1. Dostępne systemy pomiaru pracy

Na rynku istnieją różne systemy informatyczne służące do wspomagania i monitorowania pracy programistów. Dostarczają one narzędzi umożliwiających prowadzenie pracy grupowej oraz raportowanie aktywności jej uczestników. Poniżej przedstawione zostały najpopularniejsze systemy, w tym rozbudowane i szeroko wykorzystywane systemy o otwartym kodzie źródłowym.

2.1.1. Systemy kontroli wersji

Głównym zadaniem systemów kontroli wersji jest łączenie wyników pracy wielu programistów poprzez zapamiętywanie kolejnych wersji powstającego kodu i tworzenie historii zmian. Systemy te pozwalają monitorować postępy prac poszczególnych programistów. Podstawowe informacje, które można dzięki nim uzyskać to:

- jaki użytkownik i kiedy wprowadzał zmiany,
- jakie zmiany zostały wprowadzone:

– dokładne różnice między zmienionymi plikami,
– skrócone statystyki,
– sumy kontrolne poszczególnych zmian.

Informacje te można filtrować i prezentować w różnych widokach:

- kontrola jednego pracownika,
- kontrola danego okresu zmian,
- kontrola historii tworzenia i łączenia gałęzi.

Przykładami systemów kontroli wersji są najbardziej popularne i darmowe systemy: Git (`http://git-scm.com/`), Svn (`http://subversion.apache.org/`) czy Mercurial (`http://mercurial.selenic.com/`).

Git jest rozproszonym systemem kontroli wersji, udostępniany jako wolne oprogramowanie na licencji GNU GPL2. Git powstał jako alternatywa dla zamkniętego systemu BitKeeper w kwietniu 2005 roku do rozwijania projektu Linux.

Svn jest systemem kontroli wersji udostępnionym jako wolne oprogramowanie na licencji Apache. Został stworzony z myślą o zastąpieniu systemu CVS.

Mercurial jest, podobnie jak Git, rozproszonym systemem kontroli wersji. Został napisany w tym samym czasie co Git, również w celu pomocy do rozwijania projektu Linux.

2.1.2. Systemy zarządzania projektem

Systemy zarządzania projektem są platformami wspomagającymi podział prac i weryfikację postępów prac. Dają pewien obraz wykonanych pracy przez każdego członka zespołu. Do najpopularniejszych systemów tego typu należą: Redmine (`http://www.redmine.org`), Jira (`https://www.atlassian.com/software/jira`), GitHub (`https://github.com`).

Redmine jest internetowym narzędziem do zarządzania projektem i szukania błędów o wolnym i otwartym kodzie. Udostępnia kalendarz i wykres Ganta oraz dostarcza wizualnego wsparcia by łatwiej kontrolować kończące się etapy projektu. Wspiera różne systemy kontroli wersji i umożliwia kontrole podziału zadań w grupie. Narzędzie Redmine jest zbudowane z wykorzystaniem technologii Ruby on Rails, co zapewnia mu przenośność.

Jira jest zamkniętym oprogramowaniem stworzonym przez firmę Atlassian, wspomagającym zarządzanie projektem i śledzeniu błędów. Jest wykorzystywana w wielu projektach, m.in. Fedora Commons, Skype czy JBoss. Zbudowano go z wykorzystaniem technologii Java, co zapewnia mu przenośność pomiędzy różnymi platformami. Firma Atlassian udostępnia program JIRA nieodpłatnie do użytku *non-profit*.

GitHub jest serwisem hostingowym dla projektów wykorzystujących system kontroli wersji Git. Udostępnia darmowe i otwarte repozytoria oraz płatne repozytoria zamknięte. Oprócz podstawowego zadania jakim jest przechowywanie plików projektu wspomaga kontrolę tworzenia oprogramowania poprzez dodatkowe narzędzia nieobecne w systemie Git (np. bugtracker, forki repozytoriów, graficzne statystyki czy wiki z dokumentacją). Udostępnia mechanizmy dostępu i zarządzania uprawnieniami co umożliwia pracę grupową.

2.1.3. Systemy do statycznej analizy kodu

Platformami dedykowanymi do rejestrowania zmian w samym kodzie projektu, wybiegającymi poza odnotowywanie dużych zmian w strukturze całego projektu, są systemy do statycznej analizy kodu. Ich cechą jest działanie równolegle do procesu tworzenia kodu, a nie tylko obserwacja zmian etapami. Obecnie powszechnie dostępne są platformy obsługujące wszystkie popularne języki programowania. Przykładami takich systemów są: Klocwork (`http://www.klocwork.com/`), Pylint (`http://www.pylint.org/`), CodeSonar (`http://www.grammatech.com/codesonar`).

Klocwork jest zamkniętym programem przeznaczonym do statycznej analizy kodu, pozwalającym na detekcję błędów strukturalnych kodu i błędów przebiegu programu. Umożliwia pracę z najpopularniejszymi językami C, C++, Java czy C#.

Pylint jest oprogramowaniem o otwartych źródłach kodu, służącym do znajdywania błędów w kodzie i sprawdzania jego jakości. Pylint jest podobny do Pychecker – innego oprogramowania, w porównaniu z którym jest jednak bardziej rozbudowany. Wspiera między innymi sprawdzanie długości linii, zgodności nazw zmiennych ze standardem kodowania. Umożliwia też tworzenie diagramów UML na bazie stworzonego kodu źródłowego.

CodeSonar jest zamkniętym oprogramowaniem analizującym kod pod względem bezpieczeństwa i poprawności. Wspiera języki programowania C, C++ i Java. Jest wykorzystywany w wielu gałęziach gospodarki, m.in. w NASA czy w firmie Toyota. Oprócz analizy kody umożliwia wizualizację architektury programu i zbiera dane do metryk oceny jakości kodu.

2.2. Sposoby pomiaru jakości pracy programisty

Na przeprowadzenie oceny programisty można patrzeć z różnych perspektyw. Z jednej strony efektem jego pracy jest kod źródłowy, więc podstawowym kryterium oceny powinna być jakość tego kodu. Z drugiej jednak strony w każdym projekcie obecne są pewne ograniczenia czasowe, więc szybkość pracy powinna również być uwzględniona. Uogólniając można powiedzieć, że o umiejętnościach programisty świadczą: jakości kodu oraz organizacja pracy.

Normy dotyczące zarządzania jakością (np. ISO 9000:2008, ISO 15504-4:2005) definiują tzw. kluczowe wskaźniki efektywności (ang. *key performance indicator*,

KPI). Ważną cechą tych wskaźników jest to, że opisują one mierzalne procesy w postaci liczbowej. Daje to menedżerowi obiektywny obraz efektywności danego procesu w przedsiębiorstwie i pozwala na cykliczną kontrolę jakości. Poniżej przedstawiono kilka przykładowych wskaźników, które spełniają warunek mierzalności, więc kwalifikują się do grupy KPI.

2.2.1. Ocena jakości wygenerowanego kodu

Problem oceny jakości kodu mimo wielu dekadach intensywnego rozwoju branży IT wciąż pozostaje (częściowo) nierozwiązany. Wynika to faktu, iż na przestrzeni lat zmieniały się trendy w tworzeniu oprogramowania – programiści stosowali różne paradygmaty programowania (w kolejności chronologicznej: programowanie obiektowe, programowanie strukturalne, programowanie funkcjonalne) a wraz z nimi musiały się zmieniać kryteria oceny. Niektóre bowiem elementy języków były uznawane za wartościowe w jednym podejściu, a w drugim za absolutnie niedopuszczalne. Co prawda istnieją pewne fundamentalne cechy oprogramowania, które są pożądane zawsze, np. zwięzłość kodu, jednak opracowanie obiektywnej metody pomiaru jakości kodu wciąż rodzi trudności.

Wartości służące do pomiaru pewnej własności oprogramowania lub jego specyfikacji nazywane są metrykami oprogramowania. Aby metryka była użyteczna powinna być [1]:

- prosta i możliwa do obliczenia przez komputer,
- przekonująca,
- konsekwentna i obiektywna,
- spójna pod względem użytych jednostek,
- niezależna od języka oprogramowania,
- dająca przydatne informacje .

Metryki można podzielić na statyczne i dynamiczne. Metryki statyczne łączą się ściśle z analizą statyczną kodu – dziedziną inżynierii oprogramowania zajmującą się badaniem struktury kodu źródłowego. Metryki te najbardziej przydatne są dla samych programistów i innych osób bezpośrednio zaangażowanych w proces powstawania oprogramowania. Pozwalają na bieżące śledzenie jakości kodu i zwracanie uwagi na miejsca, które wymagają uproszczenia bądź szczególnie uważnego testowania.

Metryki dynamiczne to wskaźniki abstrahujące od kodu źródłowego. Badają one zachowanie programu po jego uruchomieniu. Są one mocno związane z wymaganiami klienta, stąd dla samego przedsiębiorcy, są to liczby przydatne bardziej do analizy jakości produktu niż pracy jego pracowników.

Poniżej dokonano prezentacji kilku wybranych metryk: linie kodu (ang. *lines of code*, LOC), linie kodu na plik źródłowy (ang. *lines of code per file*, LOCPF), brak spójności metod (ang. *Lack of Cohesion of Methods*, LCOM).

LOC jest chyba najprostszą metryką pozwalającą ocenić rozmiar oprogramowania. Jej wartością jest liczba linii kodu źródłowego (z reguły pomija się linie komentarzy). W specyficznych przepadkach może dawać mylne wyobrażenia

o ocenianym kodzie. Na przykład programy pisane w językach wysokiego poziomu mają znacząco mniej linii kody niż programy pisane w assemblerze. Na wartości tej metryki ma wpływ również styl programowania. Poniżej pokazano dwa przykłady kodu tej samej procedury (sortowania bąbelkowego) napisane w tym samym języku programowania (C++), ale w różnych stylach:

```
// przyklad 1
void bubblesort(std::vector<int>& A)
{
        int temp = 0;

        for (int i = 0; i < A.length()-1; i++)
        {
                for (int j = 0; j < A.length()- 1 - i; j++)
                {
                        if (A[j] > A[j+1])
                        {
                                temp = A[j+1];
                                A[j+1] = A[j];
                                A[j] = temp;
                        }
                }
        }
}

// przyklad 2
void bubblesort(std::vector<int>& A) {
        for (int i = 0; i < A.length()-1; i++)
                for (int j = 0; j < A.length()- 1 - i; j++)
                        if (A[j] > A[j+1])
                                std::swap(A[j], A[j+1]);
}
```

W pierwszym przykładzie stosowane są przeniesienia do następnej linii przy otwieraniu nowych bloków kodu (klamr {}). Oprócz tego, każde wyrażenie warunkowe i pętle są akcentowane nowym otwarciem i zamknięciem klamry, choć składnia języka C++ tego nie wymaga. Dodatkowo, w drugim przykładzie, aby bardziej zwiększyć zwięzłość (i czytelność) kodu, do operacji zamiany wartości dwóch elementów wektora, zastosowano funkcję z biblioteki standardowej, która ma dokładnie takie samo działanie jak kod z przykładu pierwszego.

LOCPF to kolejna dość powszechnie stosowana metryka służąca do oceny liczby linii kodu przypadająca na plik. Metryka ta jest szczególnie ważna ponieważ zapewnia ewaluację projektu programistycznego jako całości. Ważne jest, aby nie wliczać do tej metryki plików konfiguracyjnych ani plików wynikowych procesu budowania aplikacji. Do wyliczenia LOCPF można wykorzystać metrykę LOC:

$$\frac{1}{N}\sum_{f=1}^{N} LOC(file[f]) \quad f : file[f] \in source\_files$$

LCOM to metryka służąca do wyliczania wartości mówiącej o braku spójności metod. Spójność danej metody oznacza, że ma ona jedną kompetencję i nie wchodzi w kompetencje innych metod. Dzięki zachowaniu spójności metod powstający kod jest dobrze refaktoryzowalny, a przez to łatwiej utrzymywalny, testowalny oraz czytelny.

O spójności metod (a także klas i innych obiektów języka) mówi m.in. zasada pojedynczej odpowiedzialności (ang. *Single Responsibility Principle*, SRP). Jest to jedna z pięciu zasad tworzenia architektury oprogramowania w paradygmacie obiektowym, nazywanych SOLID (akronim od ang. *Single responsibility*, *Open-closed*, *Liskov substitution*, *Interface segregation* oraz *Dependency inversion*). Przestrzeganie tych zasad pozwala tworzyć kod, w którym w prosty sposób można wprowadzać modyfikacje i reagować na zmianę wymagań. Architektura, która przestrzega zasad SOLID cechuje się dużą stabilnością oprogramowania i elastycznym projektowaniem [2].

Pożądane jest, aby wartość LCOM dla ocenianego kodu była jak najmniejsza. Istnieje kilka definicji tej metryki. W jednej z nich [3] dla zestawu danych:

- m – liczba metod w klasie,
- a – liczba atrybutów w klasie,
- $mA(a)$ – liczba metod mających dostęp do atrybutu a,
- $sum(mA)$ – suma wartości mA po wszystkich atrybutach,

brak spójności metod obliczany jest z następującego wzoru:

$$LCOM2 = 1 - \frac{sum(mA)}{m \times a}$$

2.2.2. Ocena efektywności pracy

Oceniając efektywności pracy programisty powszechnie wykorzystuje się następujące parametry:

- czas pracy pracownika,
- ilość rozwiązanych zadań,
- efektywność mierzona stosunkiem przewidywanych do wykonanych zadań,
- ocena wygenerowanego kodu: ilość wygenerowanego kodu, rozkład kodu w plikach, przejrzystość kodu, jakość testów, jakość dokumentacji, nazewnictwo funkcji i klas, analiza statyczna kodu.

2.3. Programowe wsparcie oceny efektywości pracy programistów

Poniżej przedstawiono propozycję narzędzia wpierającego ocenę efektywności pracy programistów. Zaprojektowane je jako wtyczkę do popularnego środowiska programistycznego Eclipse. Narzędzie to umożliwia analizę kodu wybranego projektu (przeglądane są wszystkie pliki źródłowe w przestrzeni pracy, jest tworzona historia zmian plików oraz przejrzysta wizualizacja zebranych danych) oraz obliczanie dwóch podstawowych metryk: LOC i LOCPF.

2.3.1. Platforma Eclipse

Eclipse (`http://www.eclipse.org/`) to rozbudowane środowisko programistyczne stworzone przez firmę IBM i przekazane następnie społeczności Open Source. Jest ono wykorzystywane głównie do tworzenia projektów w języku Java, ale umożliwia również wykorzystanie innych popularnych języków tworzenia kodu. Środowisko to posiada mechanizm budowania rozszerzań za pomocą tzw. wtyczek, dzięki którym można rozbudowywać jego funkcje i możliwości.

Rozwój wtyczek w Eclipse umożliwia rozszerzenie PDE (ang. *Plug-in Development Enviroment*. Korzystania z tego rozszerzenia w środowisku eclipse odbywa się w odpowiedniej perspektywie (personalizacji okna głównego), dedykowanej do tworzenia kodów wtyczek oraz automatycznej ich kompilacji i debugowania. Minusem samego procesu tworzenia nowej wtyczki jest odpalanie nowej instancji systemu Eclipse z wbudowaną nową wtyczka (na etapie jej testowania), co powoduje wyraźne zużycie zasobów komputera.

2.3.2. Cechy rozwiązania

Przechowywanie danych

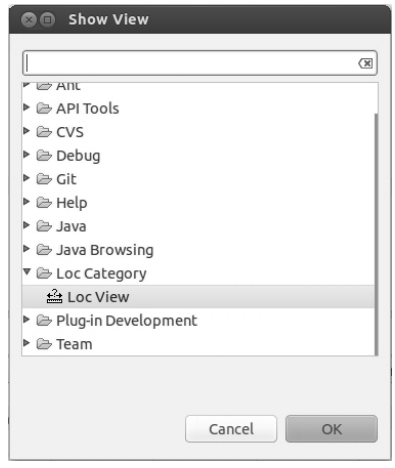
Przechowywanie historii zmian plików dla danego projektu odbywa się poprzez ich logowanie (rejestrowanie) w pliku XML. Podczas uruchamiania wtyczki sprawdzane jest, czy ten plik istnieje. Jeśli nie istnieje, zostaje utworzony i wypełniony danymi odpowiadającymi wyliczonym metrykom dla stanu zastanego. Następnie z każdą wprowadzoną zmianą plik ten uzupełniany jest informacjami o dokonanych modyfikacjach plików, z zapamiętanym stanem wcześniejszym dla danego pliku i projektu.

W schemacie pliku XML wyróżniono węzły: `header`, `file` oraz `delta`. Węzeł `file` odpowiada za opis śledzonych plików i ich stan aktualny. Każdy plik opisany jest atrybutem `path`, który zawiera pełną ścieżkę do pliku, i węzłami `loc` i `timestamp`, które określają, odpowiednio, ilość linii pliku i czas w jakim dany opis został zrobiony. Węzeł `delta` przechowuje kolejne zmiany plików. Posiada on atrybuty: `changedFilesCount` – mówiący ile plików zostało zmienionych, `locpf` – określający metrykę LOCPF w danej chwili, `timestamp` – oznaczający czas zapisu i `trackedFilesCount` – mówiący ile plików było obserwowanych. W węźle `delta` występuje podwęzeł `file` określający pliki, których dotyczy *delta*.

Wszystkie zmiany w tym rejestrze nanoszone są po wykryciu zmian w plikach źródłowych należących do przestrzeni roboczej (pliki konfiguracyjne projektu, które ulegają zmianie z każdą operacją zapisu plików, nie są śledzone).

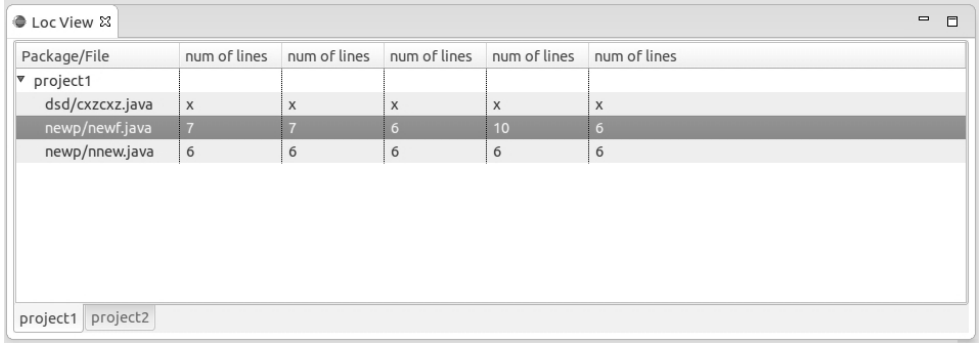
Widok i posługiwanie się wtyczką

Po zainstalowaniu wtyczki pojawi się możliwość dodania nowego widoku, w którym wtyczka będzie widoczna. Aby uaktywnić wtyczkę należy wybrać w górnym menu Eclipse pozycję *Window→Show View→Other...*, a w pojawiającym się później okienku przejść do *Loc category→Loc View*. Po tej operacji wtyczka pokaże się na dolnym panelu wtyczek (rys. 2.1).



Rys. 2.1: Widok na wybór widoku wtyczki.

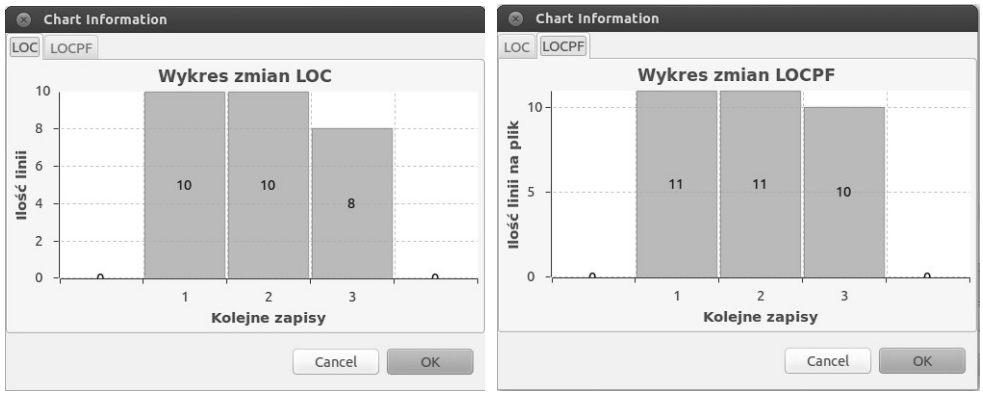
Po uruchomieniu wtyczki będzie dostępny podstawowy widok historii plików w projektach. Na dole okna zostaną uporządkowane wszystkie projekty. Dla każdej zakładki wyświetlone będą wszystkie pliki należące do danego projektu wraz historią 5 ostatnich zmian w ilości linii kodu (miara LOC). Jeżeli dla danego pliku nie istnieje zapis dla którejś ze zmian, wtedy zostaje wyświetlony znak *x*. Generowany widok pokazano na rysunku 2.2. W widoku tym można zwijać pliki należące do jednego z projektów.



Rys. 2.2: Widok na podstawowy wygląd wtyczki.

Dostęp do pozostałej historii zmian plików jest realizowany poprzez dwukrotne kliknięcie na nazwę pliku w widoku podstawowym. Operacja ta powoduje wyświetlenie okna dialogowego z wykresami. Domyślnie na pierwszym planie zostaje wyświetlony wykres ostatnich 100 zmian danego pliku. Jeśli plik posiada mniejszą bazę zmian, zostaną wyświetlone wszystkie dostępne dane. Widok zakładek z wykresami LOC oraz LOCPF przedstawiono na rysunku 2.3.

a) b)



Rys. 2.3: Widok na okno wykresów a) LOC, b) LOCPF.

2.4. Podsumowanie

Zaproponowane rozwiązanie pozwala śledzić historię dwóch wskaźników: linii kodu w pliku i średniej linii kodu na plik w projekcie. Sposób wizualizacji zapewnia łatwe analizowanie rozwoju projektu i oraz obserwowanie nie tylko ostatnich zmian, ale również dużo wcześniejszych etapów pracy.

Zbierane informacje umieszczane są w pliku XML. Przyjęto model przyrostowy, tzn. każda kolejna zaobserwowana zmiana zostaje dopisana do tego pliku. Ten prosty i czytelny mechanizm rejestracji ułatwia przeprowadzanie analiz oraz pozwala na wykorzystanie zbieranych informacji nie tylko w tej wtyczce, ale w innych programach badających efektywność pracy programisty.

Niestety takie podejście nie jest pozbawione wad. Reprezentacja danych w formacie XML nie jest optymalne pod względem zajętości pamięci. Poza tym konstrukcja programu zakłada, że plik jest otwierany, parsowany i zamykany w każdej iteracji pracy programisty, gdzie iteracjami nazywa się okresy między kolejnymi zapisaniami zmian w śledzonych plikach. Jest to również podejście nieoptymalne gdyż przy większych rozmiarach zapisanych danych danych ciągłe parsowanie pliku XML może być bardzo obciążające dla procesora.

Ważną rzeczą, o której należy pamiętać, jest łatwość użytkowania i płynność działania oprogramowania służącego do pomiaru jakości pracy oraz kodu. Programista nie powinien odwracać swojej uwagi od głównego zadania, jakim jest tworzenie kodu, przez obecność programów takich jak zaproponowana wtyczka. Pomiar pracy, agregacja danych i wstępna analiza powinny być transparentne dla programisty, a więc nie wymagać od niego dodatkowych cyklicznych akcji oraz nie spowalniać pracy środowiska. Jednocześnie, dla osoby zainteresowanej wynikami, reprezentacja wyników powinna być czytelna i dobrze usystematyzowana.

Utworzona wtyczka rzuca światło na szerokie zagadnienie jakim jest pomiar jakości pracy programisty. Nietrudno jest wyobrazić sobie, jakie możliwości analizy stwarzałoby obszerne narzędzie tego typu podłączone do środowiska programistycznego. Duże korporacje zajmujące się wytwarzaniem oprogramowania

mając na celu zwiększenie wydajności pracy oraz przede wszystkim jakości kodu chętnie inwestują środki w podobne narzędzia.

Literatura

[1] J. McCormack, D. Conway. Software characteristics and metrics. `http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/07.14.SWEng2/html/text.html#what_is_a_metric`, 2005.

[2] R.C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.

[3] B. Henderson-Sellers, L.L. Constantine, I.M. Graham. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). *Object Oriented Systems*, 3:143–158, 1996.

ROZDZIAŁ 3

REGUŁOWE WSPARCIE SEMANTYCZNEGO WNIOSKOWANIA

J. Bihun, A. Aniszczyk

3.1. Wprowadzenie

W ujęciu słownikowym semantyka to: *nauka o znaczeniu wyrazów, badająca w jakim zakresie i charakterze budowa formalna wyrazu określa jego znaczenie* (językoznawstwo); *dział semiotyki zajmujący się badaniem związków, jakie zachodzą między wyrażeniami języka a przedmiotami, do których się one odnoszą* (logika) [1]. W przybliżeniu semantyką można nazwać *teorię relacji zachodzących między językiem danej teorii a dziedziną tej teorii* [2].

Rzeczywistość można opisać za pomocą pewnych pojęć obrazujących istnienie pewnych faktów. Jednak trudno jest w pełni określić jakiś fakt bez jego odniesienia do całokształtu rzeczywistości. Dlatego ze znaczeniowym opisywaniem rzeczywistości kojarzy się dwa zagadnienia – posiadaną wiedzę oraz reguły wiążące nowe pojęcie z istniejącymi danymi.

Dla komputera dane to tylko ciąg zer i jedynek. Bez przyporządkowanego im znaczenia i wzajemnych relacji dane te nie stanowią wiedzy. Brak im szerszego kontekstu i oceny, podobnej do kontekstu i oceny rozpatrywanych przez człowieka.

Technologie sieci semantycznych web wzbogacają sieć WWW o elementy semantyczne [3]. Pozwalają na reprezentację wiedzy w formie czytelnej dla maszyn przez co może ona być wykorzystywana w dynamicznej analizie danych i procesach automatycznego wnioskowania.

Semantyczne wnioskowanie to proces dedukcji, który pozwala łączyć fakty i wyszukiwać istniejące między nimi powiązania. Celem tego procesu jest odnalezienie relacji niejawnych, ukrytych za zależnościami zdefiniowanymi przez zbiory reguł i praw. W maszynowym procesie dedukcji wykorzystywane są ontologie i reguły im towarzyszące.

Nieformalnie rzecz ujmując ontologia to zbiór faktów określających daną rzeczywistość. Dokładniejsza definicja mówi, że: *ontologia jest bezpośrednią specyfikacją konceptualizacji* [4]. Oznacza to, że ontologia stanowi opis pewnej rze-

czywistości. Rzeczywistość ta może być zawężona do jakiegoś fragmentu świata, nauki, wiedzy, zaś jej opis jest wyrażany w jakimś sformalizowanym języku.

Z informatycznego punktu widzenia ontologię definiuje pięć podstawowych typów aksjomatów:

- klasy, określane także jak typy (*classes*, *types*),
- obiekty, nazywane też indywiduami (*objects*, *individuals*),
- właściwości (*properties*),
- wartości danych (*data values*),
- reguły (*rules*).

W niniejszym rozdziale przedstawiono podstawowe informacje o technologiach sieci semantycznych web bez uciekania się do matematycznego aparatu logiki opisowej. Informacje te uzupełniono opisem eksperymentów związanych z budową dziedzinowego zasobu wiedzy oraz możliwości bezpośredniego wykorzystania tego zasobu w procesie semantycznego wnioskowania. Całość charakteryzuje się podejściem bardzo praktycznym.

3.2. Technologie sieci semantycznych web

3.2.1. Stos technologii

Sieć semantyczna web powstaje na bazie istniejących standardów i technologii, które ułożyć można w pewien stos. Na najniższym poziomie znajdują się specyfikacje odpowiedzialne za zapis znaków (ciągi znaków można zinterpretować jako nazwy przyporządkowane obiektom). Do kodowania znaków używanych w pisowni istniejących języków służy standard Unicode. Jednym ze specjalnych ciągów znaków szeroko stosowanym w dziedzinie informatyki jest URI (ang. *Uniform Resource Identifier*). Tego typu ciągi znaków pełniący rolę uniwersalnych, jednoznacznych identyfikatorów. Do ciągów URI zaliczane są również adresy URL (ang. *Uniform Resource Locator*), przy czym adresy te służą nie tylko do identyfikacji zasobów (obiektów), ale wskazują również fizyczne ich położenie.

Na następnym poziomie stosu znajduje się warstwa XML. W warstwie tej definiuje się przestrzenie nazw, dzięki którym można rozróżniać tak samo brzmiące, a inaczej interpretowane w różnych dokumentach terminy. Z warstwą XML związana jest również strukturyzacja zapisu treści w dokumentach (struktura drzewiasta). Ułatwia ona komputerowe przetwarzanie zapisanych danych oraz ich organizację.

Na warstwie XML opiera się warstwa tworzenie opisów semantycznych. Podstawowymi składnikami tej warstwy są RDF (ang. *Resource Description Framework*) i RDFS (ang. *RDF Schema*). Standard RDF określa sposób tworzenia zdań w postaci trójek składających się z: podmiotu, orzeczenia (asercji) i dopełnienia (ang. *subject*, *predicate* and *object*). RDF wraz z RDFS pozwala na definiowanie typów obiektów, właściwości obiektów oraz danych zapisanych w tych właściwościach.

Większą siłę wyrazu niż RDF przy tworzeniu ontologii ma język OWL (ang. *Web Ontology Language*). Wraz z językiem definiowania reguł SWRL (ang. *Semantic Web Rule Language*) stanowią podstawę dla kolejnych warstw omawianego stosu. Na najwyższym jego poziomach umieszczono warstwy kojarzone z logiką i dowodzeniem. Warstwy te dostarczają algorytmów wnioskowania w zakresie posiadanej ontologii oraz weryfikacji uzyskanych wyników pod kątem ich wiarygodności – zarówno pod względem poprawności dostępnych danych, jak i ich integralności (weryfikacja ta odbywa się w warstwie zaufania). Dopełnieniem technologii są odpowiednie algorytmy szyfrowania.

W poniższych podrozdziałach opisano nieco dokładniej języki pozwalające zapisywać ontologie (języki zgodne ze specyfikacją OWL) oraz reguły (język SWRL).

3.2.2. OWL

OWL to zatwierdzona przez organizację W3C specyfikacja struktury zapisu ontologii. W strukturze tej wyróżnia się pojęcia: klasy, właściwości, indywidua oraz wartości danych [5]. Zaprojektowaną ontologię można zapisać do pliku w składni: OWL/XML, RDF/XML, OWL functional syntax czy też Turtle. W samej strukturze ontologii wyróżnia się TBox (terminologię – zbiór pojęć i binarnych relacji pomiędzy nimi (właściwości, nazywanych też rolami) oraz zbiór ograniczeń (nazywanych aksjomatami) – czyli wszystkiego, co opisuje klasy, podklasy, klasy równoważne, klasy rozłączne) oraz ABox (zbiór asercji unarnych i binarnych opisujących, odpowiednio wystąpienie pojęć i ról – czyli wszystkiego, co dotyczy indywiduów).

Klasy i obiekty

Klasy mogą mieć podklasy (klasy potomne). Aby jawnie przyporządkować jakiś obiekt do danej klasy korzysta się z predykatu `rdf:type` (*jest typu*). Na przykład zdanie: „Sputnik jest robotem mobilnym” można zapisać w notacji Turtle w następujący sposób:

```
@prefix ex: <http://example.org/>.
ex:Sputnik rdf:type ex:Robot_Mobilny.
```

Obiekt należący do klasy potomnej równocześnie należy do klasy bazowej. Jeśli wymieniona w powyższym przykładzie klasa `Robot_Mobilny` byłaby podklasą klasy `Robot`, to `Sputnik` należałby również do klasy `Robot`.

W OWL klasą bazową wszystkich klas jest `Thing`. Obiekty mogą być przyporządkowane do klas w sposób jawny (poprzez użycie właściowości `rdf:type` i mechanizmu dziedziczenie). O przynależności obiektu do danej klasy mogą również decydować właściwości obiektu. Specyfikacja OWL zezwala bowiem na definiowanie ograniczeń przy deklaracjach klas. Dotyczyć to może w szczególności np. wartości jakichś właściwości. Stąd też wartość danej właściwość może stać się kryterium decydującym o przynależności danego obiektu do określonej klasy. Na przykład przynależność jakiegoś obiektu do klasy `Robot_Kołowy` może wynikać ze spełnienia nałożonego na tę klasę warunku posiadania co najmniej

jednego koła w strukturze. Co więcej, jeden obiekt może należeć równocześnie do wielu klas, jeśli został do nich jawnie przypisany lub jeśli spełnia nałożone na te klasy warunki.

Właściwości

Właściwości w OWL (ang. *properties*) dzielą się na właściwości określające powiązania między obiektami (`Object Property`) lub przyporządkowujące wartość jakiemuś obiektowi (`Data Property`). Na przykład właściwość `ex:Jest_Produkowany_Przez` może wiązać jakiegoś robota (`ex:Sputnik`) z producentem robotów (`ex:Dr_Robots`), przy czym właściwość `ex:Jest_Autonomiczny`) tego robota może mieć wartość `true` (prawda). W notacji Turtle można to zapisać w następujący sposób:

```
@prefix ex: <http://example.org/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
ex:Sputnik ex:Jest_Produkowany_Przez ex:Dr_Robots;
           ex:Jest_Autonomiczny "true"^^xsd:boolean.
```

Właściwości (`Object Property`) mogą być deklarowane jako:

- funkcyjne (`functional`) albo odwrotnie funkcyjne (`inverse-functional`),
- przechodnie (`transitive`),
- symetryczne (`symmetric`) albo asymetryczne (`asymmetric`),
- zwrotne (`reflexive`) albo niezwrotne (`irreflexive`),
- odwrotne (`inverse`),
- podrzędne (`subproperty`),
- równoważne (`equivalent`),
- rozłączne (`disjoint`).

Natomiast właściwości (`Data Property`) mogą być tylko funkcyjne.

3.2.3. SWRL

SWRL to zaproponowana przez W3C kombinacja języka OWL oraz RuleML (ang. *Rule Mark-up Language*) [6]. Pozwala ona na definiowanie reguł określających takie właściwości obiektów, które nie są bezpośrednio wprowadzone i nie wynikają z samej definicji danego aksjomatu zapisanej przy użyciu języka OWL.

Reguły SWRL mają postać implikacji `poprzednik -> następnik`, w których zarówno `poprzednik` jak i `następnik` mogą mieć postać koniunkcji wielu asercji lub operacji zależnych od zmiennych (nazwy zmiennych poprzedza się znakiem zapytania). Na przykład regułę: *jeśli y jest rodzicem x oraz z jest bratem y to z jest wujkiem x* można zapisać w następujący sposób:

```
rodzic(?x,?y), brat(?y,?z) -> wujek(?x,?z)
```

Reguły implikują nowe trójki RDF. Wyrażenia SWRL można przedstawić jako wyrażenia:

- niewykorzystujące żadnych operacji spoza języka OWL, ale pozwalające na pozbawiony niektórych ograniczeń zapis właściwości i klas,

- wykorzystujące wbudowane operacje porównań (równość, nierówności),
- wykorzystujące wbudowane operacje matematyczne (np. odejmowanie, dzielenie, pierwiastek),
- wykorzystujące boolowski operator zaprzeczenia,
- wykorzystujące operacje na łańcuchach znaków,
- wykorzystujące operacje na reprezentacji czasu lub daty,
- wykorzystujące operacje na definicjach aksjomatów,
- wykorzystujące operacje na listach.

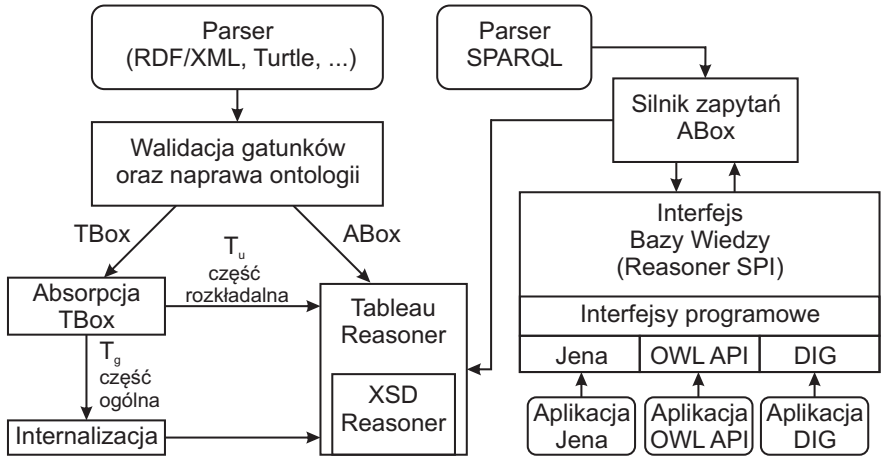
SWRL nie wspiera zliczania. Na przykład nie można stworzyć reguły, która policzy i zapisze we właściwościach firmy ilość zbudowanych przez nią, a zapisanych w ontologii, modeli robotów. Istnieją jednak niestandardowe rozszerzenia języka SWRL, które umożliwiają takie zliczanie, jak np. SQWRL (ang. *Semantic Query-Enhanced Web Rule Language*).

3.2.4. Wnioskowanie w ontologiach

Program realizujący wnioskowanie na podstawie zdefiniowanej ontologii nosi nazwę *reasonera* (co można tłumaczyć na *silnik wnioskowania*). Działanie reasonera polega na wczytaniu istniejącej ontologii, jej analizie oraz stworzeniu bazy danych, na którą składają się dane zapisane bezpośrednio i wywnioskowane. Ontologia może być zapisana przy użyciu dowolnej technologii, o ile tylko istnieje parser, który potrafi tę ontologię przetworzyć (odpowiednio do zaprojektowanego interfejsu aplikacji). Wynikiem działania reasonera może być baza trójek RDF, która może być odczytana w wyniku zadania zapytania SPARQL.

Sercem działania silnika wnioskowania jest implementacja algorytmu *tableaux* [7]. Dzięki niemu sprawdzana jest spójność ontologii. Podczas tego sprawdzania budowany jest graf o węzłach skojarzonych z obiektami i danymi oraz łukach skojarzonych z właściwościami. Algorytm rozszerza graf dopóki nie zostanie odkryta sprzeczność lub graf stanie się kompletny – żadne nowe informacje nie będą mogły zostać wygenerowane.

Architekturę silników wnioskowania zilustrowano na przykładzie popularnego reasonera Pellet (rys. 3.1). Bloki oznaczone prostokątami o zaokrąglonych rogach reprezentują elementy zewnętrzne (parsery i aplikacje, z którymi Pellet może współpracować poprzez programowe interfejsy Jena, OWL API czy DIG). Pozostałe bloki reprezentują elementy własne reasonera. Blok *Walidacja gatunku i naprawa ontologii* (ang. *species validation*) odpowiedzialny jest za przeprowadzenie procesów: identyfikacji dialektu języka OWL (jak np. Lite, DL lub Full) w jakim zdefiniowano badaną ontologię oraz naprawy ontologii (polegającej na wskazaniu i eliminacji występujących w niej niespójności). Blok *Interfejs Bazy Wiedzy* dostarcza wygodnych funkcji umożliwiających dostęp do oferowanego serwisu wnioskowania poprzez interfejs programowania serwisu SPI (ang. *Service Programming Interface*). W reasonerze Pellet można uruchamiać wiele implementacji algorytmów tableaux (domyślnie wykorzystywany jest algorytm SROIQ(D)) oraz wykorzystywać reguły napisane w języku SWRL (zapewnia to moduł *XSD reasoner*).



Rys. 3.1: Architektura reasonera Pellet (według [7]).

3.3. Narzędzia

Projektowanie i wykorzystanie ontologii ułatwiają narzędzia programowe. Zalicza się do nich między innymi:

- edytory ontologii – programy służące do modelowania/projektowania ontologii. Jednym z popularnych narzędzi tego typu jest edytor wbudowany w program Protégé. Jest to narzędzie rozwijane na Uniwersytecie Stanforda w języku Java. Obsługuje wiele wtyczek, między innymi reasonery i wizualizatory ontologii. Interfejs graficzny tego programu ma bardzo szerokie możliwości modyfikacji – pozwala na edycję zakładek oraz umieszczanie w nich narzędzi.
- reasonery – programy do analizowania ontologii i wykrywania właściwości ukrytych, wynikających ze struktury ontologii oraz towarzyszących jej reguł. Można je podłączać do różnych środowisk (w programie Protégé można korzystać z reasonerów: Hermit, FACT++ oraz Pellet). Podłączanie to może odbywać się w sposób dynamiczny (w trakcie działania programu Protégé można dowolnie przełączać reasonery, uruchamiać je, zatrzymywać ich działanie lub aktualizować analizowane przez nie dane i reguły). Reasonery dostarczają informacji o spójności tworzonej ontologii i jeśli współdziałają z edytorami, niemożliwe jest wprowadzanie sprzecznych informacji podczas modelowania ontologii.
  Podczas prób uruchomienia wspomnianych reasonerów w programie Protégé okazało się, że tylko Pellet jest w stanie poprawnie przetwarzać reguły SWRL. Jednak nawet najnowsze wersje tego reasonera zawierają błędy uniemożliwiające przetworzenie niektórych reguł [8].
- API – interfejsy programowe pozwalające na wykorzystanie ontologii we własnych programach. Wśród API obsługujących OWL najprężniej rozwijane jest OWL API napisane w języku Java. Dostarcza rozwiązań, które pozwalają na tworzenie ontologii, odczyt oraz wyszukiwanie odpowiednich informacji. Ob-

sługuje wiele składni zapisu ontologii. OWL API nie jest reasonerem, więc do pełnej obsługi potrzebne są odpowiednie biblioteki.
- silniki obsługi zapytań – istnieje kilka języków, za pomocą których można przeglądać ontologie. Jednym z nich jest SPARQL, wspierany przez Protégé (wsparcie to jest ograniczone – nie ma możliwości przeszukiwania trójek RDF powstałych w wyniku uruchomienia silnika wnioskowania, przeszukiwane są tylko wyjściowej bazy wiedzy).
- wyszukiwarki ontologii – usługi służące do wyszukiwania opublikowanych ontologii, jak np. Swoogle czy DAML Ontology Library. Dzięki tym narzędziom można przeszukiwać ontologie stworzone przez specjalistów na całym świecie.
- wizualizatory ontologii – przedstawiają ontologie w formie grafu (Protégé posiada wbudowaną wizualizację OWLviz, ponadto obsługuje wizualizację SOVA stworzoną na Politechnice Gdańskiej).
- biblioteki programowe – służą do programowego przetwarzania ontologii, komunikacji z bazami wiedzy itp. Jednym z bogatszych interfejsów programowania aplikacji zaimplementowano w środowisku Jena (posiada interfejs webowy, obsługę plików OWL wielu typów i obsługuje reasoner Pellet).
- bazy wiedzy – służą do przechowywania i udostępniania wiedzy zapisanej w ontologiach. Jedną z bardziej znanych implementacji bazy wiedzy jest Open Link Virtuoso. Rozwiązanie do dostarcza interfejsu zapytań SPARQL, pozwala na import plików zapisanych w języku RDF/XML, obsługuje tylko część możliwości języka OWL (`http://docs.openlinksw.com/virtuoso/rdfsparqlrule.html#rdfsparqlrulemake`).

3.4. Przypadki użycia

W prezentacji z roku 2009 [9] wymieniono kilka pól zastosowań SWRL:

- inżynieria – diagnostyka,
- ekonomia, biznes – reguły rynkowe
- prawo – badanie legalności,
- medycyna – określanie zgodności, dobór leków,
- internet – weryfikacja uprawnień dostępu.

Diagnostyka oparta o system reguł pozwala na niezależną od człowieka analizę problemów. Niektóre usterki mogą być samodzielnie diagnozowane przez urządzenie, zgłaszane użytkownikowi lub nawet naprawiane. Ponadto taki system oparty na regułach diagnozowania umożliwia proponowanie rozwiązań użytkownikom zgłaszającym problem [10], [11]. Technologia ta jest wykorzystywana przez niektóre serwisy – klient wypełnia ankietę, a system analizując udzielone odpowiedzi przyporządkowuje zgłaszaną usterkę do odpowiedniej klasy i proponuje rozwiązania. Daje to zysk w postaci oszczędności pracy serwisantów oraz większego zadowolenia u klientów, którzy nie muszą czekać na obsługę.

W zakresie komercyjnym technologie sieci semantycznej pozwalają np. na personalizację usług i aplikacji. Reguły umożliwiają lepsze dostosowanie wyświetlanych reklam, co zwiększa szansę na zainteresowanie odbiorcy danym produk-

tem, a także zmniejsza frustrację wywołaną zawartością, która użytkownikowi może wydawać się niewłaściwa. Reguły usprawniają również zarządzanie cenami produktów – dzięki zbiorowi reguł można wprowadzić automatyczne przecenianie produktów lub przedstawiać użytkownikowi-menedżerowi listę produktów, których ceny powinny zostać zmienione. W ten sposób można ograniczyć konieczność magazynowania towarów na które nie ma popytu lub uzyskać większy przychodu na produktach chętnie wybieranych [12].

Istnieją ontologie powstałe na bazie istniejącego prawa. Pomagają one w analizie sprawy, określeniu legalności i dobraniu odpowiednich przepisów. Tego typu ontologie mogą być wykorzystane przez prawników [13].

Dobór leków, diety czy metod leczenia do stanu pacjenta można przedstawić w formie reguł, które zawierają informacje o synergii między lekami, przeciwwskazaniach, skutkach ubocznych i zakresie stosowania [14]. Wykorzystanie odpowiedniej ontologii pozwala na bardziej dopasowaną do stanu pacjenta kurację i zapobiega błędom.

Wykorzystanie reguł pozwala również na wybiórcze, uzależnione od sytuacji przyznawanie dostępu do zasobów internetowych. W pracy [15] opisano system kontroli dostępu do informacji medycznych bazujący na przyznanych uprawnień (stopień pokrewieństwa do pacjenta, pełnomocnictwo itp.) zgodnie z obowiązującymi przepisami.

3.5. Przykład zastosowanie reguł w procesie semantycznego wnioskowania

W celu zobrazowania efektów jakie może przynieść zastosowanie reguł podczas semantycznego wnioskowania przeprowadzono proste eksperymenty. Polegały one na zaprojektowaniu dziedzinowej ontologii oraz zestawu reguł oraz ich wykorzystaniu do klasyfikacji robotów. Projekt dziedzinowej ontologii przygotowano w programie Protégé 4.3.0 (z reasonerami Fact++ 1.6.3, Hermit 1.3.8 i Pellet 2.2.0). Program wypisujący listę klas i właściwości odpowiadających danemu obiektowi przy zadanej ontologii i regułach zaimplementowano w języku Java, wykorzystując OWL API i reasonerem Pellet (wersja 2.2.0).

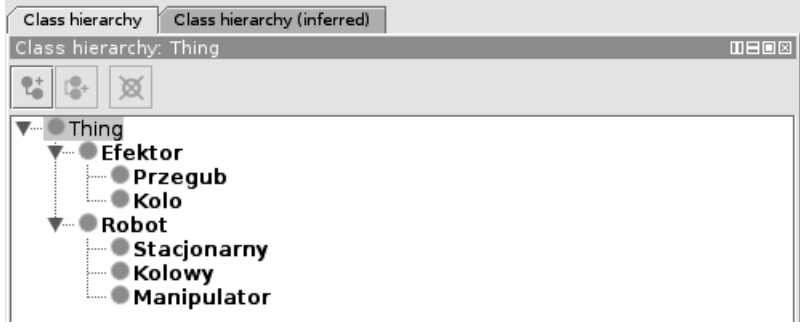
3.5.1. Problem klasyfikacji robotów

Robotyka to interdyscyplinarna nauka, której celem jest tworzenie robotów – maszyn wyposażonych w autonomię oraz możliwość ekspresji poprzez efektory. Roboty można klasyfikować na podstawie posiadanych przez nie cech. Dzięki nim robot może zostać przypisany do odpowiedniej grupy, można też określić jego pewne właściwości wynikające konstrukcji.

Projektując dziedzinową ontologię można obrać różne strategie. Można na przykład podjąć próbę stworzenia szczegółowego opisu, z wieloma informacjami o każdym możliwym elemencie lub też zaproponować opis bardziej ogólny, z wyróżnieniem głównych klas oraz właściwości. Na przykład fakt, iż dany robot jest manipulatorem można zapisać na wiele sposobów:

- poprzez jawne przypisanie danego obiektu do klasy `Manipulator`. Jest to najprostsza metoda, specjalnie nie różniąca się od wpisania rekordu do konkretnego pola w bazie danych.
- poprzez zdefiniowanie właściwości `Ramie` i reguły *robot posiadający ramię jest manipulatorem*. W tej metodzie korzysta się częściowo z mechanizmu wnioskowania, co pozwala opisać robota przez deklarację jego budowy. Opis ten jest jednak bardzo ogólny, nie zawiera żadnych informacji o właściwościach poszczególnych ramion.
- poprzez osobne zdefiniowanie pojedynczych przegubów ramienia, przypisanie ich robotowi oraz dokonanie odpowiedniej klasyfikacji z zastosowaniem reguł. Jest to metoda pozwalająca na bardzo szczegółowy opis robota i jego parametrów. Pewną trudność można napotkać przy opisie układów składających się z dwóch manipulatorów (na wzór robotów humanoidalnych).
- poprzez osobną definicję przegubów, ramion i samego robota (opisywane są wszystkie elementy z odpowiednimi właściwościami). Ta metoda generuje dużo informacji, jednak jest bardzo rozbudowana, nieczytelna, rodząca trudności przy wychwytywaniu błędów.

Wnioskowanie na podstawie ontologii Wnioskowanie o przynależności obiektów do klas może odbywać się na podstawie samych aksjomatów OWL, bez użycia reguł SWRL (pierwsza z metod wyżej wymienionych tak właśnie działa, kolejne wymagają zastosowania bardziej rozbudowanych ontologii i reguł). Zobrazowano to na przykładzie małej ontologii, zawierającej definicje kilku klas pokazanych na rysunku 3.2. Ponadto w stworzonej ontologii zdefiniowano dwa obiekty: `IRB_140` – robota i `Obrotowa_podstawa` – przegub typu platforma obrotowa oraz utworzono właściwości wiążące obiekty: `Posiada_Przegub` i `Posiada_Kolo`.



Rys. 3.2: Hierarchia klas wykorzystanych podczas testów.

Robotowi `IRB_140` przypisano posiadanie platformy obrotowej. W celu przypisania robota `IRB_140` do manipulatorów zdefiniowano klasę `Manipulator` jako podklasę `Robot` z warunkiem:

```
Posiada_Przegub some Przegub
```

Użycie `some` w tym przypadku oznacza, że występuje przynajmniej 1 aksjomat `Posiada_Przegub` z dopełnieniem `Przegub` dla danego indywiduum. Sprawdzono działanie tego przypisania – niezależnie od użytego reasonera `IRB_140` nie został przypisany do klasy `Manipulator`. Jednak udało się przypisać robota `IRB_140` do klasy `Manipulator` poprzez zdefiniowanie równoważności, a nie podklasy: klasa `Manipulator` jest równoważna wszystkim obiektom spełniającym podany wyżej warunek (równoważność pozwala zdefiniować warunki konieczne i wystarczające). Ponadto, po dodaniu większej liczby przegubów, próbowano stworzyć podklasę manipulatorów redundantnych (manipulatorów, których liczba stopni swobody jest większa niż 6, jednak przy założeniu, że każdy przegub wprowadza jeden stopień swobody). Żaden z reasonerów nie był w stanie zwrócić poprawnego przypisania na podstawie tej reguły. Jedynie reguła `min 1` równoważna regule `some` jest przetwarzana poprawnie. Nie były przetwarzane reguły równości (`exactly`) ani nierówności (`min`, `max`).

W następnej kolejności spróbowano stworzyć klasę robotów stacjonarnych (w uproszczeniu robotów, które nie posiadają kół). Klasie `Robot_Stacjonarny` przypisano równoważność

```
Posiada_Kolo exactly 0 Kolo
```

Warunek ten nie został zrealizowany w żadnym reasonerze.

Podjęto jeszcze jedną próbę zdefiniowania manipulatora: przy użyciu restrykcji nałożonej na klasę `Manipulator`:

```
Liczba_Stopni_Swobody min X
```

gdzie `Liczba_Stopni_Swobody` to właściwość o wartości liczbowej całkowitej. Taką właściwość przypisano robotowi `IRB_140` i nadano jej wartość 6 (zgodnie ze specyfikacją tego manipulatora). Przy `X = 0` do klasy `Manipulator` został zaklasyfikowany robot `IRB_140` oraz robot Sputnik (robot mobilny, bez właściwości `Liczba_Stopni_Swobody`). Sprawdzono również inne restrykcje: `max` i `exactly`. W przeciwieństwie do właściwości między obiektami (`Object Properties`) restrykcje dotyczące właściwości przechowujących dane (`Data Properies`) są dobrze zrealizowane w reasonerach.

Następnie stworzono podklasę manipulatorów o 6 stopniach swobody. Dla tej klasy ustanowiono restrykcję:

```
Liczba_Stopni_Swobody value 6
```

Restrykcja ta zadziałała poprawnie: robot `IRB_140` został przypisany do klasy manipulatorów o 6 stopniach swobody, a po zmianie wartości liczby stopni swobody na 5 przestał być do niej klasyfikowany.

Następnie stworzono obiekt `Ramie` i przypisano go do klasy `Efektor`. Wprowadzono również warunek, którego efektem miało być przypisanie robotów z co najmniej jednym ramieniem do klasy `Manipulator`:

```
Posiada_Efektor min 1 Ramie
```

Aby był to warunek konieczny i wystarczający, klasa `Manipulator` musiała być zdefiniowana jako równoważna klasie z tą restrykcją. W efekcie jakiekolwiek obiekty posiadające w swojej strukturze ramię stawały się manipulatorami.

Wnioskowanie oparte na regułach Do wnioskowania z wykorzystaniem reguł SWRL wykorzystano przedstawioną już hierarchię klas (rys. 3.2). Robotowi `Sputnik` przyporządkowano właściwość `Posiada_Efektor` z wartością `Kola_Sputnika`, a robotowi `IRB_140` – właściwość `Posiada_Efektor` z wartością `Obrotowa_podstawa`. Ponieważ `IRB_140` posiada przegub, powinien on być zaklasyfikowany do manipulatorów. Reguła to zapewniająca ma postać:

```
Przegub(?p), Robot(?r), Posiada_Efektor(?r, ?p) -> Manipulator(?r)
```

Oprócz tej reguły stworzono analogiczne reguły dla robotów kołowych i stacjonarnych (bez kół):

```
Robot(?r), (Posiada_Efektor max 0 Kolo)(?r) -> Stacjonarny(?r)
```

W regułach tych zmieniano `max` na `exactly`, `min` oraz przypisywano im różne wartości. Dla opracowanego zestawu reasoner Pellet nie przyporządkował manipulatora do klasy robotów stacjonarnych przez zaprzeczenie właściwości posiadania kół (`Posiada_Efektor Kolo`) – co było zgodne z założenia otwartości świata (negacja faktu musi być jawnie podana).

Stwierdzenie *manipulator posiadający więcej niż 6 stopni swobody jest redundantny* w języku reguł można zapisać w następujący sposób:

```
Robot(?r), (Posiada_Efektor min 6 Przegub)(?r) -> Redundantny(?r)
```

W celu sprawdzenie działania reasonera stworzono nieprawdziwą regułę i dopisano 5 pozostałych przegubów jakie ma manipulator IRB 140:

```
Robot(?r), (Posiada_Efektor min 5 Przegub)(?r) -> Stacjonarny(?r)
```

Okazało się, że reasoner Pellet nie jest w stanie klasyfikować na podstawie tej reguły.

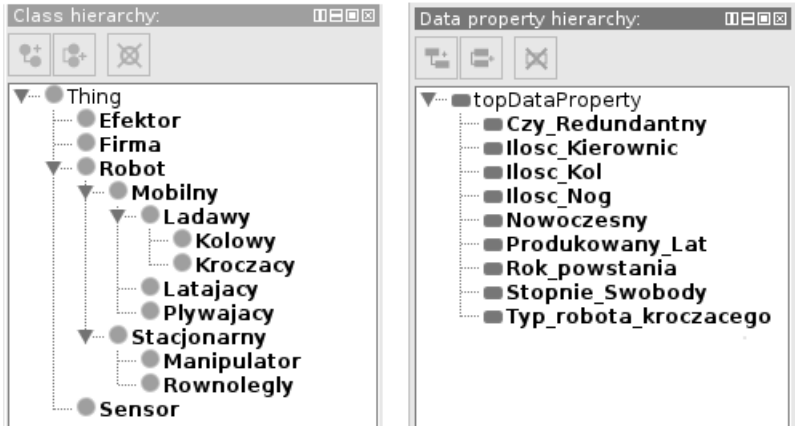
Reguły wymagające zliczania aksjomatów nie działają w dostępnych reasonerach, zarówno jeśli są to reguły wykorzystujące składnię OWL, jak i SWRL. Obsługiwane są zaś reguły bazujące na informacjach o istnieniu danej relacji między obiektami. To znaczy, że można tworzyć regułę wymagającą istnienia przynajmniej jednego obiektu o określonej relacji z innymi, lecz nie da się stworzyć reguły wymagającej jakiejkolwiek innej restrykcji względem liczebności predykatu.

W kolejnej części zaprezentowano projekt ontologii opartej na obiektach, których opis w mniejszym stopniu tworzy sieć powiązań między indywiduami i klasami, a w większym zawiera informacje w formie danych. Stworzenie takiej ontologii jest możliwe dzięki regułom SWRL, które pozwalają na wprowadzanie funkcji w aksjomaty typu `Data Property`.

Rozbudowa ontologii Do stworzenia ontologii wykorzystano program Protégé. W celu szybkiej weryfikacji reguł wykorzystano wtyczkę dostarczającą reasoner Pellet. Niestety reasoner w dostępnej wersji zawierał pewne niedociągnięcia, więc dodatkowo skorzystano z programu napisanego w języku Java, wykorzystującego OWL API oraz inną wersję reasonera Pellet.

Oprócz klas robotów w zaprojektowanej ontologii uwzględniono kilka innych klas: `Firma`, `Efektor` i `Sensor` oraz właściwości (rys. 3.3). Wszystkie bardziej

skomplikowane zależności starano się zamieścić we właściwościach obiektów i odpowiednich regułach SWRL.



Rys. 3.3: Hierarchia klas i właściwości rozbudowanej ontologii.

Zadeklarowane właściwości typu `Data Property` odpowiadają cechom, które stosunkowo łatwo pozyskać i jednocześnie łatwo wykorzystać do różnicowania robotów. Ponadto dwie właściwości typu `Object Property` (niepokazane na rysunku) służą do określenia zależności między robotem, a jego elementami: robot może posiadać efektor lub sensor i być produkowanym przez określoną firmę.

Klasy i właściwości uszczegółowiono zgodnie z możliwościami języka OWL poprzez dodatkowe warunki:

- rozłączność klas robot, efektor, sensor i firma,
- narzucenie klas dla właściwości łączących dwa obiekty,
- narzucenie właściwości typu danej, jeśli dane te mają wartość logiczną prawda-fałsz lub wartość liczbową całkowitą (rok produkcji, czas produkcji (lata), liczba kół, kierownic, stopni swobody i nóg, redundancja, brak degeneracji)
- inwersja właściwości: *robot X jest produkowany przez firmę Y* oraz *firma Y produkuje robota X*.

Wykorzystując reguły SWRL można dostarczyć czytelniejszego opisu niż same deklaracje klas i właściwości. Ponadto wykorzystując strukturę klas jako szkielet wszelkie inne informacje można przechowywać jako właściwości. Np. czas od stworzenia robota jest dobrym kryterium do podziału robotów na nowoczesne i starsze. Równocześnie jednak taką informację lepiej zachowywać jako subiektywną ocenę konstrukcji, a nie podstawę klasyfikacji.

Rozbudowa reguł Reguły wykorzystują informacje o robocie i określają pewne jego cechy. Poniżej przedstawiono zaimplementowane reguły:

- definicja inwersji *jeśli robot Y posiada efektor lub sensor X, to efektor lub sensor X jest częścią robota Y* – przypadek ten opisano w podrozdziale 3.2.3. Dzięki tej

definicji z każdym sensorem i efektorem można związać listę robotów, w których one występują:

```
Robot(?r), Ma_Efektor(?r, ?e) -> Jest_Czescia_Robota(?e, ?r)
Robot(?r), Ma_Sensor(?r, ?e) -> Jest_Czescia_Robota(?e, ?r)
```

- określenie cech dynamiki robota na podstawie jego parametrów – są to reguły takie jak: degeneracja robota kołowego, typ robota kroczącego, redundancja manipulatora. Cechy te łatwo wyznaczyć przy znajomości takich liczbowych parametrów robota jak: liczba kół i kierownic w robocie kołowym lub liczba stopni swobody manipulatora:

```
Kroczacy(?r), Ilosc_Nog(?r, ?l), greaterThan(?l, 3)
 -> Typ_robota_kroczacego(?r, "statycznie stabilny")
Kroczacy(?r), Ilosc_Nog(?r, ?l), equal(?l, 2)
 -> Typ_robota_kroczacego(?r, "humanoidalny, dynamicznie stabilny")
Kolowy(?r), Ilosc_Kierownic(?r, ?l), Ilosc_Kol(?r, ?p),
add(?k, ?l, ?p), greaterThan(?k, 1), lessThan(?k, 4)
 -> Brak_Degeneracji(?r, true)
Manipulator(?r), Stopnie_Swobody(?r, ?l), greaterThan(?l, 6)
 -> Czy_Redundantny(?r, true)
```

- przydzielenie robota do klasy – robot jest kroczący gdy posiada nogi, a kołowy gdy posiada koła. Aby dokonać takiej klasyfikacji wystarcza sprawdzić, czy ta właściwość ma wartość liczbową większą od 0.

```
Ilosc_Nog(?r, ?l), greaterThan(?l, 0) -> Kroczacy(?r)
Ilosc_Kol(?r, ?l), greaterThan(?l, 0) -> Kolowy(?r)
```

- wyliczenie nowej właściwości – jak stary jest dany robot. Liczbowy zapis roku wytworzenia czy rozpoczęcia produkcji robota jest odejmowany od roku bieżącego. Dzięki wyznaczonej właściwości można określić czy robot jest nowoczesny (powstał wcześniej niż 5 lat temu), czy nie.

```
Rok_powstania(?r, ?rok), subtract(?wiek, 2014, ?rok)
 -> Produkowany_Lat(?r, ?wiek)
Produkowany_Lat(?r, ?wiek), greaterThan(?wiek, 5)
 -> Nowoczesny(?r, false)
Produkowany_Lat(?r, ?wiek), lessThanOrEqual(?wiek, 5)
 -> Nowoczesny(?r, true)
```

Łącznie zdefiniowano jedenaście reguł, w których zastosowano obce językowi OWL operacje porównań, sumowania i dodawania.

3.6. Podsumowanie

3.6.1. Wykorzystanie regułowego wsparcia wnioskowania w robotyce

W robotyce podstawowym zagadnieniem jest tworzenie robota. Jest to proces twórczy, w którym dedukcja, zdefiniowana jako wyciąganie wniosków z niezależnych od człowieka przesłanek, jest elementem drugoplanowym. Mimo to

wybór odpowiednich elementów zarówno fizycznych (efektory, sensory), jak i algorytmów sterowania wymaga analizy dostępnej wiedzy oraz oceny użyteczności. Dlatego poniżej omówiono funkcje zaprojektowanej ontologii, które mogą być przydatne w robotyce.

Utworzona ontologia wraz z regułami reprezentuje kompleksowy zbiór informacji na temat robotów. Definiowanie nowego robota wraz ze wszystkimi jego właściwościami wymaga wprowadzenia niewielkiego zbioru danych.

Pierwszym, oczywistym zastosowaniem ontologii jest wyszukiwanie robotów należących do danej klasy. Można to zrobić formułując odpowiednie zapytanie (nakładając ograniczenia lub stosując reguły). Dzięki temu można uzyskać przykłady konkretnych typów robotów. Posiadając taką wiedzę można przyjrzeć się zastosowanym w danym robocie rozwiązaniom konstrukcyjnym i wykorzystać je we własnym projekcie.

Reguły określają podział robotów kroczących ze względu na zastosowane w nich rozwiązania konstrukcyjne – na podstawie liczby nóg takiego robota. Liczba nóg jest łatwa do obliczenia i wynikają z niej pewne właściwości robota oraz charakter jego ruchu. Ponadto manipulatory są dzielone pod kątem ich redundancji, a dla robotów kołowych można stwierdzić, czy nie są zdegenerowane. Wprawdzie nie następuje tu przypisanie do klasy lecz dodanie właściwości zawierającej opis, jednak nadal jest to określenie pewnego typu i może zostać wykorzystane jako kryterium wyszukiwania.

Oprócz klasyfikacji użycie języka reguł pozwala na analizowanie właściwości poprzez działania na danych, które są ich wartościami. Na przykład ważnym zagadnieniem w wypadku tak szybko rozwijającej się dziedziny jak robotyka jest aktualność pewnych rozwiązań. Roboty mobilne powstałe ponad dwadzieścia lat temu mogły mieć sterowanie oparte na licznikach (feed forward) – jest to rozwiązanie przestarzałe, dające niedokładne śledzenie trajektorii, a brak sprzężenia zwrotnego powoduje narastanie błędu. Ważne dla konstruktora jest, aby wzorować się na najnowszych, najbardziej aktualnych rozwiązaniach. Zaprojektowana ontologia pozwala wyliczyć wiek robotów, a stąd już jest tylko mały krok do zakwalifikowania robotów do grup: nowoczesne i przestarzałe. Dzięki temu można ograniczyć zapytania do robotów nowoczesnych.

Oprócz roku powstania robot może być zdefiniowany poprzez zbiór efektorów i sensorów jakie posiada. Równocześnie każdemu efektorowi i sensorowi za pomocą reguły można przypisać roboty, w których te elementy występują. Dzięki temu projektant lub badacz może poznać strukturę robota oraz ocenić funkcje, które może zrealizować za pomocą oferowanego sprzętu.

Pewne możliwości wykorzystania ontologii mają także związek z funkcjami, jakie posiadają aplikacje obsługujące wnioskowanie. Szczególnie warte podkreślenia jest to, że aplikacje mają możliwość udostępnienia podglądu aksjomatów, które doprowadziły do nowego faktu. Dzięki dokładnym informacjom o procesie wnioskowania, można zweryfikować poprawność jego działania lub rozszerzyć dostępną wiedzę dodając nowe reguły i warunki.

3.6.2. Wnioski

Technologie sieci semantycznej to interesująca koncepcja, która pozwala na implementację stworzonych lub odkrytych przez człowieka reguł w programie komputerowym. Technologie te umożliwiają analizę wprowadzonych danych, odnajdywanie między nimi dodatkowych zależności i dostarczanie pewnych rozwiązań, których samodzielne pozyskanie byłoby dla człowieka uciążliwe lub wymagałoby specjalistycznej wiedzy.

Przedstawienie wiedzy w postaci ontologii i reguł ułatwia proces wnioskowania. Oprócz tego, system oparty na regułach weryfikuje zebraną wiedzę i daje możliwość detekcji oczywistych sprzeczności.

Istotnym ograniczeniem w szerokim zastosowaniu technologii sieci semantycznych web są braki w oprogramowaniu. Tyczy się to przede wszystkim reasonerów, które nie obsługują całej specyfikacji OWL ani też nie wspierają wnioskowania z wykorzystaniem reguł SWRL. Problemy te powodują, że powstające teraz ontologie mogą być nieaktualne względem tego, co będzie wymagane w przyszłości, gdy niuanse związane z obsługą pewnych struktur OWL zostaną rozwiązane.

Korzystanie z SWRL daje pewną przewagę nad użyciem samego języka OWL w praktycznych zastosowaniach. Możliwość skorzystania z działań arytmetycznych, operacji na właściwościach i swobodnego posługiwania się parametrami, które w specyfikacji OWL ograniczają sztywne zasady, otwiera drogę do przeprowadzania bardziej skomplikowanego wnioskowania i uogólniania. Ponadto wykorzystanie właściwości jest bardziej intuicyjne niż operacje na klasach opierające się na mechanizmach wprowadzania restrykcji.

Literatura

[1] Słownik języka polskiego. Wydawnictwo Naukowe PWN SA, `http://sjp.pwn.pl`. Dostęp: 2014-05-31.

[2] J. Jadacki. Co to jest i do czego służy semantyka. Spotkanie GLLI. Uniwersytet Opolski, Maj 2010.

[3] T. Zawadzka. *Integracja heterogenicznych źródeł wiedzy z wykorzystaniem logiki opisowej*. Praca doktorska, Politechnika Gdańska, Wydział Elektroniki, Telekomunikacj i Informatyki, 2008.

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

[5] W3C. OWL 2 Web Ontology Language Document Overview (Second Edition). `http://www.w3.org/TR/owl2-overview/`. Dostęp: 2014-06-4.

[6] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/SWRL/`. Dostęp: 2014-06-4.

[7] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007. Software Engineering and the Semantic Web.

[8] M. Kuba. OWL 2 and SWRL Tutorial. `http://dior.ics.muni.cz/~makub/owl/`. Dostęp: 2014-06-12.

[9] M. O'Connor. The Semantic Web Rule Language. *11th Intl. Protégé Conference*, Amsterdam, The Netherlands, Czerwiec 2009.

[10] Department of Computer Science, University of Aarhus. *An OWL/SWRL based Diagnosis Approach in a Pervasive Middleware*, 1993.

[11] L. Ding, P.M. Kolari, Z. Ding, S. Avancha. Using ontologies in the semantic web: A survey. R. Sharman, R. Kishore, R. Ramesh, redaktorzy, *Ontologies*, wolumen 14 serii *Integrated Series in Information Systems*, strony 79–113. Springer US, 2007.

[12] A. Mayr. Drupal Commerce: building Commerce Discount rules based on node categories. `http://www.agoradesign.at/blog/drupal-commerce-building-commerce-discount-rules-based-node-categories`, 2014. Dostęp: 2014-06-12.

[13] J. Breuker, A. Boer, R. Hoekstra, K. van den Berg. Developing content for lkif: Ontologies and frameworks for legal reasoning. *Proceedings of the 2006 Conference on Legal Knowledge and Information Systems: JURIX 2006: The Nineteenth Annual Conference*, strony 169–174, Amsterdam, The Netherlands, 2006. IOS Press.

[14] K. Saeed, T. Nagashima. *Biometrics and Kansei Engineering*. Springer Publishing Company, Incorporated, 2012.

[15] D. Beimel, M. Peleg. Using OWL and SWRL to represent and reason with situation-based access control policies. *Data & Knowledge Engineering*, 70(6):596 – 615, 2011.

ROZDZIAŁ 4

ROZPROSZONE BAZY WIEDZY

B. Kochanowski, M. Krupop

4.1. Wstęp

W związku z bardzo szybkim rozwojem technologii informacyjnych oraz rozległymi obszarami ich zastosowania pojawiła się potrzeba gromadzenia i komputerowego przetwarzania wiedzy. Aby było to możliwe wiedzę należało opisać w sposób formalny, a przy tym prosty i zrozumiały. Zaproponowano więc wiele języków służących do budowy dziedzinowych baz wiedzy oraz opracowano wiele sposobów przetwarzania zgromadzonej wiedzy w użyteczną informację [1].

Opracowanie modelu wystarczająco dobrze odzwierciedlającego rzeczywistość czy też zaproponowanie elastycznego sposobu reprezentacji wiedzy nie należą do zadań łatwych. Ich trudność wynika w dużej mierze ze złożoności procesów zachodzących w otaczającej nas rzeczywistości oraz licznych niuansów ujawniających się podczas ich studiowania. Dlatego podczas budowy modelu konceptualnego rozważane są zazwyczaj tylko wybrane fragmenty rzeczywistości, zaś sam model powstaje po przyjęciu znaczących uproszczeń. Co więcej, podczas modelowania dąży się do wyboru takiej metody reprezentacji, która pozwoli na automatyczne przetwarzanie danych przez maszyny bądź specjalne programy – komputerowych agentów [2]. Dzięki takiemu podejściu można tworzyć rozproszone bazy wiedzy i zapewnić ich automatyczną integrację na poziomie danych.

Bazy wiedzy kojarzy się ze zbiorami reguł postaci *jeżeli-to* (ang. *if-then*) lub grafowymi strukturami reprezentującymi wiedzę ekspertów z danej dziedziny. Bazy wiedzy są przetwarzane przez interpreter lub tzw. silnik wnioskowania (ang. *inference engine*), którego zadaniem może być np. sprawdzanie spójności lub dedukcja faktów.

W niniejszym rozdziale podjęto temat opracowania prostej, dziedzinowej bazy wiedzy agregującej informacje pochodzące z różnych źródeł. Jako obszar zainteresowania wybrano zagadnienia związane z produkcją piwa, zaś sam model danych został stworzony z wykorzystaniem języka OWL. Zaprojektowany model może zostać wykorzystany przez przedsiębiorstwa browarnicze zainteresowane stworzeniem opisu warzonego przez siebie asortymentu w postaci firmowej bazy wiedzy.

Rozdział rozpoczyna się analizą problemów związanych z modelowaniem baz wiedzy (w szczególności ontologii), po którym następuje krótkie omówieniem stosowanych w tym obszarze formalizmów. Rozdział kończy prezentacja autorskiej bazy wiedzy.

4.2. Metody formalne

4.2.1. Ontologie

W informatyce do reprezentacji wiedzy wykorzystuje się formalizm logiki opisowej oraz powstałe na jego bazie ontologie. Istnieje wiele definicji ontologii. W jednej z prostszych określa się ją jako „uporządkowany zbiór, składający się ze zbiorów obiektów oraz relacji pomiędzy nimi (binarnych i unarnych)" [2].

Z każdą ontologią można związać następujące pojęcia [3]:

- konceptualizację (model jest uogólnieniem pewnego wycinka rzeczywistości),
- konsensus (opisywana dziedzina musi być prawdziwa i zgodna z przekonaniami osób z nią związanych),
- ograniczoność (wynika z ograniczonego zakresu opisywanej wiedzy),
- formalizm (model jest zapisywany w języku formalnym, zrozumiałym przez ludzi i komputery),
- jawność (definiowanie pojęć z danej dziedziny i relacji między nimi odbywa się jawnie, z wykorzystaniem formalnych języków reprezentacji).

Za centralną część ontologii uważa się klasy. Dzięki nim możliwe jest kategoryzowanie pojęć charakterystycznych dla danej dziedziny czy domeny. Każda klasa może posiadać również swoje podklasy, za pomocą których można opisać dane pojęcia bardziej szczegółowo oraz tworzyć rozległe taksonomie. Obok klas istotną częścią ontologii są właściwości opisujące związki pomiędzy klasami i ich instancjami lub wartości ich atrybutów. Ponadto dodatkowymi elementami uzupełniającymi całość ontologii są ograniczenia.

Dzięki takiej budowie ontologie mogą służą jako pewnego rodzaju szkielet bądź słownik przeznaczony do definiowania konkretnych instancji danej klasy.

Proces tworzenia modelu ontologii wymaga zdefiniowania klas reprezentujących pojęcia z danej dziedziny w pewnej ustalonej hierarchii. Najpierw zaczyna się od pojęć najbardziej ogólnych, których podklasy będą reprezentować wyspecjalizowane ich podzbiory. Tworzenie modelu wieńczy zdefiniowanie właściwości lub atrybutów wraz z ograniczeniami. Na bazie modelu ontologii powstają bazy wiedzy zawierające deklaracje instancji klas zgodnych z zaprojektowanym modelem. Jak napisano w [4], granica pomiędzy pojęciem ontologii i bazą wiedzy nie jest ostra.

4.3. Języki formalnej reprezentacji ontologii

Bardzo ważną własnością języków używanych do opisu ontologii jest stopień ich formalności. Jak podają M. Uschold i M. Gruninger [5] stopień formalności określający dany język reprezentacji jest bardzo ściśle związany ze specyfiką pro-

cesu wymagającego przetwarzania ontologii. Przykładem może być komunikacja międzyludzka. Autorzy zaznaczają, iż w takim przypadku wykorzystywany jest nieformalny sposób reprezentacji ontologii. W wielu przypadkach jest on wystarczający, aby w poprawny sposób wyczuć intencję drugiej osoby. Zupełnie inna sytuacja ma miejsce w przypadku przetwarzania ontologii przez komputery lub wirtualnych agentów. Niestety nie posiadają oni tak rozbudowanej inteligencji jak ludzie, dlatego do poprawnego wnioskowania potrzebują sformalizowanej metody reprezentacji. Zatem formalny sposób reprezentacji jest zarezerwowany dla ontologii poddawanej komputerowemu przetwarzaniu. Taka postać reprezentacji jest najczęściej bardzo skomplikowana i trudna do interpretacji przez człowieka.

Języki do opisu ontologii ze względu na stopień ich formalizacji można podzielić na [5, 6]:

- języki nieformalne - brak jasnych reguł co do reprezentacji ontologii, a przez to duże prawdopodobieństwo wystąpienia niejasnego przekazu oraz wielu dwuznaczności,
- języki semi-nieformalne - bardziej ograniczona i ustrukturyzowana wersja języków nieformalnych,
- języki semi-formalne - stanowią grupę tzw. sztucznych języków reprezentacji, dostosowanych do wymagań języków formalnych,
- języki formalne - języki z bardzo szczegółowo zdefiniowaną semantyką, teorematami oraz dowodami.

Mówiąc o sposobach zapisu danych należy wspomnieć o tzw. metajęzyka. Jedna z najprostszych definicji określa metajęzyki jako: „języki służące do definiowania innych języków" [7]. Jednym z najpopularniejszych i często stosowanych metajęzyków jest XML oparty na wykorzystaniu znaczników pozwalających na reprezentację danych w postaci strukturalnej. Obecnie większość języków lub standardów służących do zapisu ontologii bazuje na XML [6]. Poniżej omówiono przykłady wybranych języków zapisu ontologii.

4.3.1. Język naturalny

W przypadku modelowania rzeczywistości z wykorzystaniem języka naturalnego wiedza reprezentowana jest za pomocą zdań lub różnego rodzaju wyrażeń. Bardzo dobrym przykładem takiego modelowania są instrukcje obsługi urządzeń. Dzięki zawartej tam wiedzy można łatwo dokonać szybkiej diagnozy wadliwie działającego urządzenia. Zaletą stosowania języka naturalnego jest możliwość bardzo szybkiego przekazu wiedzy. Niestety, język ten nadaje się tylko do opisu bardzo prostego modelu rzeczywistości. Bardziej złożone fragmenty rzeczywistości wymagają większej ilości struktur potrzebnych do ich zapisania.

Z reprezentacją wiedzy za pomocy języka naturalnego wiąże się jedno, bardzo istotne ograniczenie. Mianowicie jest to forma zrozumiała jedynie wśród osób jednakowo interpretujących dany język oraz symbole. Dlatego też reprezentacji wiedzy za pomocą języka naturalnego trudno uznać za uniwersalną formę reprezentacji [4].

4.3.2. Język logiki opisowej

Pojęcie logiki opisowej (ang. *Description Logic*, DL) jest kojarzone z rodziną formalizmów stosowanych do reprezentacji wiedzy (ang. *Knowledge Representation*, KR), w których wiedzę dziedzinową opisuje się definiując najpierw pojęcia z tej dziedziny (tj. pewną terminologię), a potem właściwości obiektów i indywiduów (tj. opis świata) korzystając z tych definicji. Użycie słowa „logika" wskazuje, że formalizm ten ma mocne oparcie w formalnej logice. Istotną cechą DL jest możliwość przeprowadzania wnioskowania, w którym na bazie istniejącej wiedzy dedukuje się nowe fakty.

W architekturze systemów DL wyróżnić można trzy główne komponenty: język opisu, algorytm (silnik) wnioskowania, bazę wiedzy. Język opisu pozwala na reprezentację wiedzy o danej dziedzinie za pomocą następujących elementów: pojęć (ang. *concepts*), indywiduów (ang. *individuals*) oraz ról (ang. *roles*). Pojęcia kojarzone są z klasami grupującymi jakieś obiekty o wspólnych cechach. Z kolei indywidua są konkretnymi przykładami takich obiektów. Natomiast role reprezentują binarne relacje występujące pomiędzy indywiduami [6]. Stąd w bazie wiedzy (ang. *Knowledge Base*, KB) wyróżnia się następujące części [8]:

- TBox – terminologię (ang. *terminology*), nazywaną czasem słownikiem dziedzinowym. Słownik ten reprezentuje zestaw pojęć, które można podzielić na: pojęcia pierwotne oraz pojęcia zdefiniowane poprzez określony zestaw osobników i ról (relacji między osobnikami). Obiektami TBox są klasy, dla których wyznaczane są pewne dopuszczalne reprezentacje i na których można zastosować mechanizmów wnioskowania.
- ABox – opis świata reprezentowany przez zbiór asercji (ang. *assertions*). Elementami tego zbioru są indywidua opisane za pomocą terminów ze słownika. Podobnie jak TBox opis ten ogranicza liczbę dopuszczalnych interpretacji dla swoich obiektów.
- RBox – informacje o rolach i ich zależnościach (czasami ta część nie jest wyróżniana w literaturze).

Formalnie konceptom opisanym w języku DL dostarcza znaczenia tzw. interpretacja. Do podstawowych problemów wnioskowania w DL zalicza się:

- równoważność (ang. *equivalence*) (pojęcia C i D są równoważne, jeśli zbiory ich wystąpień są zawsze takie same);
- zawieranie się (ang. *subsumption*) (pojęcie C zawiera się w pojęciu D, jeśli zbiór wystąpień pojęcia C jest zawsze podzbiorem zbioru wystąpień pojęcia D);
- spełnialność lub inaczej niesprzeczność (ang. *satisfiability*, *non-contradictory*) (pojęcie C jest spełnialne, jeśli może mieć wystąpienia);
- spójność zbioru asercji (ang. *consistency*) (badane jest, czy istnieje model i czy z asercji w ABox wynika, że dane indywiduum jest instancją danego opisu pojęcia, sprawdzanie spełnialności opisów oraz spójności zbioru asercji są użyteczne przy ocenie czy baza wiedzy jest w ogóle sensowna);
- rozłączność (ang. *disjointness*) (pojęcia C i D są rozłączne, jeśli ich zbiory wystąpień są zawsze rozłączne).

Zdania w logice opisowej można reprezentować używając różnych symboli. Do najczęściej stosowanych sposobów zapisu tych zdań należy sposób zilustrowany poniższym przykładem:

$$\text{Person} \sqcup \exists \text{hasChild}.\top$$

Znaczenie tego zdania jest następujące: „Osoby posiadające dziecko".

4.3.3. Formalizmy obiektowe

Formalizmy obiektowe pozwalają na reprezentowanie wiedzy z danej dziedziny w postaci wirtualnych modeli, odzwierciedlających rzeczywiste obiekty lub ich zachowania. Najczęściej modelowaniem obejmowane są struktury klas. Określają one szereg właściwości lub atrybutów charakterystycznych dla wszystkich obiektów należących do danej klasy. Z klasami wiąże się też hierarchia: każda klasa może posiadać zbiór podklas dziedziczących jej cechy oraz być klasą potomną klasy nadrzędnej; klasy uogólniają pewne zbiory obiektów rzeczywistych, przy czym klasy pochodne są specjalizacjami klas pierwotnych. W modelowaniu obiektowym występuje również pojęcie instancji klasy. Instancjami są konkretne obiekty o właściwościach i atrybutach zgodnych ze wzorcem, z którego powstały.

Przykładem języka wspierającego modelowanie obiektowe jest UML (ang. *Unified Modeling Language*). Pozwala on na graficzną reprezentację klas w postaci diagramów oraz zobranowanie statycznych związków, jakie występują między klasami. Niestety język UML nie spełnia wymogów co do formalnego sposobu reprezentacji ontologii [6]. Brak mu mianowicie formalnej semantyki, czyli szczegółowo zdefiniowanych reguł do zapisu ontologii.

4.3.4. Grafy konceptualne

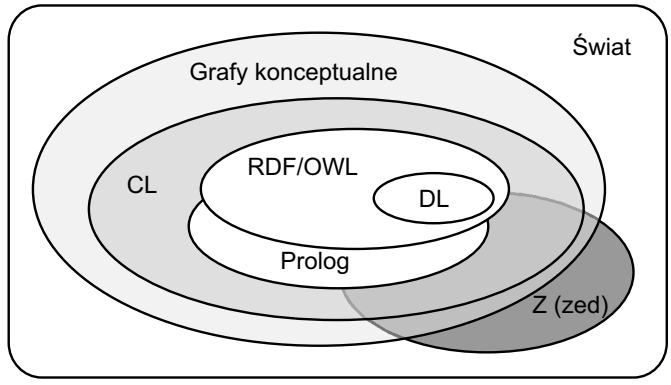
Idea grafów konceptualnych powstała na bazie grafów Pierce'a (od nazwiska amerykańskiego logika Charlesa S. Pierce'a) oraz koncepcji sieci semantycznych. Grafy te dostarczają większej siły ekspresji niż logika predykatów [6]. Zbudowane modele można wymieniać dzięki standardowi wymiany (ang. *Conceptual Graphs Interchange Format*, CGIF) opracowanemu razem ze standardem ISO/IEC IS 24707:2007 (*Information technology — Common Logic (CL): a framework for a family of logicbased languages*). Na rysunku 4.1 przedstawiono miejsce grafów konceptualnych na tle innych języków i formalizmów.

Grafy konceptualne są bardzo wygodnym narzędziem pozwalającym na przetwarzanie/modelowanie języka naturalnego. Na przykład zdanie „John jedzie autobusem do Bostonu" (ang. *John is going to Boston by bus*) można reprezentować graficznie jak na rysunku 4.2. Zdanie to można też zapisać w formacie CGIF:

```
[*x] [*y]
(Go ?x) (Person John) (City Boston) (Bus ?y)
(Agnt ?x John) (Dest ?x Boston) (Inst ?x ?y)
```

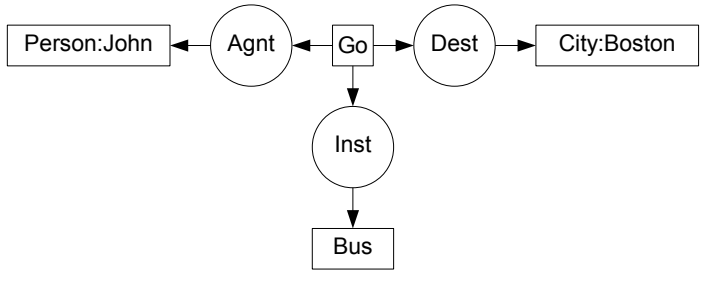
lub równoważnie w notacji CLIF (ang. *Common Logic Interchange Format*):

```
(exists (x y)
   (and (Go x) (Person John) (City Boston) (Bus y)
        (Agnt x John) (Dest x Boston) (Inst x y)))
```



Rys. 4.1: Miejsce grafów konceptualnych wg [9].

Prostokąty występujące na rysunku 4.2 przedstawiają istniejące obiekty lub zdarzenia z pewnego fragmentu opisywanej rzeczywistości. Obiekty te w terminologii grafów konceptualnych zostały nazwane konceptami lub pojęciami. Natomiast występujące na rysunku okręgi określają relacje zachodzące pomiędzy tymi obiektami. W prezentowanym przykładzie `Agnt` oznacza agenta czyli głównego wykonawcę czynności, `Inst` określa narzędzie wykorzystane przy danej czynności oraz `Dest` określa jej cel [10].



Rys. 4.2: Przykład grafu konceptualnego [10] wg ISO/IEC IS 24707:2007.

Sieci semantyczne służą do formalnej reprezentacji pewnego wąskiego fragmentu rzeczywistości. Wykorzystują one dwa rodzaje obiektów. Do pierwszego należą koncepty reprezentujące pewne fizyczne byty, atrybuty lub zdarzenia charakterystyczne dla opisywanej dziedziny. Do drugiego zaliczane są relacje wyrażające zależności między obiektami. Związki między obiektami można określić mianem semantyki grafów konceptualnych. Opisywany język reprezentacji wykorzystuje również elementy logiki, które wyznaczają czy określony koncept jest elementem opisywanej rzeczywistości czy też nie. W tym celu używane są kwantyfikatory: ogólny oraz egzystencjalny, a także operatory: negacji, alternatywy i koniunkcji [10, 6].

4.3.5. RDF i OWL

Standard RDF (ang. *Resource Description Framework*) został opracowany z myślą o maszynowym przetwarzaniu wiedzy oraz generowaniu semantycznych opisów (metadanych) zasobów opublikowanych w sieci web. Jest używany jako ogólna metoda do tworzenia modeli konceptualnych oraz ich zapisu z wykorzystaniem różnych sposobów notacji oraz serializacji danych. Główna idea RDF zakłada możliwość opisu pewnego bardzo wąskiego fragmentu rzeczywistości w !postaci tzw. trójek lub trypletów [11]:

- podmiot/klasa - określa zasoby czyli pewne obiekty lub zdarzenia występujące w modelowanej dziedzinie,
- predykat - może być interpretowany jako pewne własności lub atrybuty danego obiektu,
- obiekt - jest to wartość danego predykatu lub inaczej wartość określonego atrybutu obiektu.

Standard OWL (ang. *Web Ontology Language*) zbuwano na RDF, przy czym oparto go na semantyce logiki opisowej. Dlatego też składnia OWL jest bogatsza, większy też jest słownik oraz siła ekspresji języka. Zarówno RDF jak i OLW (oraz OWL2 - nowsza wersja standardu OWL) należą do tzw. stosu technologii sieci semantycznych web opisanych w rozdziale 3

4.4. Budowa modelu bazy wiedzy w praktyce

Celem, jaki postawiono sobie przy opracowywaniu tematu niniejszego rozdziały było skonstruowanie własnej bazy wiedzy w oparciu o dostępne dziedzinowe bazy wiedzy gromadzące informacje o różnych stylach piwa.

Na etapie analiz za pomocą wyszukiwarki *Swoogle* znaleziono bazową ontologię dziedzinową dostarczającą słownik do wykorzystania podczas modelowania. Dysponując tym słownikiem można było stworzyć własne opisy faktów. Wystarczyło zaimportować ten słownik do własnej ontologii, aby ją później uzupełniać o opisy własnych instancji. Zaletą przyjętego rozwiązania było wykorzystanie architektury rozproszonej. Baza główna mogła znajdować się w dowolnym miejscu w sieci, wystarczyło, że była dostępna. Wykorzystując tę bazę każda firma produkująca piwo mogłaby stworzyć własną bazę wiedzy, opisującą własne wyroby, oraz uzupełnić ją instancjami danych stylów piwa. Dysponując potrzebnymi elementami, czyli słownikiem oraz bazami firmowymi zbudowanymi z wykorzystaniem tego samego słownika można było przystąpić do integracji rozproszonych informacji w jednym miejscu.

4.4.1. Założenia

Zadanie budowy własnej bazy wiedzy można było zrealizować na dwa sposoby. Pierwszym sposób polegał na konstruowaniu zapytań w języku SPARQL z poziomu programu zarządzającego zintegrowaną już bazą wiedzy (tj. bazą, która powstała poprzez zaimportowanie do niej poszczególnych grafów) i uzy-

skiwaniu w ten sposób potrzebnych informacji o właściwościach poszczególnych stylów piwa oraz ich instancjach. Drugi ze sposobów polegał na zadawaniu pytań do rozproszonych baz wiedzy i integrowaniu odpowiedzi (zwracanych grafów) w jednej strukturze na bieżąco (tj. przy każdym zapytaniu).

4.4.2. Wykorzystane narzędzia

W celu realizacji postawionego zadania wykorzystano następujące narzędzia:

- Protege(`www.http://protege.stanford.edu/`) – edytor służący do tworzenia i modyfikacji ontologii,
- Apache Jena (`www.http://jena.apache.org/`) – API umożliwiające operowanie na bazach wiedzy, ich agregację, wyświetlanie potrzebnych informacji,
- Swoogle (`www.http://swoogle.umbc.edu/`) - wyszukiwarka pozwalająca na znalezienie dokumentów semantycznego Weba na podstawie słów kluczowych; dodatkowo z dokumentów tych wyodrębniane są metadane co służy do określenia relacji między nimi [7].

4.4.3. Zaprojektowana baza wiedzy

Do obsługi bazy wiedzy stworzono program Semantic Beer Base, który umożliwia wczytywanie słownika oraz baz firmowych zawierających instancje różnych stylów piwa. Program, za pomocą Jeny łączy poszczególne ontologie firmowe w jeden graf, a następnie wyświetla wszystkie te instancje, dodając właściwości reprezentowanego przez nie stylu piwa. Informacje te wyświetlane są za pomocą struktury drzewiastej. Przykład działania aplikacji, wyświetlającej wczytane informacje przedstawiono na rysunku 4.3.

Sam model bazy wiedzy nie był skomplikowany. Zaprojektowana ontologia miała służyć do reprezentacji najważniejsze pojęć związanych z produkcją piwa. Występowały w niej następujące klasy: *piwo*, *style piwa*, *składniki*, *browary*, *wydarzenia* i inne. Klasa *składniki* miała klasy potomne reprezentujące konkretne typy składników: *zboża*, *chmiel*, *słód*, *drożdże*, *woda* itp. Specjalizacjami klasy *zboże* były *jęczmień* oraz *pszenica*. Warto zauważyć, że do organizacji klas w tej ontologii użyto pojęć ze słownika zewnętrznego, w którym klasa *piwo* jest podklasą klasy *napój alkoholowy*, a klasa *browar* podklasą klasy *organizacja*.

Przykładem użycia ontologi może być klasa stylu piwa *Lager*, posiadająca właściwość *madeFrom* służącą do określania składników potrzebnych do uwarzenia danego stylu piwa. Zgodnie z informacjami z istniejącej ontologii do uwarzenia piwa w stylu *Lager* potrzeba chmielu *Saaz* oraz słodu *Pils*.

W ontologii uwzględniono również ograniczenia. Przykładowo, składnikiem potrzebnym do uwarzenia danego stylu piwa nie może być klasa *browar*, *wydarzenie*, czy inny *styl piwa*. Tworzenie ograniczeń jest szczególnie użyteczne w przypadku, gdy pewnym własnościom danej klasy można przypisać jakieś konkretne przedziały wartości. Ograniczenia chronią definiowaniem niepoprawnych wartości dla instancji określonej klasy [4].

Rys. 4.3: Widok działającej aplikacji.

Podczas działania program wyświetla instancje piw ontologii firmowych i dla każdej z nich dodaje nazwę stylu, który reprezentuje dana instancja. Możliwe jest również wyświetlenie atrybutów przypisanych danym stylom. W tym przypadku są to składniki, z których można uwarzyć piwo. Informacje te pobierane są z głównej ontologii, a dokładniej, ze wspomnianego atrybutu *madeFrom*.

Użytkownik może dodawać własne ontologie, które powstały na podstawie głównej ontologii piwnej. Dodając, a następnie wyświetlając tę ontologię, w głównym drzewie pojawi się kolejną gałąź odpowiadającą danej ontologii kompanii piwowarskiej. W tej gałęzi dodane zostaną kolejne instancje tej ontologii. Z kolei dla każdej z tych instancji wyróżniona zostanie klasa *styl piwa*, do której należą. Wykorzystując nazwę *stylu piwa* oraz informacje zawarte w głównej ontologii, program dla każdej klasy wyświetla własność jej przypisaną. Własnością tą są składniki potrzebne do uwarzenia piwa w danym stylu.

Na rysunku 4.3 zaznaczono poszczególne elementy składowe drzewa. Jedynką oznaczono nazwę kompani piwowarskiej, dwójką - nazwę instancji. Element oznaczony numerem 3 to nazwa klasy *stylu piwa*, natomiast element oznaczony numerem 4 to jeden ze składników. W tym przypadku jest to składnik *KentGoldings*), pozostałymi składnikami są zaś *Pale*, *Chocolate*, *Black* oraz *Munich*.

Literatura

[1] G. Paquette. *Visual Knowledge Modeling for Semantic Web Technologies: Models and Ontologies*. Information Science Reference, 2010.

[2] W. Gliński. Ontologie. próba uporządkowania terminologicznego chaosu. *Od informacji naukowej do technologii społeczeństwa informacyjnego: praca zbiorowa / pod red. Barbary Sosińskiej-Kalaty i Marii Przastek-Samokowej*. Wydawnictwo SBP, Warszawa, 2005.

[3] M. Roszkowski. Do czego potrzebne są nam ontologie? charakterystyka funkcjonalna ontologii jako narzędzi reprezentacji wiedzy, 2013.

[4] N. Noy, D. McGuinness. What is an ontology and why we need it? Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[5] M. Uschold, M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–136, 1996.

[6] A. Sobczak. Analiza wybranych języków stosowanych do budowy ontologii. *Studia i Materiały Polskiego Stowarzyszenia Zarządzania Wiedzą*, number 7. Polskie Stowarzyszenie Zarządzania Wiedzą, Warszawa, 2006.

[7] W. Gliński. Języki i narzędzia do tworzenia i wyszukiwania ontologii w kontekście semantycznego weba. *Od informacji naukowej do technologii społeczeństwa informacyjnego: praca zbiorowa / pod red. Barbary Sosińskiej-Kalaty i Marii Przastek-Samokowej*. Wydawnictwo SBP, Warszawa, 2005.

[8] J. Józefowska. Ontologie. logiki deskrypcyjne. `http://www.cs.put.poznan.pl/jjozefowska/wyklady/ai/Ontologie.pdf`.

[9] H. Delugach. Common Logic in support of metadata and ontologies. *ISO/IEC JTC1 SC32 Open Forum*, Berlin, Germany, Kwiecień 2005.

[10] J.F. Sowa. Conceptual graphs. *Handbook of Knowledge Representation*. Elsevier, 2008.

[11] World Wide Web Consortium. *RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation*, 2004.

ROZDZIAŁ 5

INTERFEJSY KOMUNIKACJI CZŁOWIEK – KOMPUTER

A. Klama, M. Kotyla

Odkąd informatyzacji i komputeryzacja zagościły na ścieżce rozwoju ludzkiej cywilizacji kluczowym zagadnieniem stało się opracowywanie metod zapewniających sprawną i wygodną interakcję pomiędzy człowiekiem a komputerem. Od lat naukowcy i inżynierowie próbują znaleźć odpowiedź na pytanie jak w sposób nieskomplikowany a zarazem efektywny komunikować się z urządzeniami? Odpowiedź na to pytanie tylko na pozór wydaje się prosta. W budowie urządzeń wejściowych można wykorzystać różne kanały porozumiewania się: od komunikatów tekstowych, poprzez urządzenia wskaźnikowe aż do urządzeń reagujących na ruch. Podobnie wiele opcji można wykorzystać w budowie urządzeń wyjściowych, choć najczęściej przekazywane są tam dźwięki i obrazy.

W niniejszym rozdziale dokonano przeglądu rozwiązań używanych do komunikacji pomiędzy człowiekiem i komputerem oraz zaprezentowano pomysł na nowy sposób przeglądania folderów z użyciem kontrolera *Razer Hydra*.

5.1. Przegląd dostępnych rozwiązań

5.1.1. Urządzenia wejściowe

Urządzenia wejściowe służą człowiekowi do przekazywania komunikatów i danych do komputera. Poniżej dokonano krótkiego ich przeglądu.

Mysz komputerowa jest powszechnie znanym narzędziem używanym niemal z każdym komputerem. Zasada jej działania jest niezwykle prosta – przesuwając urządzenie po małym, płaskim obszarze obserwowany jest ruch wskaźnika na ekranie. Urządzenie wyposażone jest w przynajmniej dwa przyciski (pierwsze myszy były nawet jednoprzyciskowe). Lewy przycisk odpowiada za wybór, prawy zaś pozwala na otwieranie menu lub na rozpoczęcie akcji – jak ma to miejsce w niektórych grach. Myszy mogą być wyposażone w dodatkowe przyciski rozszerzające ich funkcjonalność oraz rolkę, służącą do przewijania.

A jak sprawdza się, o ile mysz została przesunięta? Informacji tej dostarcza

sensor zainstalowany na spodzie urządzenia. Sensor ten może mieć różną konstrukcję, a dane o ruchu mogą być pozyskiwane za pośrednictwem:

- kulki osadzonej w gnieździe z rolkami (mysz kulkowa),
- diody elektroluminescencyjnej i elementów światłoczułych (mysz optyczna),
- diody laserowej i elementów światłoczułych (mysz laserowa).

Najdokładniejsza z wymienionych jest mysz laserowa. Mysz tego typu działa niemal na każdej powierzchni, ma najwyższą rozdzielczość oraz nie posiada sensorów mechanicznych, które łatwo ulegają zabrudzeniom.

Klawiatura komputerowa to urządzenie pozwalające na przekazywanie komputerowi sygnałów odpowiadających wciśniętym klawiszom. W układzie klawiatury wyróżnia się następujące rodzaje klawiszy: alfabetyczne, numeryczne, odpowiadające znakom specjalnym i funkcjom oraz klawisze programowalne – ich funkcja zostaje wybrana przez użytkownika. Powszechnie stosowane układy klawiatur to: QWERTY (najpowszechniejszy), QWERTZ, AZERTY, Dvoraka (zoptymalizowany pod kątem szybkiego pisania). Najczęściej klawiatura służy do wprowadzania tekstu.

W jaki sposób komputer rozpoznaje, który klawisz został naciśnięty? Naciśnięcie klawisza powoduje przesłanie pary sygnałów (ang. *scancode*) do komputera, który potrafi je zinterpretować. Sygnały te mogą być przesłane do komputera za pomocą: przewodu – obecny standard to USB (dawniej PS/2), podczerwieni – klawiatura musi mieć kontakt z odbiornikiem, fal radiowych – zasięg około 5 metrów (technologia bluetooth).

Konstrukcja mechaniczna klawiatur również jest bardzo zróżnicowana. Wyróżniamy klawiatury: mechaniczne – stosowane w maszynach do pisania, stykowe – klawisz powoduje zwarcie: sprężynowe, membranowe – membrana oddziela obwody drukowane w czasie, gdy klawisz nie jest naciśnięty, z gumą przewodzącą – dociśnięcie gumy powoduje obniżenie rezystancji, bezstykowe: optoelektroniczne – klawisz wsuwa/wysuwa przesłonę transoptora, pojemnościowe – zmiana pojemności kondensatora, kontaktronowa – klawisz przysuwa magnes do kontaktronu powodując zwarcie, ekranowe: klasyczna – klawiatura narysowana na ekranie, wybór klawisza za pomocą urządzenia wskazującego (myszy), dotykowa – klawiatura narysowana na ekranie, klawisz wybierany dotknięciem.

Joystick to urządzenie wejściowe składające się z drążka obracanego na podstawie. Zmiana kąta i położenia drążka powoduje wygenerowanie sygnału odpowiedniego do ruchu. Joysticki wyposażone są również w przyciski, zapewniające im dodatkową funkcjonalność.

Wyróżniamy dwa rodzaje joysticków: analogowe – pozwalają na ruch w dowolnym kierunku (siła z jaką wychyla się drążek może mieć znaczenie) oraz cyfrowe – pozwalają na detekcję ruchu w ośmiu kierunkach (cztery podstawowe i cztery pośrednie).

Joysticki są często stosowane w lotnictwie, grach komputerowych i przy sterowaniu maszynami (np. wózkami inwalidzkimi).

Trackpoint to mały joystick. Występuje w postaci gumowego kapturka zamontowanego w klawiaturze komputerowej (zwykle w laptopach pomiędzy kla-

wiszami H, G oraz B). Jest to urządzenie wskazujące, podobnie do myszy, trackballa czy touchpada. W zależności od producenta bywa również nazywane: Touchstick, Pointstick, Trackstick, FineTrack, AccuPoint, VectorPad, NX Point.

Gamepad to urządzenie sterujące stosowane w grach. Posiada zestawy przycisków: z prawej strony znajdują się przyciski akcji, zaś po lewej stronie – przyciski odpowiadające za ruch (zwykle jest to jeden przycisk w kształcie krzyża).

Kamera internetowa przekazuje obraz w czasie rzeczywistym do komputera (najczęściej za pośrednictwem złącza USB) lub sieci komputerowej. Może być to urządzenie zewnętrzne lub wbudowane (spotykane w laptopach).

Trackball to urządzenie wskazujące, składające się z kulki umieszczonej w gnieździe wyposażonym w czujniki wykrywające ruch w dwóch osiach – podobnie jak mysz kulkowa. Użytkownik przesuwając kulkę determinuje ruch wskaźnika. W przeciwieństwie do myszy trackball nie wymaga podnoszenia i przestawiania w chwili, gdy skończy się powierzchnia robocza.

Mikrofon to sensor przekształcający dźwięk na sygnał elektryczny przekazywany do komputera. Wyróżniamy różne rodzaje mikrofonów:

- pojemnościowe – jako jedna z płyt kondensatora działa w nich membrana, a drgania powodują zmiany odległości pomiędzy płytami (zmienia się pojemność);
- dynamiczne – działają na zasadzie indukcji elektromagnetycznej: poruszona membrana powoduje ruch małych cewek umieszczonych w polu magnetycznym, a co za tym idzie, zmianę prądu indukcyjnego w cewce;
- wstęgowe – charakteryzuje je użycie falistej metalowej wstęgi zawieszonej w polu magnetycznym, jej drgania wytwarzają sygnał elektryczny;
- węglowe – wykorzystuje się w nich granulat węglowy, znajdujący się pomiędzy dwoma metalowymi płytami, drgania powodują zmiany nacisku płyt na granulat, co implikuje zmianę jego rezystancji, a więc zmianę przepływającego prądu;
- piezoelektryczne – wykorzystują zdolność niektórych materiałów do wytwarzania napięcia w chwili, gdy następuje zmiana ciśnienia, dzięki temu wibracje zamieniane są na sygnał elektryczny;
- laserowe – wiązka laserowa skierowana jest w nich na okno lub inną drgając pod wpływem dźwięku powierzchnię, drgania powierzchni oraz zmiana kąta odbicia wiązki są wykrywane i przetwarzane na sygnał audio;
- światłowodowe – wykrywają zmiany w natężeniu światła i przetwarzają je na sygnał elektryczny;
- wodne – mają nadajnik w postaci metalowego kubka wypełnionego wodą z dodatkiem kwasu siarkowego. Fala akustyczna powoduje poruszanie w górę i w dół w cieczy, a ponieważ rezystancja elektryczna pomiędzy przewodem a kubkiem jest odwrotnie proporcjonalna do wielkości menisku wytworzonego wokół igły, powstają sygnały elektryczne;
- MEMS – mikrofony mikromechaniczne, składają się z małych elementów krzemowych, które są stabilne i powtarzalne, charakteryzują się niskimi szumami oraz niskim poborem mocy;

- ECM – mikrofony elektretowe, wykorzystuje się w nich materiał ze stałym ładunkiem elektrycznym jako jedną z okładzin kondensatora.

Ekran dotykowy to urządzenie wizualne pozwalające na kontrolowanie komputera za pomocą prostych jak i bardziej skomplikowanych gestów. Gesty są wykonywane za pomocą specjalnych wskaźników lub palców. Ze względu na technologię wykrywania dotyku ekrany te dzielimy na:

- rezystancyjne – działające na zasadzie zmiany oporności pod wpływem siły nacisku;
- SAV (ang. *surface acoustic wave*) – po dotyku część fali akustycznej jest absorbowana przez ekran;
- pojemnościowe – ekran składa się z izolatora (szkło) pokrytego przezroczystym przewodnikiem. Ponieważ ludzkie ciało również jest przewodnikiem, dotyk powoduję zmianę pojemności i dlatego ruch zostaje wykryty. Nie działają poprzez materiały izolujące takie jak rękawiczki;
- podczerwone – dotyk powoduje przerwanie strumienia światła podczerwonego emitowanego przez sieć diod LED zamontowanych w obudowie.

Obecnie trwają prace nad ekranami haptycznymi, które na skutek dotyku zmieniają swoje właściwości odczuwalnie dla użytkownika.

Monitor braillowski służy zarówno jako urządzenie wyjściowe jak i wyjściowe dla osób niewidomych. Jest to podłużna tablica składająca się z „ramek" znakowych. W każdej ramce znajduje się 6 lub 8 otworów z ruchomymi elementami, które mogą być wypukłe lub płaskie. Za pomocą tych elementów (a dokładniej ich ułożenia) informacja jest przekazywana do komputera.

Touchpad to urządzenie wskazujące działające na podstawie wbudowanego w powierzchnię wyspecjalizowanego sensoru dotykowego. Sensor pozwala na przełożenie ruchu i położenia palców na względną pozycję w systemie operacyjnym (w przypadku komputera – na pozycję wskaźnika na ekranie).

Tablet graficzny, zwany również digitizerem, to urządzenie umożliwiające użytkownikowi rysowanie na ekranie w podobny sposób jak przy użyciu kartki i ołówka. Urządzenie składa się z (płaskiej) podkładki oraz ze specjalnego urządzenia przypominającego pióro.

Kierownica to kontroler używany zwykle do sterowania komputerem podczas grania w wyścigowe gry komputerowe. Zastępuje myszkę oraz klawiaturę. Zwykle występuje w zestawie z podnóżkiem z pedałami. Zapewnia duży zakres ruchów. Bywa wyposażona w siłowe sprzężenie zwrotne, które pozwala na urealnienie gry.

Pióro świetlne to mało popularne urządzenie wejściowe. Kształtem przypomina długopis podłączony do komputera. Końcówka pióra to światłoczuły element, który wykrywa światło i w ten sposób lokalizuje położenie kursora. To urządzenie działa na podstawie rejestrowania częstotliwości pracy monitora, zatem nie nadaje się do monitorów LCD.

Razer Hydra to kontroler ruchu i orientacji stosowany w grach komputerowych. Do wykrywania położenia i orientacji wykorzystuje słabe pole magnetyczne. Jego dokładność to 1 mm/1^o. Ma sześć stopni swobody. Obecnie dostępna wersja jest przewodowa, trwają prace nad wersją bezprzewodową.

Kinect to urządzenie zastępujące kontroler gier. Pozwala na interakcję z urządzeniem za pomocą gestów oraz komend głosowych bez wykorzystywania dodatkowych urządzeń. Składa się z dwóch kamer (RGB – do rozpoznawania twarzy i kolorów, druga zwraca informację o głębokości), promiennika podczerwieni oraz macierzy czterech mikrofonów. Promiennik podczerwieni wyświetla przed urządzeniem punkty, które są wykrywane przez kamerę głębi oraz przetwarzane do rozdzielczości obrazu z kamery RGB. Czujnik działa w zakresie 0,4–6,5 m. Technologia ta jest wrażliwa na zbyt duże nasłonecznienie (w zbyt dużym nasłonecznieniu przestaje działać prawidłowo).

5.1.2. Urządzenia wyjściowe

Urządzenia wyjściowe to urządzenia za pomocą których komputer przekazuje komunikaty lub dane człowiekowi. Poniżej dokonano przeglądu standardowych rozwiązań spotykanych na rynku.

Monitor lub wyświetlacz to urządzenie wyjściowe, przekazujące wizualne komunikaty od komputera. Obecnie królują monitory ciekłokrystaliczne (TFT-LCD), które są stosunkowo cienkie. Dawniej używano monitorów kineskopowych (CRT), które były ciężkie i grube. Monitory komputerowe charakteryzują się różnymi rozmiarami określanymi za pomocą przekątnej podawanej w calach, rozdzielczości oraz proporcji ekranu.

Monitor braillowski służy zarówno jako urządzenie wyjściowe jak i wyjściowe dla osób niewidomych (opis jak przy urządzeniach wejściowych).

Projektor to urządzenie optyczne służące do wyświetlania na ekranie projekcyjnym obrazu z komputera. Składa się ze źródła światła oraz i układu optycznego formującego wiązkę światła.

Drukarka to urządzenie wyjściowe pozwalające nanieść na papier pewne dane (widoczne na ekranie komputera lub specjalnie sformatowane). Drukarki mogą być używane przez wielu użytkowników w sieci lokalnej (choć w domowych zastosowaniach częściej podłączane są bezpośrednio do komputera lub nawet bez podłączenia do komputera – gdy materiał do drukowania pobierany jest bezpośrednio z kart pamięci).

Ze względu na możliwości druku w kolorze drukarki dzielone są na: kolorowe oraz monochromatyczne (czarno-białe). Drukarki można również podzielić ze względu na technologię wydruku. Wyróżniamy drukarki:

- igłowe – niegdyś bardzo popularne, do drukowania wykorzystują taśmę barwiącą, są tanie w eksploatacji i dają możliwość wydruku wielu kopii na papierze samokopiującym;
- atramentowe – drukowanie polega w nich na umieszczaniu na papierze maleńkich kropel atramentu. Wydruki kolorowe powstają z czterech podstawowych pojemników z atramentem – cyan, magenta, czarnego oraz żółtego(CMYK). Drukarki te umożliwiają druk w kolorze białym. Koszty eksploatacji tego typu drukarek są wysokie;
- laserowe – druk odbywa się w nich poprzez naniesienie na papier cząsteczek toneru. Wałek selenowy jest elektryzowany i naświetlany, przez co traci ładunek

elektryczny i nie przyciąga cząsteczek tonera. Następnie toner jest przenoszony na papier, rozgrzewany i wprasowywany w kartkę.

Ploter to urządzenie pozwalające na wydruk wielkoformatowy. Oprócz tego umożliwiają grawerowanie, wycinanie laserem, kreślenie map.

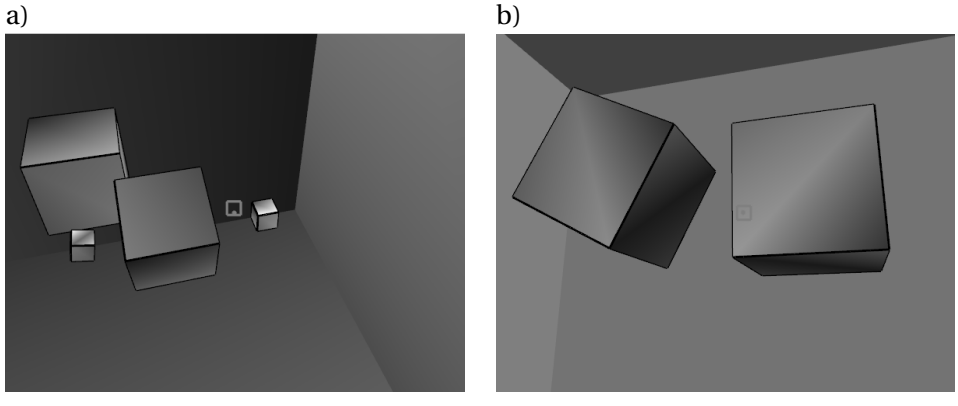
Głośniki to urządzenia peryferyjne przekazujące dźwięk z komputera. Zwykle podłączane są do urządzenia za pomocą złącza minijack 3,5 mm.

5.2. Praktyczne wykorzystanie kontrolera Razer Hydra

W niniejszym podrozdziale opisano sposób wykorzystania kontrolera Razer Hydra do poruszania się w trójwymiarowym wirtualnym świecie oraz manipulacji obiektami tam umieszczonymi.

5.2.1. Struktura wirtualnego świata

Wirtualny świat jest reprezentowany jako przestrzeń zagnieżdżających się pudełek. Wewnątrz głównego pudełka znajdują się mniejsze, a wewnątrz nich – kolejne pudełka (rys. 5.1). Pudełka znajdujące się wewnątrz świata mogą być dowolnie przybliżane, oddalane, przesuwane, chowane jedno w drugie. Wszystkie te operacje można wykonać za pomocą kontrolera Razer Hydra.



Rys. 5.1: Przykładowy stan wirtualnego świata a) pudełko podstawowe, b) wnętrze wybranego wirtualnego pudełka.

Strukturę wirtualnego świata można porównać do struktury drzewiastej, na bazie której tworzone są systemy katalogów i plików w komputerowych systemach operacyjnych: system operacyjny → foldery → pliki. Taka struktura nadaje się doskonale do organizacji różnych magazynów danych. Na jej bazie można zbudować nowe interfejsy menadżerów plików, przenosząc zarządzanie plikami i katalogami z przestrzeni dwuwymiarowej do trójwymiarowej. Tak też właśnie uczyniono w opisywanym projekcie.

Wizualizację wirtualnego świata zaimplementowano za pomocą biblioteki OpenGL (`http://www.opengl.org/`), zaś obsługę kontrolera - za pomocą do-

starczonych bibliotek i autorskiego kodu. Program został napisany z wykorzystaniem bibliotek Qt 5.2.1 oraz kompilatora MSVC2010. Do opracowania metody interpretacji zmian stanów kontrolera skorzystano z prac [1, 2].

5.2.2. Kontroler Razer Hydra

Kontroler Razer Hydra (rys. 5.2) jest urządzeniem wejściowym, które stosunkowo niedawno pojawiło się na rynku. Pozwala ono na poruszanie się w wirtualnej przestrzeni 3D oraz wydawanie poleceń za pomocą gestów. Producent dostarcza do niego oprogramowanie Sixense (`http://sixense.com/hardware/sixensesdk`).

Rys. 5.2: Kontroler Razer Hydra.

Razer Hydra składa się z bazy oraz dwóch pałeczek-kontrolerów. Pałeczki-kontrolery dają łącznie 12 stopni swobody. Baza jest traktowana jako punkt odniesienia dla kontrolerów (środek świata znajduje się tam, gdzie baza, tj. pomiędzy dwoma kontrolerami). Baza jest połączona z komputerem za pomocą złącza USB. Jeżeli użytkownik pomyli się i odłoży kontrolery odwrotnie na miejsce (prawy z lewym zostaną zamienione miejscami), to po każdym takim odłożeniu następuje ponowna kalibracja urządzenia. Każdy kontroler posiada przyciski oraz joystick. Można im programowo nadać dowolne funkcje.

Sygnały z kontrolera są przekazywane do programu za pomocą struktury:

```
typedef struct _sixenseControllerData {
  float pos[3];
  float rot_mat[3][3];
  float joystick_x;
  float joystick_y;
  float trigger;
  unsigned int buttons;
  unsigned char sequence_number;
  float rot_quat[4];
  unsigned short firmware_revision;
```

```
  unsigned short hardware_revision;
  unsigned short packet_type;
  unsigned short magnetic_frequency;
  int enabled;
  int controller_index;
  unsigned char is_docked;
  unsigned char which_hand;
  unsigned char hemi_tracking_enabled;
} sixenseControllerData;
```

5.2.3. Obsługa kontrolera

Razer Hydra jest wspierany przez biblioteki programowe działające w systemach MS Windows, MacOS i Linux. Producent biblioteki dla Windows skompilował ją w MS Visual Studio 2010 w wersjach dla procesorów 32 i 64-bitowych. Biblioteki pozwalają obsłużyć jednocześnie kilka kontrolerów w jednym programie. Programowa obsługa kontrolera sprowadza się do wywołania kilku funkcji:

- Podczas włączenia programu, wykonywana jest funkcja `sixenseInit()`. Inicjalizuje ona bibliotekę dostarczoną przez firmę Sixsense.
- Wykonanie funkcji `sixenseSetActiveBase(0);` powoduje wybranie stacji bazowej, z której pomiary chcemy odczytywać.
- Wykonanie `sixenseUtils::getTheControllerManager()->update( &acd );` wypełnia strukturę `acd` danymi odczytanymi z kontrolerów.

Zaimplementowana aplikacja pozwala poruszać się w wirtualnym świecie zgodnie z sygnałami przesyłanymi z kontrolera (sygnały te po przetworzeniu są interpretowane jako odpowiednie działanie). W aplikacji obsługiwane są proste gesty: rozciąganie – powoduje zwiększenie przedmiotu/pudełka (jest uzyskiwane przez oddalanie od siebie kontrolerów), pomniejszanie – powoduje odwrotne skutki do zwiększania (jest uzyskiwane przez zbliżanie do siebie kontrolerów). Przyciski pozwalają na chwytanie przedmiotów oraz zmianę pozycji wybranego przedmiotu poprzez odpowiedni ruch kontrolera (obrót, przesunięcie). Również za pomocą przycisków można wejść do środka danego pudełka i manipulować przedmiotami umieszczonymi w środku.

Literatura

[1] R.A. El-Laithy, J. Huang, M. Yeh. Study on the use of microsoft kinect for robotics applications. *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, strony 1280–1288. IEEE, 2012.

[2] S. Obdrzálek, G. Kurillo, F. Ofli, R. Bajcsy, E. Seto, H. Jimison, M. Pavel. Accuracy and robustness of kinect pose estimation in the context of coaching of elderly population. *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, strony 1188–1193. IEEE, 2012.

ROZDZIAŁ 6

GRY ANALOGOWE Z KOMPUTEREM

M. Niestrój, Ł. Pawlik

Wraz z nastaniem ery komputerów wiele dziedzin codziennego życia zostało poddane cyfryzacji. Dotyczy to między innymi sfery gier, w której tradycyjną planszę, piony, karty i inne rekwizyty zastąpiły wirtualne twory rezydujące w pamięci komputera i wizualizowane na jego ekranie. Choć popularność gier w wersji elektronicznej w ostatnich kilkunastu latach bardzo wzrosła, to jednak wciąż wytrawni gracze chętnie prowadzą rozgrywki w świecie rzeczywistym. Czy w takim razie można powiązać cechy gier analogowych i cyfrowych w jednym rozwiązaniu? Czy można stworzyć pomost pomiędzy światem wirtualnym i rzeczywistym tak, aby można było korzystać z rzeczywistej planszy, a przy tym nie tracić komputerowego wsparcia (przy analizie dobrych i złych ruchów, podczas pilnowania reguł gry itp.)? Odpowiedzi na te pytania można szukać w literaturze ([1, 2, 3]). Można też zagłębić się w treść niniejszego rozdziału. Dokonano w nim przeglądu wybranych narzędzi i metod oraz opisano projekt autorskiego rozwiązania umożliwiającego prowadzenie gry w warcaby z komputerem na analogowej planszy.

6.1. Cyfrowe przetwarzanie obrazu

Cyfrowe przetwarzanie obrazów zajmuje się metodami reprezentacji obrazów w postaci cyfrowej, algorytmami ich przetwarzania oraz technikami ich akwizycji. Jest to popularna i prężnie rozwijająca się dziedzina nauki [4]. Chociaż największy postęp nastąpił w latach 60 i 70 *XX* wieku, to praktyczne zastosowania wyników tych badań nastąpiło dopiero na początku *XXI* wieku, gdy rozwój technologii informatycznych pozwolił na przetwarzanie znacznych ilości danych w czasie rzeczywistym.

Komputerowe przetwarzanie obrazów nie należy do zadań łatwych. Fakt ten często zaskakuje ludzi, którym analiza obserwowanego świata nie nastręcza trudności. Wielu wydaje się, że ta czynność jest prosta i naturalna. Za tą prostotą stoi jednak skomplikowana struktura działania ludzkiego mózgu, organu wzroku oraz posiadana przez ludzi zdolność uczenia się. Nauka nie odkryła jeszcze wszystkich niuansów działania ludzkiego umysłu. Stąd też implementacja algorytmów

odzwierciedlających jego działanie jest wielce złożona. Poniżej opisano algorytmy wykorzystane w implementacji wspomnianego we wprowadzeniu autorskiego rozwiązania.

6.1.1. Wykorzystane algorytmy

Podczas budowy systemów służących przetwarzaniu obrazów krytycznym momentem może okazać się właściwy dobór metod i algorytmów. Do realizacji planowanego rozwiązania wybrano algorytmy opisane poniżej (w większości dostępne w bibliotece OpenCV `http://docs.opencv.org`).

Wykrywanie krawędzi algorytmem Canny'ego – jest to jedna z najczęściej używanych metod detekcji krawędzi w cyfrowym przetwarzaniu obrazu. Zaproponował ją Johna F. Canny w 1986 roku. Cel, jakiemu służy, określają trzy założenia ([5] oraz [6]):

- skuteczne wykrywanie – znalezienie jak najwięcej rzeczywistych krawędzi,
- poprawna lokalizacja – zlokalizowanie krawędzi jak najbliżej rzeczywistego położenia,
- minimalny nadmiar – wytyczenie krawędzi pojedynczą linią, odrzucenie fałszywych krawędzi generowanych przez szumy na obrazie.

Przed przystąpieniem do wykrywania krawędzi dobrą praktyką jest zredukowanie jego rozmiaru poprzez konwersję do odcieni szarości. Taki krok wykonano w implementacji autorskiego rozwiązania. Poniżej przedstawiono przebieg realizacji metody Canny'ego z uwzględnieniem takiej konwersji [7]:

1. **Konwersja do odcieni szarości**
   Obraz przed przetwarzaniem można skonwertować do odcieni szarości (co upraszcza dalsze jego przetwarzanie).
2. **Redukcja szumu**
   Pierwszym, głównym etapem algorytmu jest usunięcie szumu z obrazu. Stosuje się do tego splot obrazu z filtrem Gaussa. Jako parametr należy podać odchylenie standardowe rozkładu Gaussa. Filtr rozmywa obraz, tym samym usuwając niechciane zakłócenia. Standardowo używany jest filtr rozmiaru 5×5, który wygląda następująco:

$$\frac{1}{159}\cdot\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

3. **Wyliczanie natężenia i kąta gradientu obrazu**
   Następnym krokiem jest wyliczenie gradientu w kierunku x oraz y. Do wyliczenia gradientu można wykorzystać operatory detekcji krawędzi (np. Sobela), które zwracają wartość pierwszej pochodnej dla kierunku pionowego

D_y jak i poziomego D_x. Przykładowy operator Sobela:

−1	0	1
−2	0	2
−1	0	1

1	2	1
0	0	0
−1	−2	−1

Na postawie D_x i D_y wyliczane jest natężenia gradientu:

$$D = \sqrt{D_x^2 + D_y^2}$$

oraz kąt gradientu (zaokrąglany do 0°, 45°, 90° lub 135°):

$$\theta = \arctan\left(\frac{D_x^2}{D_y^2}\right)$$

4. **Usuwanie pikseli nie będących maksimum gradientu wzdłuż jego kierunku**
   Kontury uzyskane w poprzednich krokach posiadają pewną grubość, która jest nierównomierna. Zgodnie z założeniami metoda powinna dostarczyć kontury rysowane kreską grubości jednego piksela. Dlatego w kolejnym etapie przetwarzania usuwane są piksele, które nie mają maksymalnej wartości natężenia. W efekcie otrzymywane są linie o pożądanej grubości jednego piksela.
5. **Progowanie z histerezą**
   W ostatnim kroku budowany jest histogram uzyskanych wcześniej obrazów konturów. W kroku tym ustawia się dwa parametry: HT – próg wysoki (ang. *high threshold* oraz LT – próg niski (ang. *low threshold*).
   - $D > HT$ – jeśli wartość gradientu w danym punkcie jest większa od progu wysokiego, to znajduje się tam krawędź.
   - $D \in \langle LT, HT \rangle$ – badane jest otoczenie piksela. Jeżeli wypadkowy gradient jest większy od LT to znajduje się tam krawędź.
   - $D < LT$ - piksel nie jest częścią krawędzi.

Transformacja Hougha – została zaproponowana przez Paula Hougha 1962 roku i jest szczególnym przypadkiem transformaty Radona. Mtoda ta początkowo służyła o wykrywania linii prostych na obrazie, później ją uogólniono do wykrywania figur dających się opisać analitycznie (np. kół).

W standardowym podejści metoda zwraca wektor parametrów opisujących obraz za pomocą linii. W przestrzeni obrazu linia może być opisana równaniem:

$$y = mx + b$$

gdzie *x, y* są punktami w przestrzeni obrazu, a *m i b* parametrami linii (zapis ten nie ogranicza pionowych linii). W transformacie Hougha wykorzystuje się parametry *r* (odległością punktu początkowego układu od linii) oraz θ (kątem pomiędzy wektorem *r* i osią *x*) [8]. Wpisywane są one w równanie:

$$r = x\cos\theta + y\sin\theta$$

lub dla współrzędnej y:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

Znalezienie przecięć linii – metoda ta korzysta z wyników działania transformacji Hougha, tj. układu równań z niewiadomymi x i y.

$$\begin{cases} x\cos\theta_1 + y\sin\theta_1 = r_1 \\ x\cos\theta_2 + y\sin\theta_2 = r_2 \end{cases}$$

Zapisując powyższe w postaci macierzowej otrzymamy

$$AX = b$$

gdzie

$$A = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ \cos\theta_2 & \sin\theta_2 \end{bmatrix} \qquad X = \begin{pmatrix} x \\ y \end{pmatrix} \qquad b = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$$

co daje:

$$X = A^{-1}b$$

Filtracja znalezionych linii – metoda ta pozwala na rozwiązanie problemu znajdowania na obrazie planszy w postaci szachownicy (występującemu podczas implementacji autorskiego rozwiązania). Jej istotę można streścić następująco: informacja dostępna po wykonaniu transformacji Hougha jest nadmiarowa – wykrywane są wszystkie linie na obrazie, także te, które nie opisują krawędzi planszy. Znalezienie planszy należy więc poprzedzić odpowiednią filtracją i weryfikacją.

1. **Filtracja ze względu na orientację linii**.
   Pierwszym etapem filtracji jest selekcja tylko tych linii, które mieszczą się w danym zakresie orientacji. Kąt położenia planszy nie jest znany, więc wymagane jest jego wyznaczenie. W tym celu dokonywana jest segregacja linii na przedziały o wielkości 1°. Znając liczbę linii w każdym przedziale możliwe jest wyznaczenie przedziału orientacji o zadanym zakresie, w którym znajduje się najwięcej prostych. Środek wyznaczonego przedziału jest prawdopodobnym kątem położenia planszy na obrazie. Filtracja linii odbywa się przez odrzucanie tych, których orientacja zbytnio różni się od wyznaczonego kąta położenia planszy.
2. **Grupowanie podobnych linii**
   Wiele krawędzi jest opisywanych przez więcej niż jedną linię. Powodem tego jest słaba jakość obrazu oraz pionki ustawione na planszy, które deformują wykrywane krawędzie. W celu zmniejszenia liczby wykrytych linii zaimplementowano procedurę grupowania linii o podobnym położeniu i orientacji. Na podstawie wyznaczonej grupy tworzony jest opis jednego odcinka reprezentującego całą grupę. W wyniku działania algorytmu znacznie zmniejsza się liczba informacji a tym samym upraszcza się procedura wyznaczania położenia planszy oraz opisujących ją linii.

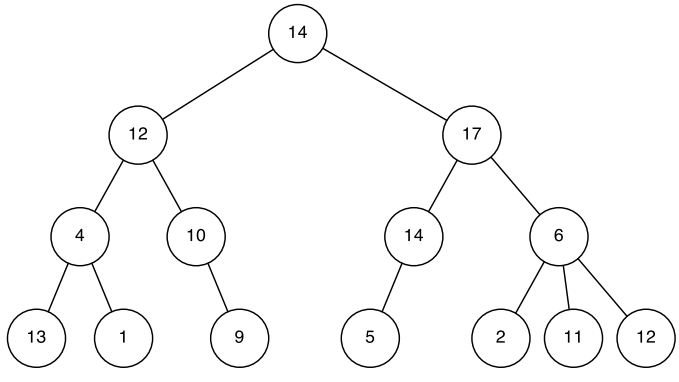
3. **Szukanie planszy na obrazie**
   Znajdywanie planszy na obrazie oraz opisu linii oddzielających poszczególne pola wykonywane jest następującym algorytmem:
   a) **Wyznaczenie dolnej krawędzi** planszy przez znalezienie linii równoległej i najbliżej położonej dolnej krawędzi obrazu.
   b) **Wyznaczenie bocznych krawędzi** planszy poprzez znalezienie odcinków prostopadłych do krawędzi dolnej, z punktem końcowym znajdującym się najbliżej punktu końcowego krawędzi dolnej.
   c) **Wyznaczenie pionowych linii oddzielających pola planszy** przez wybieranie odcinków o podobnej orientacji oraz o odpowiedniej odległości od krawędzi bocznych.
   d) **Wyznaczenie poziomych linii oddzielających pola planszy** przez selekcję odcinków odpowiednio oddalonych od dolnej krawędzi planszy.

6.2. Sztuczna inteligencja

Zadaniem aplikacji komputerowej, po odczytaniu ruchu gracza, jest wybranie ruchu dla komputera. Problemem tym zajmuje się dziedzina informatyki zwana sztuczną inteligencją. Warcaby należą do dwuosobowych gier logicznych. Oznacza to, że określony jest zbiór reguł. Dodatkowo gracze posiadają pełną informację o stanie gry oraz o możliwych posunięciach przeciwnika.

6.2.1. Wykorzystane algorytmy

Algorytm mini-max – algorytm ten pomaga analizować stan gry i wybierać kolejne ruchy gracza. Do tego typu zadań wykorzystuje się tak zwane drzewa przeszukiwań (rys. 6.1), które są specjalnym typem grafów. Każdy element w drzewie nazywany jest węzłem (oznaczony na rysunku jako okrąg), który przyjmuje pewną wartość (zapisaną w środku okręgu). Dodatkowo każdy węzeł posiada ojca (za wyjątkiem korzenia, znajdującego się na górze drzewa) oraz może posiadać kilku potomków. Relacje te reprezentują widoczne połączenia między węzłami.

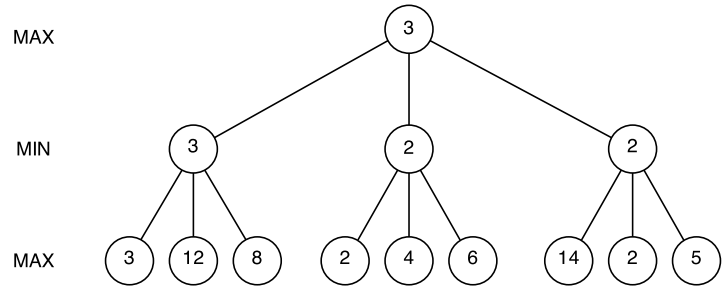


Rys. 6.1: Przykładowe drzewo przeszukiwań.

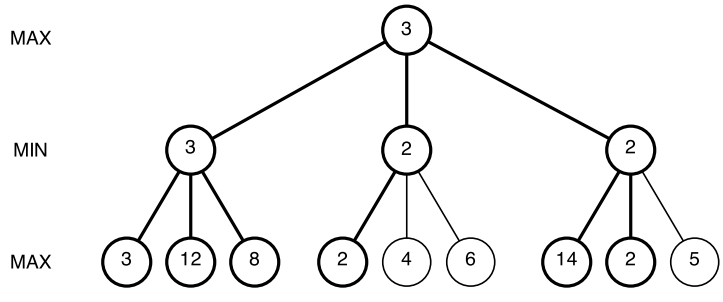
Opisane drzewo przeszukiwań służy do wyznaczania osiągalnych stanów gry, na podstawie wszystkich możliwych ruchów w danym stanie.

Dla gier dwuosobowych, do których należą warcaby, nie można bezpośrednio zastosować metod przeszukiwania wszystkich możliwych ruchów. Algorytm decyzyjny musi uwzględniać także wszystkie możliwe reakcje przeciwnika. Okazuje się, że istnieje optymalna strategia gry stosowana dla gier deterministycznych z pełną informacją, nazywana procedurą mini-max (`http://pl.wikipedia.org/wiki/Algorytm_min-max`).

Zakładamy, że w rozgrywce biorą udział gracze MIN i MAX i że pierwszy ruch należy do gracza MAX. Generowane jest drzewo przeszukiwań zaczynając od aktualnego stanu gry aż do uzyskania zakładanej głębokości drzewa - liczby ruchów graczy. Na rysunku 6.2 przedstawiono przykładowe drzewo przeszukiwań o głębokości równej 2. Wartość liści (węzłów najgłębiej położonych) w drzewie określana jest za pomocą funkcji heurystycznej oceniającej stan gry. W najprostszym przypadku jest to różnica liczby pionków gracza MAX i gracza MIN. Wartości pozostałych węzłów wyznaczane są na podstawie poziomu niższego. Dla elementów warstwy określanej jako MIN przyjmowana jest najmniejsza wartość spośród wszystkich synów, dla węzłów na poziomie MAX wybierana jest największa wartość spośród synów. Optymalny ruch MAX-a wskazywany jest przez gałąź korzenia o największej wartości.



Rys. 6.2: Drzewo przeszukiwań generowane na potrzeby algorytmu mini-max.



Rys. 6.3: Drzewo przeszukiwań generowane na potrzeby algorytmu mini-max z cięciami alfa-beta.

Odcięcia alfa-beta – algorytm ten pozwala zwiększyć efektywność algorytmu mini-max (http://pl.wikipedia.org/wiki/Algorytm_alfa-beta). Poniżej zaprezentowany jest pseudokod algorytmu mini-max z cięciami alfa-beta:

```
function minimax-alfa-beta(n,alfa,beta,glebokosc)
begin
  if glebokosc==maxglebokosc then return(h(n))

  mozliwe_ruchy = liczba_potomkow(n)
  while (not empty(mozliwe_ruchy)) and (alfa < beta) do
  begin
    nowy_n = pierwszy(mozliwe_ruchy)
    mozliwe_ruchy = reszta(mozliwe_ruchy)

    w = minimax-alfa-beta(nowy_n,alfa,beta,glebokosc+1)

    if parzysty(glebokosc) and (w > alfa) then alfa = w     // MAX
    if nieparzysty(glebokosc) and (w < beta) then beta = w  // MIN
  end

  if parzysty(glebokosc) then return(alfa)                  // MAX
  else return(beta)                                         // MIN
end
```

Optymalizacja procedury mini-max polega na sprawdzaniu tylko tych gałęzi drzewa, które mają wpływ na wynik rodziców – czyli mają wpływ na wynik całego algorytmu. Na rysunku 6.3 pogrubionymi liniami zaznaczono obliczane węzły drzewa przeszukiwań. Zgodnie z ideą wprowadzonej optymalizacji nie ma sensu przeszukiwania wszystkich liści drugiej gałęzi wychodzącej z korzenia. Pierwszy liść wspomnianej gałęzi ma wartość równą 2. Wynika z tego, że jego rodzic przyjmie wartość większą lub równą 2, dla jakichkolwiek wartości pozostałych liści. Tymczasem wartość korzenia drzewa zmieni się tylko wtedy, gdy jeden z jego synów przyjmie wartość większą od 3. Można stąd wywnioskować, że przeszukiwanie pozostałych liści drugiej gałęzi drzewa jest zbędne – nie wpłynie one na wynik całej procedury.

6.3. Autorska aplikacja

6.3.1. Schemat działania i przepływ danych w programie

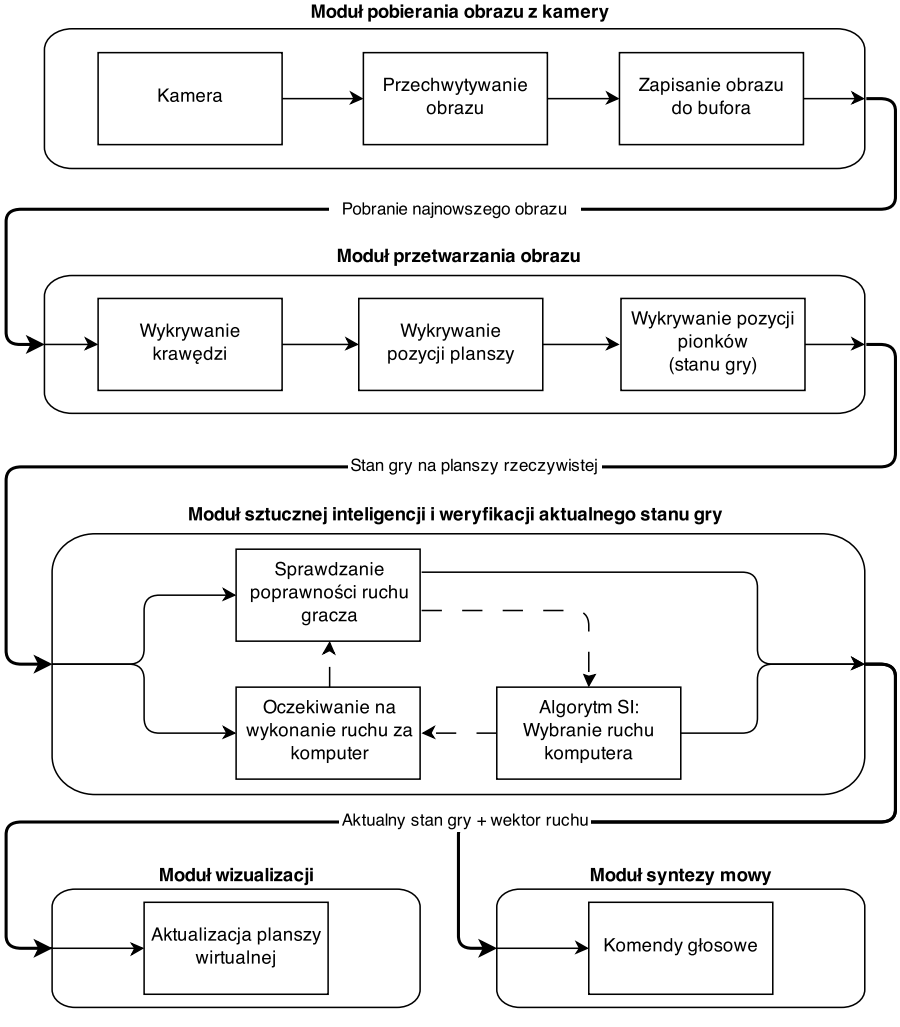
W celu pogrupowania poszczególnych operacji oraz umożliwienia wymiany wybranej części programu na nową, bez zmiany ogólnej struktury kodu, program został podzielony na pięć głównych modułów. Przepływ danych pomiędzy poszczególnymi modułami zaprezentowano na rysunku 6.4.

Moduł pobierania obrazu z kamery zapisuje cyklicznie obrazy do bufora. Celem tej operacji jest szybka dostępność aktualnego obrazu przez algorytmy go przetwarzające.

Moduł przetwarzania obrazu dzieli się na trzy główne operacje: wykrywanie krawędzi, wykrywanie pozycji planszy oraz wykrywanie pozycji pionków na plan-

szy. Każdy z tych algorytmów dostarcza nowych informacji, które w konsekwencji pozwalają wnioskować o aktualnym stanie gry na planszy rzeczywistej.

Moduł sztucznej inteligencji i weryfikacji aktualnego stanu gry jest prostą maszyną stanową. Przejścia pomiędzy stanami zaznaczone są na rysunku linią przerywaną. Sprawdzenie poprawności ruchu gracza jest algorytmem, który sprawdza czy stan gry na planszy rzeczywistej jest możliwy do osiągnięcia po wykonaniu przez gracza swojego ruchu. Gdy wynik jest pozytywny, wywoływany jest algorytm sztucznej inteligencji. Wybrany wektor ruchów oraz nowy stan gry przekazywane są do modułów wizualizacji oraz syntezy mowy. W następnym kroku program oczekuje na interakcję gracza i wykonanie wybranego przez komputer ruchu. Jest to jednocześnie ostatni stan w opisanej maszynie stanowej. Po zwery-



Rys. 6.4: Przepływ danych w programie.

fikowaniu prawidłowego stanu gry na planszy rzeczywistej następuje powrót do pierwszego stanu, czyli oczekiwania na ruch gracza.

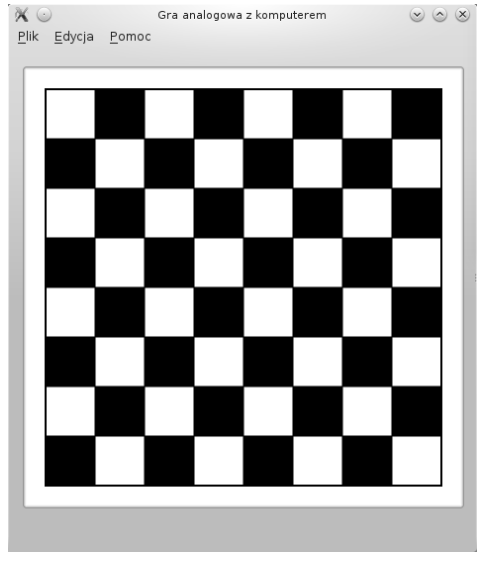
Ostatnim etapem w przepływie danych w programie jest zakomunikowanie użytkownikowi o aktualnej konfiguracji planszy oraz wybranym ruchu komputera. Wykonywane jest to przez tak zwany interfejs maszyna-człowiek, na który składa się moduł wizualizacji oraz moduł syntezy mowy.

6.3.2. Interfejs użytkownika

Na potrzeby aplikacji stworzono interfejs w bibliotece QT (`http://qt-project.org`), pozwalający obsługiwać aplikację w jak najprostszy sposób. Na rysunku 6.5 przedstawiono widok główny aplikacji. Okno składa się z:

- Paska menu – jest to pasek umiejscowiony się na górze okna aplikacji, znajduje się na nim menu programu:
  - ***Plik*** – w rozwijanej liście istnieje możliwość wyboru: *Nowa gra*, *Podgląd*, *Exit*
  - ***Edycja*** – istnieje możliwość zmienienia standardowych ustawień aplikacji w zakładkach: *Kamera*, *Poziom*
  - ***Pomoc*** – informacja o autorach programu w *O programie*.
- Pola szachownicy – główne pole aplikacji, służy do wyświetlania stanu gry.

Rozpoczęcie gry – następuje poprzez wybór przycisku *Nowa gra* w menu *Plik*. Do rozpoczęcia gry potrzebne jest także odpowiednie ustawienie kamery. Podgląd z podłączonej kamery możliwy jest przez wybranie przycisku *Podgląd kamery* w menu *Plik*.



Rys. 6.5: Interfejs główny programu.

Zmiana ustawień aplikacji – w aplikacji można zmienić poziom trudności algorytmu sztucznej inteligencji. Możliwe jest ustawienie wartości z przedział od 1 (łatwa gra) do 15 (trudna gra). Zmiana poziomu dostępna jest po wybraniu *Poziom trudności* w menu *Edycja*. Pojawi się wtedy okno z domyślnie ustawionym poziomem 5 (rys. 6.6).

Rys. 6.6: Ustawienia poziomu trudności sztucznej inteligencji.

Rys. 6.7: Ustawienia kamery.

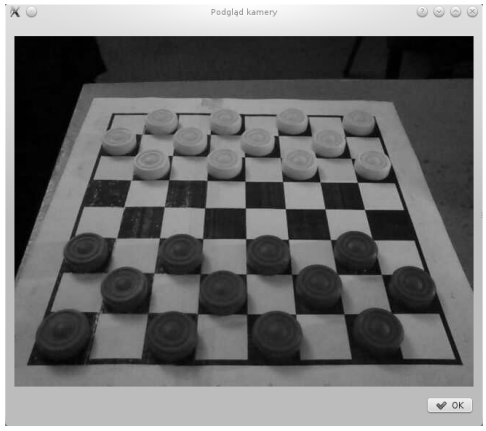
Po wybraniu *Kamera* z menu *Edycja* pojawi się okno do wyboru ustawień kamery (rys. 6.7). Istnieje możliwość podłączenia kamery lokalnej (np. USB, kamera laptopa) lub zdalnej (kamera połączona przez interfejs sieciowy, należy podać adres kamery). Domyślnie ustawiona jest kamera zdalna, udostępniana przez aplikacje *Ip Webcam* na systemie *Android*.

Zmiany wprowadzone do aplikacji zostaną uwzględnione dopiero podczas uruchomienia nowej gry.

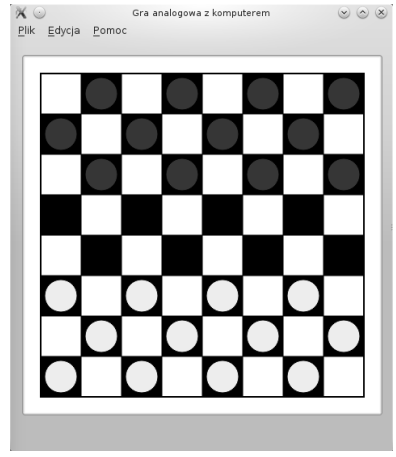
Wyjście z aplikacji – następuje poprzez naciśnięcie *Exit* w menu *Plik* lub wciśnięcie przycisku „*x*”. Następnie należy potwierdzić wyjście z aplikacji.

6.3.3. Przykładowe działanie programu

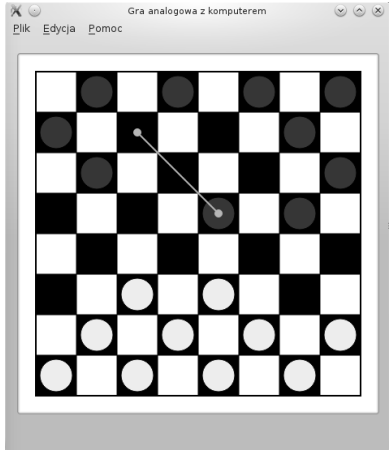
Po uruchomieniu aplikacji pojawi się okno główne (rys. 6.5). Można wtedy zmienić ustawienia programu. Rozpoczynając grę należy wyświetlić okno podglądu kamery, w którym powinien być wyświetlony obraz planszy. Planszę należy ustawić tak, aby była widoczna cała (rys. 6.8). Kiedy plansza zostanie odczytana przez program, w oknie głównym wyświetli się układ pionków na planszy. Należy zauważyć, że wyświetlana szachownica jest odwrócona do widoku z kamery (rys. 6.9). Komputer powiadamia komunikatem głosowym o ruchu gracza. Gracz rozpoczyna grę, robiąc ruch pionkiem. Jeśli jest on dozwolony, komputer wykonuje swój ruch, informując o tym gracza głosem oraz zaznaczeniem przesunięcia narysowanym na szachownicy (rys. 6.10). Gracz powinien fizycznie przesunąć pion przeciwnika zgodnie z tymi wskazówkami (rys. 6.11). Po tym ruchu wygląd szachownicy zaktualizuje się i gracz będzie mógł wykonać swój ruch. Następne ruchy wyglądają podobnie.



Rys. 6.8: Podgląd z kamery.



Rys. 6.9: Początkowy stan gry, ruch gracza.



Rys. 6.10: Ruch komputera.

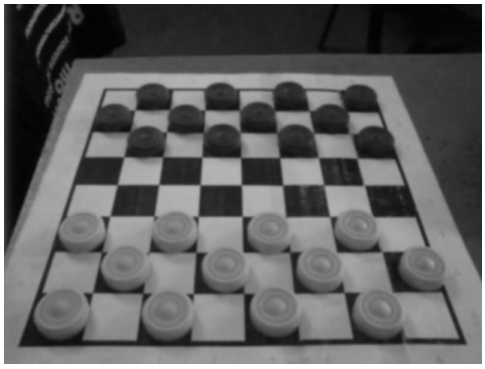
Rys. 6.11: Ustawienie pionów po korekcie.

6.3.4. Interfejs maszyna-człowiek

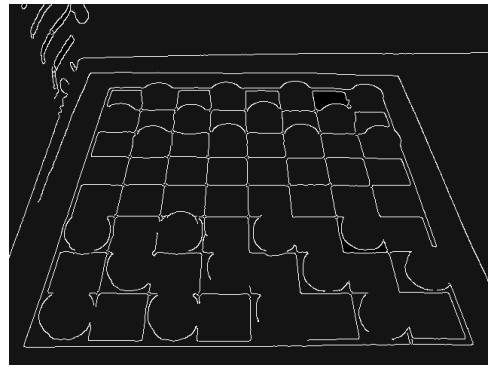
Interfejs maszyna-człowiek jest częścią aplikacji odpowiedzialną za przechwycenie informacji od człowieka i przekazanie jej komputerowi, a także za realizację komunikacji w drugą stronę. Informacje przekazywane komputerowi są zbierane zwykle za pomocą podstawowych urządzeń wejścia takich, jak: mysz komputerowa, klawiatura, gamepad, joystick, kamera, mikrofon. Komunikaty do użytkownika mogą być przekazywane przez: wyświetlacz, głośnik, wibracje. Każde z wymienionych urządzeń wymaga odpowiedniej obsługi programowej.

W stworzonej aplikacji dane pochodzące od człowieka dostarczane są przez: mysz komputerową, klawiaturę oraz kamerę. Mysz i klawiatura pozwalają na zmianę ustawień programu w interfejsie graficznym. Obraz przechwycony przez

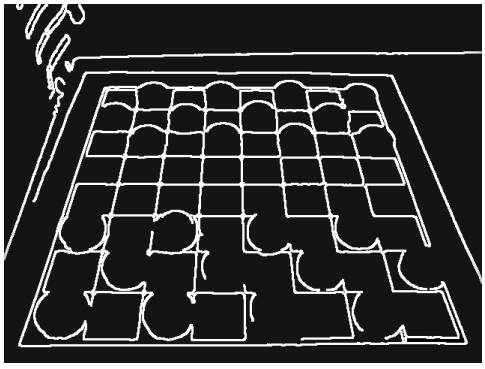
kamerę (rys. 6.8) wymaga przetworzenia aby był zrozumiany przez logikę aplikacji. Dlatego najpierw go rozmywano (usuwanie szumu), następnie konwertowano do odcieni szarości (rys. 6.12). Po tym na obrazie wykrywano kontury z zastosowanie filtru Canny (rys. 6.13).



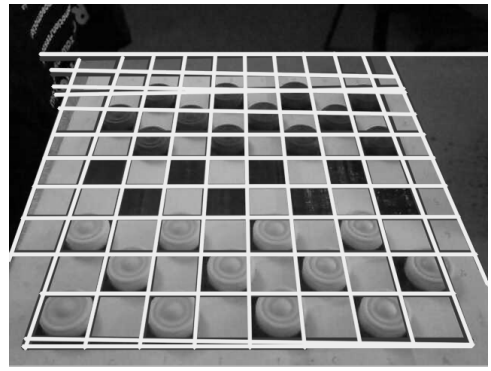
Rys. 6.12: Konwersja do odcieni szarości.



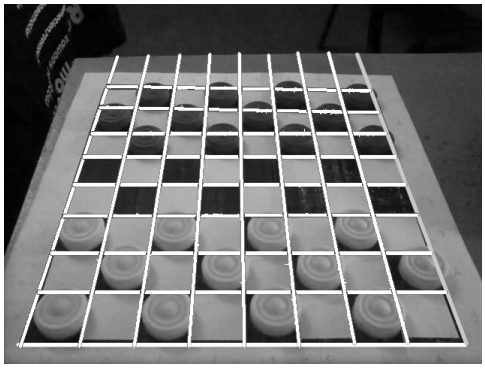
Rys. 6.13: Detekcja konturów (filtr Canny).



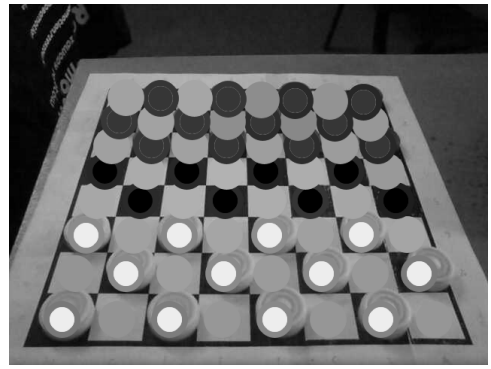
Rys. 6.14: Pogrubienie konturów.



Rys. 6.15: Wykrycie linii (transf. Hougha).



Rys. 6.16: Wykrycie planszy.



Rys. 6.17: Wykrycie pionków na planszy.

W kolejnym kroku użyto operacji morfologicznych, aby pogrubić kontury. Detekcja linii odbywa się za pomocą probabilistycznej transformacji Hougha, która zwraca równania wykrytych prostych (rys. 6.15). Następnie wykrywane jest położenie planszy na obrazie. Rozpoczyna się to przez odnalezienie początku planszy (rys. 6.16). W ostatnim kroku odbywa się detekcja pionków pomiędzy przecięciami pozostałych linii (rys. 6.17).

6.4. Podsumowanie

Sporym wyzwaniem w podejmowanym projekcie było zapewnienie niezawodności działania programu ze względu na panujące oświetlenie, pojawiające się cienie, chwilowe przesłanianie planszy przez ręce oraz klasyfikowanie położenia pionków do danych pól na planszy. Stworzona aplikacja pozwala na grę z komputerem za pośrednictwem naturalnych dla człowieka interfejsów, którymi są wzrok i słuch.

Literatura

[1] E. Molla, V. Lepetit. Augmented reality for board games. *ISMAR*, strony 253–254. IEEE, 2010.

[2] S. Scher, R. Crabb, J. Davis. Making real games virtual: Tracking board game pieces. *ICPR*, strony 1–4. IEEE, 2008.

[3] P. Song, S. Winkler, J. Tedjokusumo. A tangible game interface using projector-camera systems. J.A. Jacko, redaktor, *HCI (2)*, wolumen 4551 serii *Lecture Notes in Computer Science*, strony 956–965. Springer, 2007.

[4] R. Tadeusiewicz, P. Korohoda. *Komputerowa analiza i przetwarzanie obrazów*. Wydawnictwo Fundacji Postępu Telekomunikacji, 1997.

[5] M. Wnuk. Systemy wizyjne. Materiały do kursu, `http://rab.ict.pwr.wroc.pl/dydaktyka/sywiz/index.html`. Dostęp: Czerwiec 2014.

[6] M. Wnuk. Cyfrowe przetwarzanie obrazów i sygnałów. Materiały do kursu, `http://rab.ict.pwr.wroc.pl/~mw/Docs/arek00009.html`. Dostęp: Czerwiec 2014.

[7] R. Ludwiczuk. Algorytm Canny'ego detekcji krawędzi w procesie segmentacji obrazów medycznych. *II Konferencja Entuzjastów Informatyki*, Maj 2004.

[8] E. Rafajłowicz, W. Rafajłowicz. *Wstęp do przetwarzania obrazów przemysłowych*. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010.

ROZDZIAŁ 7

ŚRODOWISKO DO TESTOWANIA INTERFEJSU HAPTYCZNEGO

M. Szymański, V. Tsiselskyi

Podczas operowania przedmiotami człowiek wykorzystuje dotyk. Dzięki temu zmysłowi jest w stanie identyfikować przedmioty oraz określać ich właściwości. Dotyk dostarcza też informacji o wzajemnych reakcjach pomiędzy manipulującym a przedmiotem, na który oddziałuje. Bez rozwinięcia tego zmysłu prosta czynność podniesienia szklanki może okazać się bardzo trudnym zadaniem. Dlatego wprowadzenie interfejsu haptycznego w rozwiązaniach technicznych powinno usprawnić manipulowanie obiektami i sterowanie urządzeniami.

W niniejszym rozdziale przedstawiono propozycję środowiska do testowania działania interfejsu haptycznego zastosowanego przy sterowaniu dźwigiem. Skupiono się na badaniu wpływu przeniesienia sił oddziałujących na ładunek na drążek sterujący operatora. Wprowadzenie interfejsu haptyczny w tym konkretnym przypadku powinno pozwolić na sprawniejsze przenoszenie ładunków.

7.1. Wstęp

Technologie haptyczne są stosowane przy rozwiązywaniu problemów związanych z oddziaływaniem człowieka z maszyną poprzez zmysł dotyku. Wykorzystuje się w nich różne techniki pozwalające na rejestrację i emulację: siły, wibracji, ruchu, chropowatość powierzchni itp. Prostym przykładem zastosowania technologii haptycznej jest umieszczenie wzdłuż osi jezdni specjalnie spreparowanej chropowatości. Najechania na nią kołami generuje wibracje informujące kierowcę, że wyjeżdża poza swój pas, a więc wykonuje operację niebezpieczną. Nieco inne zadanie spełniają chropowatości instalowane na przejściach dla pieszych. Ich zadaniem jest ułatwienie orientacji niewidomym, aby łatwiej lokalizowali miejsca, w których należy się zatrzymać przed przekroczeniem jezdni.

Jednym z zastosowań interfejsów haptycznych są wyświetlacze taktylne. Spotykane są na przykład: wyświetlacze brajlowskie – wspomagające czytanie tekstów (alfabet Braile'a jest przeznaczony do czytania za pomocą dotyku) oraz

Optacon'y (ang. *OPtical to TActile CONverter*) – wspomagające przeglądanie obrazów poprzez konwersję kolorów na wypukłości.

Dotykiem można również sterować tablety i telefony komórkowe. Ponieważ ekrany dotykowe w tych urządzeniach nie oferują zmieniającej się elastyczności czy też chropowatości (używając jakiegoś przycisku czy nieaktywnego elementu graficznego interfejsu użytkownik normalnie nie czuje żadnej różnicy), dlatego wyposaża się je w silniczki generujące wibracje odpowiednio do podjętej akcji.

Dość wcześnie interfejsy haptyczne zastosowano w grach komputerowych, a dokładniej mówiąc, w konstrukcjach joystików i kierownic ze sprzężeniem siłowym (ang. *force-feedback*). Technologia ta pozwalała zasymulować opór stawiany przez prawdziwą kierownicę samochodu lub wolant samolotu oraz tworzyć kontrolery innych urządzeń [1, 2].

Do poważniejszych zastosowań interfejsów haptycznych zalicza się wsparcie wykonywania teleoperacji. Przykładem jest tu technologia sterowania statkiem powietrznym *Fly-by-wire*. Umożliwia ona przekazywanie stanu poprzez połączenia niemechaniczne, a w efekcie umożliwia łatwe sterowanie zdalne oraz częściowe sterowanie poprzez komputer z zachowaniem optymalnych cech sterowności i stateczności. W ostatnim czasie można też zaobserwować szybki rozwój robotów sterowanych haptycznie stosowanych w medycynie. Przykładem może być tu projekt ReMeDi, w ramach którego ma powstać manipulator umożliwiający zdalne wykonywanie badań palpacyjnych. Sterowanie tym robotem odbywać się ma za pośrednictwem konsoli, która przekazywać ma operującemu lekarzowi reakcję na wywieraną zdalnie przez niego siłę.

> **Interfejs haptyczny** – jest to interfejs komunikacji zbudowany dla człowieka, w którym wykorzystuje się technologie haptyczne. Interfejsy haptyczne nabierają coraz większej popularności, ponieważ są dla ludzi intuicyjne i pozwalają na przekazywanie informacji w sytuacjach, gdy skorzystanie z pozostałych zmysłów jest z jakichś powodów utrudnione.

W praktycznych zastosowaniach wykorzystuje się czujniki sił na efektorze jak te, pokazane na rysunku 7.1. Dają one co najmniej przyłożenie sił zewnętrznych w trzech kierunkach przestrzeni kartezjańskiej.

7.2. Propozycja wykorzystania sprzężenia haptycznego przy przenoszeniu ładunku dźwigiem

Obecnie technologia haptyczna nie jest wykorzystywana w budowie przemysłowych kontrolerów takich urządzeń, jak np. dźwigi lub koparki. Ich operatorzy podczas pracy muszą polegać jedynie na zmysłach wzroku i słuchu. Wydaje się, że dodanie sprzężenia haptycznego przyniosłoby wiele korzyści. Przykładowo:

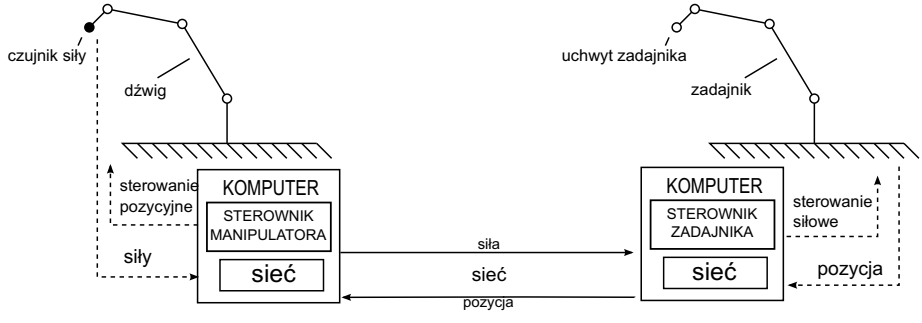
- Dzięki sprzężeniu haptycznemu łatwo byłoby wykryć potencjalną awarię dźwigu. Podobnie jak kierowca może z łatwością stwierdzić problem z układem kierowniczym lub zawieszeniem samochodu, tak operator dźwigu mógłby poczuć niepoprawne działanie mechanizmu.

Rys. 7.1: Wygląd trzyosiowego czujnika siły.

- Przekazywanie sił na drążki sterujące pozwoliłoby na wyczuwanie bezwładności i stanu podwieszonych ładunków, np. zahaczenie się podczas podnoszenia, przechył na skutek silniejszych podmuchów wiatru. Dzięki odebranym sygnałom możliwa byłaby korekta sterowania lub przerwanie ryzykownych operacji.
- Sprzężenie czuciowe wbudowane w kontroler mogłoby reagować na zbliżanie się do granic możliwości danego urządzenia, np. stawiać zwiększający się opór przy próbie podnoszenia coraz to cięższych przedmiotów (w szczególności opór nie do pokonania).

7.2.1. Podstawy teoretyczne

Na rysunku 7.2 przedstawiono ogólny schemat budowy układu telemanipulacji ze sprzężeniem haptycznym. Zadajnik otrzymuje od układu wykonawczego (w rozważanym przypadku – od dźwigu) informację z czujnika siły. Przesyłane siły są przeskalowywane i odzwierciedlane na zadajniku. Dzięki temu użytkownik czuje te siły tak, jakby sam trzymał za linę dźwigu. Poruszając zadajnikiem użytkownik generuje informacje o konfiguracji (położeniu i orientacji), która jest przenoszona na urządzenie wykonawcze. W przypadkach granicznych (np. kolizji) końcówka zadajnika nie podda się manipulacjom użytkownika odzwierciedlając w ten sposób opór kolidującego obiektu.



Rys. 7.2: Układ telemanipulacji ze sprzężeniem haptycznym.

Od strony układu wykonawczego zadanie sterowanie sprowadza się do sterowania konfiguracji w przestrzeni zadaniowej. Od strony zadajnika problemem jest sterowanie siłowe, zapewniające równoważność przeskalowanej wypadkowej sił działających na efektor zadajnika i wypadkowej sił otrzymywanych z urządzenia wykonawczego. Uzyskanie takiego efektu wymaga najpierw skompensowania sił działających na efektor zadajnika, wynikających z dynamiki tegoż zadajnika.

Dla przykładu: równanie dynamiki manipulatora sztywnego wyraża się wzorem:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D(q) = u$$

Kompensacja odbywa się poprzez podanie na wejście u momentu wyliczonego poprzez rozwiązanie zadania dynamiki odwrotnej:

$$u = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D(q)$$

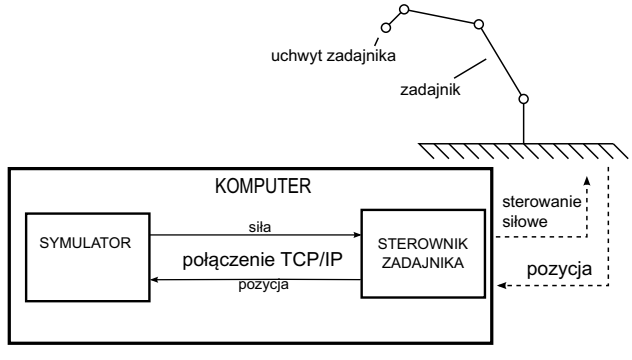
Przy takim sterowaniu manipulator utrzymuje swoją pozycję, ale nie stawia oporu próbom zmiany swojej konfiguracji. W celu odtworzenia sił z urządzenia wykonawczego na zadajniku do powyższego równania dodaje się człon $J^T F$. Prawo sterowania wygląda wtedy następująco:

$$u = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D(q) + J^T F$$

gdzie J oznacza jakobian zadajnika, a F jest wektorem sił działających na efektor urządzenia wykonawczego. Więcej szczegółów o modelowaniu i sterowaniu z siłowym sprzężeniem zwrotnym można znaleźć w pracy [3].

7.2.2. Założenia symulacji

Do sprawdzenia założeń teoretycznych dotyczących implementacji interfejsów haptycznych potrzebne jest środowiska testowego składające się zasymulowanego układu wykonawczego oraz interfejsu umożliwiającego podłączanie zadajnika lub jego symulatora. Na rysunku 7.3 przedstawiono propozycję schematu takiego środowiska.



Rys. 7.3: Środowisko do testowania interfejsów haptycznych.

W symulatorze będącym egzemplifikacją tej propozycji końcówka dźwiga jest reprezentowana przez punkt. Symulacja zaś umożliwia podjechanie do obiektu, podpięcie go do końcówki dźwiga za pomocą wirtualnej linki oraz wykonanie podstawowych manipulacji.

Położenia przegubów manipulatora są uzyskiwane z enkoderów lub innych urządzeń pomiarowych. Następnie są one przesyłane do komputera, gdzie trafiają natychmiastowo do symulatora. Powoduje to widoczne zmiany w położeniu efektora w środowisku wirtualnym. Komunikacja realizowana z wykorzystaniem protokołu TCP/IP.

Środowisko wirtualne modeluje zjawiska fizyczne występujące w prawdziwym świecie. W szczególności są to zderzenia obiektów oraz siła grawitacji. Użytkownik porusza się w tym środowisku jedną bryłą (kulą), dostając informację zwrotną o siłach działających na tą bryłę. Siły te są traktowane tak, jakby były przyłożone w jednym punkcie w środku kuli i sumowane wektorowo (co daje siłę wypadkową działającą na efektor – czujnik sił). Wypadkowa siła jest następnie przesyłana przez połączenie TCP-IP do sterownika zadajnika. Sterownik przelicza tę siłę na momenty sterujące silnikami robiąc to w taki sposób, aby na uchwycie odtworzona została zadana siła.

7.3. Realizacja praktyczna

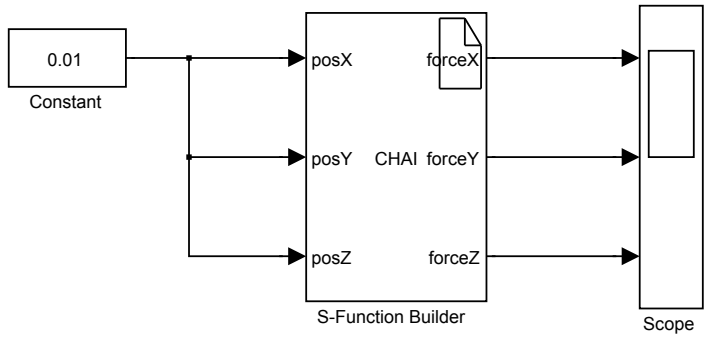
Do realizacji zadania przebadano dwa środowiska symulacyjne obsługujące sprzężenie siłowe: Open Haptic oraz Chai3D. Ostatecznie do dalszych prac wybrano środowisko Chai3D. Umożliwia ono symulację świata wirtualnego z wystarczającą symulacją fizyki i pozwala obsługiwać kilku urządzeń haptycznych. Chcąc skorzystać z jakiegoś urządzenia należy uzupełnić brakujące fragmenty klasy `CMyCustomDevice`. W plikach źródłowych klasy umieszczone są wskazówki, jak to zrobić.

Podczas pracy systemu następuje przesyłanie wektora położenia efektora z zadajnika do środowiska symulacyjnego Chai3D oraz przesyłanie wektora sił z środowiska Chai3D do zadajnika. Algorytmy sterujące zadajnikiem mają współdziałać ze środowiskiem MATLAB/Simulink. Docelowo dane o położeniu będą odczytywane z enkoderów manipulatora Delta [4] i po przeliczeniu na współrzędne w przestrzeni zadaniowej będą przesyłane do bloku typu `S-function` w Simulinku (rys. 7.4).

`S-function` jest to bloczek pozwalający na wykonywanie procedur napisanych w języku C podczas symulacji (kod danej procedury jest zamieszczany w oknie parametrów danego bloczka). Stosowany jest w przypadku projektowania funkcji o niewielkim rozmiarze kodu.

Na potrzeby omawianego rozwiązania stworzono bibliotekę `Chai3d_connect`. Kompletny interfejs biblioteki przedstawiono na poniższym listingu.

```
#ifdef __cplusplus
        extern "C" {
#endif
```



Rys. 7.4: Model w Simulinku.

```
        void setPosX(double setVal);
        void setPosY(double setVal);
        void setPosZ(double setVal);
        double getForceX();
        double getForceY();
        double getForceZ();

#ifdef __cplusplus
        }
#endif
```

Interfejs biblioteki opakowuje klasę `HapticDeviceConnect` (C++) w funkcje języka C, co spowodowane jest wymaganiami bloczka `S-function`. Klasa `HapticDeviceConnect` przechowuje odbierany wektor siły i wysyłany wektor położenia (z perspektywy zadajnika).

Funkcje należące do interfejsu biblioteki wywołują metody klasy `HapticDeviceConnect` służące do odczytu poszczególnych współrzędnych z przechowywanego przez klasę wektora siły i zapamiętania wektora położenia w odpowiednim obiekcie. W konstruktorze klasy uruchamiany jest wątek, który odbiera dane przesyłane po TCP/IP umieszczając je w wektorze siły oraz wysyła dane umieszczone w wektorze położenia po TCP/IP.

Obsługę wątków i sieci TCP/IP zrealizowano za pomocą biblioteki `boost` (moduły `asio` i `thread`). Z powodu wykorzystania osobnego wątku wysyłającego i odbierającego dane, dostęp do każdego z wektorów zabezpieczono mutexem.

Interfejs TCP/IP po stronie symulatora zrealizowano podobnie. Jak wspomniano wcześniej, aby dodać własne urządzenie do Chai3D, należy zaimplementować klasę `CMyCustomDevice` uzupełniając jej brakujące fragmenty.

Wykonane przez autorów modyfikacje obejmują: dodanie obiektu klasy `tcp_server` jako pola, uruchomienie wątku obsługującego przesyłanmie danych w konstruktorze klasy `CMyCustomDevice` (wątek jest metodą klasy `tcp_server`), oraz napisanie kodu odczytującego wektor położenia z klasy `tcp_server` i zapisującego wektor sił do klasy `tpc_server` (w celu wysłania).

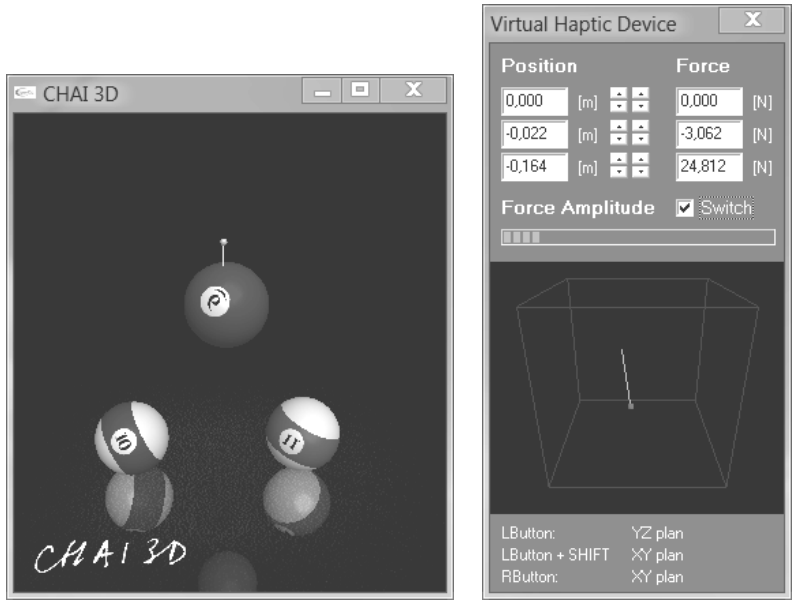
Metoda `startService()` klasy `tcp_server` uruchamia serwer TCP/IP. Po odebraniu połączenia przechodzi w tryb wymiany danych. Zapisany w klasie wektor sił jest przesyłany przez TCP/IP, a odebrane dane są zapisywane w klasie w wektorze położenia. Tak samo jak w klasie `HapticDeviceConnect` opisanej wcześniej, dostępu do obu wektorów bronią mutexy.

7.4. Wyniki

Założenia projektu obejmowały symulację sterowania dźwigiem. Ciągła interakcja zadajnika (robota typu Delta) ze środowiskiem symulacyjnym nie była jeszcze możliwa. Z tego powodu symulację sterowania dźwigiem wykonywano osobno, z użyciem wirtualnego urządzenia haptycznego. Przedstawiono to na zrzutach ekranu (rys. 7.5).

Wynikiem pracy są biblioteki umożliwiające połączenie środowiska symulacyjnego Chai3D z robotem typu Delta rozwijanym w ramach projektu ReMeDi. Opracowane rozwiązanie umożliwiaj łatwe i szybkie prototypowanie interfejsów haptycznych oraz ich testowanie. Do opracowania sterownika urządzenia można bowiem wykorzystać MATLAB/Simulink (nadającym się do łatwego i szybkiego implementowanie algorytmów sterowania) a przy testowaniu można wykorzystać wsparcie wizualizacji oraz obliczeń fizycznych oferowane przez Chai3D.

Choć modelowanie konkretnej sytuacji jest stosunkowo wygodne (polega na napisaniu odpowiedniego kodu w C++ korzystającego z biblioteki Chai3D), to problemem może okazać się niewielka baza trójwymiarowych modeli oraz ko-



Rys. 7.5: Okno symulatora przedstawiające wizualizację sterowania dźwigiem (z lewej) oraz parametry wirtualnego zadajnika (z prawej).

nieczność ręcznego podawania parametrów materiału, z którego zrobiono dane obiekty. Jednak ta ostatnia własność niekoniecznie musi być wadą, ponieważ pozwala na łatwe eksperymentowanie.

Warto nadmienić, że proponowane rozwiązanie można z łatwością wdrożyć na różnych platformach. Ponadto z uwagi na użycie protokołu TCP/IP, środowisko MATLAB można zastąpić zupełnie innym oprogramowaniem i używać tylko części bibliotek od strony Chai3D.

Literatura

[1] P. Bachman, M. Chciuk, A. Milecki. Sterowanie aktywnym dżojstikiem dotykowym hapticuz 1-dof/dc. *Pomiary, Automatyka, Robotyka*, 2:519–522, 2012.

[2] P. Gawłowicz, M. Chciuk, P. Bachman. Robot sterowany trzyosiowym dżojstikiem dotykowym z cieczą magnetoreologiczną. *Pomiary, Automatyka, Robotyka*, 2:703–709, 2009.

[3] P. Owczarek, D. Rybarczyk, F. Stefański. Modelowanie i sterowanie z siłowym sprzężeniem zwrotnym elektrohydraulicznego manipulatora w środowisku wirtualnym. *Modelowanie inżynierskie*, 15(46), 2013.

[4] A. Olsson. Modeling and control of a Delta-3 robot, 2009. Department of Automatic Control, Lund University, ISRN LUTFD2/TFRT–5834–SE.

ROZDZIAŁ 8

WYKORZYSTANIE ALGORYTMU GENETYCZNEGO W ROZWIĄZYWANIU PROBLEMU SPEŁNIANIA OGRANICZEŃ

W. Domski, W. Górniak, D. Kwaśnik

8.1. Wstęp

Współcześnie w wielu dziedzinach nauki napotyka się problemy związane z poszukiwaniem najlepszych lub nierzadko najgorszych rozwiązań. Wynika to z tendencji do maksymalizowania zysków lub minimalizacji strat przy jednoczesnym spełnieniu pewnych ograniczeń.

Problemy te można zilustrować przykładem robota mającego za zadanie dojechać do celu startując z aktualnego położenia. W najprostszym przypadku rozwiązaniem jest odcinek łączący punkt początkowy z końcowym. Jednak w szczególnej sytuacji, gdy w środowisku robota pojawią się przeszkody (ograniczenia), wymiar problemu gwałtownie wzrośnie. Innym przykładem zagadnienia z ograniczeniami jest problem hetmanów. Polega on na rozstawieniu figur na szachownicy w taki sposób, aby nie atakowały się wzajemnie. Hetman atakuje wzdłuż kolumn, wierszy oraz przekątnych. W zależności od rozmiaru planszy oraz ilości hetmanów należy odpowiedzieć na pytanie czy istnieje wspomniane rozstawienie figur oraz ile jest możliwych rozwiązań problemu.

Zagadnienie ułożenia hetmanów czy też wyznaczenia trajektorii platformy mobilnej są przykładami problemów z ograniczeniami, tzw. CSPs (ang. *Constraint satisfaction problems*). Algorytmy rozwiązujące CSPs mają wiele praktycznych zastosowań. Współcześnie ciężko wskazać dziedziny wiedzy, w których tego typu algorytmy nie byłyby wykorzystywane.

W niniejszym rozdziale zaprezentowano praktyczny problem planowania zagospodarowania przestrzeni wystawowej. Polega on na rozmieszczeniu płaskich elementów (jakimi są pawilony handlowe) opisanych przez swoje parametry charakterystyczne (szerokości i wysokości) w dwuwymiarowej przestrzeni (jaką jest hala wystawowa) przy uwzględnieniu obszarów wyłączonych z użytku (obszarów zabronionych).

8.2. Problem CSP

8.2.1. Przykłady

Tendencja coraz częstszego sprowadzania problemów z wielu dziedzin życia codziennego do CSP skutkuje rozwojem narzędzi do ich rozwiązywania, ze szczególnym naciskiem na języki programowania. W artykule [1] opisano techniki programistyczne służące do modelowania i rozwiązywania problemów planowania, harmonogramowania, przeszukiwania i optymalizacji z ograniczeniami (ang. *Constraint programming*, CP). W pracy [2] przedstawiono problem harmonogramowania satelitów obserwacyjnych Ziemi, w którym każdy satelita może dostać nowe zadanie do wykonania w niedeterministycznym czasie. Oczekiwane jest, że aktualny harmonogram działań zmieniał się w jak najmniejszym stopniu. W artykule [3] opisano metody dynamicznego sterowania ruchem ulicznym wykorzystujące reprezentację DCSP (ang. *distributed* CSP).

Formalizm CSP dobrze odzwierciedla charakter i jest przydatnym narzędziem do rozwiązania następujących problemów:

- kolorowanie mapy: przy znanych nazwach i wyznaczonych na mapie granicach krain geograficznych oraz przy danym zbiorze dostępnych kolorów należy tak pokolorować obszary krain, aby sąsiadujące ze sobą krainy miały różne kolory,
- planowanie produkcji przemysłowej: przy dostępnych materiałach oraz zasobach ludzkich i maszynowych należy ułożyć harmonogram produkcji zaspokajający planowaną liczbę zamówień,
- zadanie optymalnego rozmieszczania elementów na płaszczyźnie występujące np. podczas cięcia laserowego [4].

8.2.2. Formalna definicja

Formalnie CSP definiuje się jako trójkę [5]: $\langle X, D, C\rangle$, gdzie: $X = \{X_1, \ldots, X_n\}$ jest zbiorem zmiennych, $D = \{D_1, \ldots, D_n\}$ jest dopuszczalną dziedziną rozwiązania, zaś $C = \{C_1, \ldots, C_m\}$ jest zbiorem ograniczeń. Poszukiwany jest taki zbiór zmiennych, który należy do dziedziny rozwiązania i spełnia wszystkie narzucone ograniczenia. Otrzymany wynik nie musi być jedynym poprawnym, nie jest też zapewniona jego optymalność.

Powyższa definicja umożliwia rozwiązanie dowolnego problemu przedstawionego w postaci wyżej opisanego formalizmu za pomocą grupy algorytmów opracowanych dla typowego CSP. Problemy z ograniczeniami można klasyfikować i dzielić ze względu na:

- rodzaj zmiennych:
  - dyskretne o skończonej dziedzinie, np. problem spełnialności, w którym należy ustalić czy dla danej formuły logicznej istnieje takie podstawienie zmiennych zdaniowych, żeby formuła była prawdziwa,
  - dyskretne o nieskończonej dziedzinie, np. układanie harmonogramu prac, gdzie poszukiwane są ramy czasowe ich wykonania,
  - ciągłe,

- rodzaj ograniczeń:
  - pojedyncze, czyli zawierające tylko jedną zmienną, np. kolor,
  - podwójne, czyli zawierające parę zmiennych,
  - wyższego rzędu, w których występuje 3 i więcej zmiennych,
  - „z miękkimi ograniczeniami" (preferencjami), prowadzącymi do problemów optymalizacji z ograniczeniami.

8.3. Algorytm genetyczny

Algorytmy genetyczne należą do algorytmów ewolucyjnych inspirowanych naturą i zjawiskami występującymi w przyrodzie (jak np. poszukiwanie pożywienia przez kolonię mrówek). Generują one rozwiązania suboptymalne, co oznacza spełnienie warunków zadania, natomiast nie gwarantuje on odnalezienia rozwiązania optymalnego.

W literaturze opisano przykłady wykorzystania algorytmów genetycznych w wielu dziedzinach. W biologii wykorzystuje się je do poszukiwania struktur RNA (problem wyszukiwania najlepszych dopasowań strukturalnych, [6]). W pracy [7] zaproponowano nowe podejście, inspirowane zachowaniem insektów, które pozwoliło na efektywną implementację poszukiwań w lokalnych otoczeniach. W [8] zaprezentowano wykorzystanie algorytmu genetycznego do zautomatyzowania mechanizmów wspomagających projektowanie systemów sterowania (ang. *Computer-Aided Control System Design*, CACSD). Algorytmy genetyczne zastosowano również do rozwiązywania problemów związanych z automatycznym rozmieszczaniem elementów w zadanej, ograniczonej przestrzeni (jak np. podczas optymalizacji struktur w krzemie [9, 10]).

8.3.1. Podstawowe pojęcia

Istnieje kilka podstawowych pojęć, które są stosowane w opisach działania algorytmów genetycznych. Poniżej wymieniono najważniejsze z nich.

Populacja jest zbiorem *osobników*. Każdy z osobników posiada *chromosom*, który przechowuje *genotyp* danego osobnika. Natomiast *fenotypem* nazywamy zbiór interpretowalnych cech opisujących osobnika. Genotyp jest uporządkowaną strukturą kodującą tę informację. Ocena przystosowania danego osobnika do kierunku, w jakim ewoluuje populacja jest wyliczana *funkcją przystosowania*. Funkcja ta może być utożsamiona z funkcją celu zagadnienia optymalizacji. Ponadto każdy z genotypów musi mieć jednoznacznie zakodowane rozwiązanie, co pozwala na rzutowanie każdego genu w chromosomie na przestrzeń stanów.

8.3.2. Mechanizmy zachodzące w populacji

Wyróżnia się dwa podstawowe mechanizmy algorytmów genetycznych:

- krzyżowanie – polega na wymianie podciągów bitowych pomiędzy dwoma osobnikami. Jedną z metod implementacji tego mechanizmu jest losowanie punktu krzyżowania, a następnie wymiana ostatnich podciągów bitowych po-

między osobnikami. Proces ten posiada interpretację w świecie rzeczywistym odnoszącą się do wydawaniem na świat potomstwa.
- mutację – występuje rzadziej niż krzyżowanie, ale za to pełni bardzo istotną rolę w rozwiązywaniu problemów optymalizacyjnych. Sporadyczna mutacja może przyczynić się do eksploracji obiecujących obszarów przestrzeni stanów, do których krzyżowanie mogłoby nie doprowadzić.

8.3.3. Generowanie nowej populacji

Podstawowym etapem działania każdego algorytmu genetycznego jest ocena osobników danej populacji i wygenerowanie nowej. Osobniki, które posiadają wyższą wartość funkcji oceny mają większą szansę na przejście do nowej generacji. Mechanizm ten pozwala promować coraz to lepsze rozwiązania – osobników, którzy są coraz lepiej przystosowani do nałożonych warunków.

Do wygenerowania nowej populacji można posłużyć się mechanizmem ruletki [11]. Polega on na przydzielaniu osobnikom odpowiedniego prawdopodobieństwa ponownego wylosowania, proporcjonalnego do wartości funkcji oceny. Jest to proces losowy, który promuje kandydatów posiadających wyższą wartość funkcji przystosowania. Jednocześnie osobniki słabiej przystosowane posiadają pewien udział w nowej populacji. Jest to istotne, aby nie utknąć w minimum lokalnym, jak to może mieć miejsce w metodach gradientowych [12].

8.4. Problem zagospodarowania przestrzeni wystawowej

8.4.1. Założenia

Problem zagospodarowania przestrzeni wystawowej można zdefiniować w następujący sposób. Niech w_h oraz h_h będą, odpowiednio, szerokością i wysokością przestrzeni wystawowej. W przestrzeni tej mogą pojawić się elementy dodatkowe – prostokąty reprezentujące obszary zabronione. Obszary zabronione opisywane są czwórką $[x_z, y_z, w_z, h_z]$, gdzie: x_z – współrzędna X'owa środka obszaru zabronionego, y_z – współrzędna Y'owa środka obszaru zabronionego, w_z – szerokość obszaru zabronionego,h_z – wysokość obszaru zabronionego. Aby lepiej oddać rzeczywisty kształt hali wystawowej przyjęto, że obszary zabronione mogą pokrywać się.

Parametry hali wystawowej oraz obszarów zabronionych są danymi wejściowymi do algorytmu. Do danych wejściowych należą również parametry prostokątnych pawilonów handlowych. Każdy z nich opisywany jest dwójką $[w_b, h_b]$, gdzie: w_b – zadana szerokość pawilonu handlowego, h_b – zadana wysokość pawilonu handlowego. Pawilony te powinny być tak rozlokowane, aby nie kolidowały z obszarami zabronionymi oraz aby mieściły się w przestrzeni wystawowej.

Podsumowując, w problemie zagospodarowania przestrzeni wystawowej danymi wejściowymi są: wielkości hali, zbiór obszarów zabronionych, zbiór zadanych wielkości pawilonów handlowych. Poszukiwane jest rozlokowanie pawilonów handlowych. Do rozwiązania tego problemu wybrano algorytm genetyczny. Aby można go było wykorzystać, problem wymaga odpowiedniego zakodowania.

8.4.2. Projekt algorytmu

Etapem przygotowawczym przed uruchomieniem algorytmu jest zakodowanie zbioru ograniczeń i zbioru danych do postaci interpretowalnej przez algorytm. Dla przypomnienia: genotyp każdego osobnika zawiera informacje o rozwiązaniu problemu i jest w późniejszych etapach działania algorytmu odpowiednio interpretowany (na jego podstawie wyliczana jest funkcja przystosowania).

Istnieją dwie metody zapisu informacji w osobniku: *zapis stanu* oraz *zapis sekwencja operacji prowadząca do danego stanu.* Najbardziej intuicyjne wydaje się być reprezentowanie wiedzy zawartej w danym osobniku w formie stanu – zakodowanych pozycji wszystkich pawilonów handlowych w hali wystawowej. Wiele osobników reprezentuje wiele stanów, które interpretowane są jako różne rozmieszczenia pawilonów w hali.

Innym sposobem reprezentacji informacji przez osobnika jest wykorzystanie sekwencji ruchów. Zakłada się, że plansza przyjmuje stan początkowy z wyliczoną funkcją przystosowania. Każdy z kandydatów w populacji posiada zakodowaną sekwencję ruchów, która jest interpretowana jako operacje translacji i rotacji wykonywane na pojedynczych pawilonach od stanu początkowego do bieżącego. Obliczanie funkcji przystosowania wymaga więc wyliczenia stanu bieżącego na podstawie tych informacji indywidualnie dla każdego z osobników.

Prezentowane dalej opisy dotyczą przebiegu pojedynczej iteracji algorytmu według pierwszego z wymienionych sposobów.

Wczytywanie danych
Zakładając, że informacja w osobniku zapisywany jest stan hali wystawowej, konieczne jest przyjęcie współczynnika rozdzielczości. Ów współczynnik pozwala na dyskretyzację położenia pawilonu na hali oraz późniejszą ujednoliconą interpretację. W opisywanym przypadku przyjęto, że wartość tego współczynnika wynosi 16. Wartość ta jest interpretowana jako liczba bitów potrzebnych na zapisanie pozycji na osi X'ów, bądź Y'ów.

Zatem liczba bitów potrzebnych na zapisanie całościowej informacji o położeniu pojedynczego pawilonu wynosi 33. Zapisana zostaje w nim pozycja wraz z interpretacją, na którą przypada pojedynczy bit, a która jest później rozpoznawana jako brak rotacji (dla wartości 0) lub obrót o 90° (dla wartości 1).

Chromosom zapisywano jako wektor bajtów celem umożliwienia późniejszego zwiększenia rozdzielczości. Taki zapis pozwala znacznie uprościć przeprowadzane operacje i jednocześnie pozostawiało dużą elastyczność podczas manipulacji rozdzielczością (na wypadek dokładniejszego pozycjonowania). Stąd też całkowita szerokość chromosomu wyrażona w bajtach wynosi $33 \cdot N$, gdzie N to liczba pawilonów.

Generowanie populacji początkowej
Algorytm w początkowej fazie tworzy zadaną liczbę osobników o zdefiniowanej szerokości chromosomu. Następnie każdy z genów zapisywany jest losową wartością 0 lub 1 przy użyciu binarnego generatora liczb pseudolosowych [13]. Etap ten jest wykonywany jednorazowo przed rozpoczęciem działania osobników.

Realizacja mechanizmów selekcji
Pierwszym z etapów w selekcji jest krzyżowanie. Losowane są dwa osobniki, które będą podlegać wymianie części genów. Po wybraniu owych kandydatów z populacji następuje losowanie punktu krzyżowania (tzw. *locusu* [14]), po czym losowana jest liczba genów, które mają zostać zamienione miejscami. Po takiej operacji tworzone są dwa nowe osobniki, które różnią się od swoich rodziców. Kandydaci dodawani są do nowej populacji.

Drugi z etapów polega na mutacji genów. Proces ten jest realizowany przez zamianę pojedynczego genu na jego stan przeciwny, tzn. w przypadku występowania 0 wartość ta zmieniana jest na 1 oraz analogicznie dla wystąpienia 1. Częstość mutacji opisywana jest prawdopodobieństwem jej zajścia. Możliwa jest zmiana tego parametru za pomocą interfejsu aplikacji.

Po wykonaniu dwóch powyższych operacji na całym zbiorze populacji każdy z osobników jest oceniany. Za funkcję oceny przyjęto sumę odległości pomiędzy wszystkimi pawilonami. W przypadku, w którym pawilony zachodzą na siebie, częściowo lub w całości znajdują się na obszarze zabronionym, bądź wychodzą poza halę, odległość ta jest przemnażana przez -1. Po wyliczeniu wszystkich odległości są one sumowane, natomiast gdy choć jedna z nich jest ujemna, to brane są wyłącznie takie pod uwagę. Wówczas sumowana jest odwrotność tych odległości. Taka konstrukcja funkcji kryterialnej promuje rozwiązania, w których pawilony są jak najbardziej od siebie oddalone. Wartość funkcji celu zapisywana jest do dalszej części – selekcji. Na podstawie wyników otrzymanych konstruowany jest proces ruletki opisany w 8.3.3. Dzięki niemu możliwe jest wyłonienie statystycznie większościowej liczby z wyższą funkcją celu, natomiast dzięki temu, że jest to proces losowy kandydaci słabsi również są ujęci w tworzeniu nowej populacji.

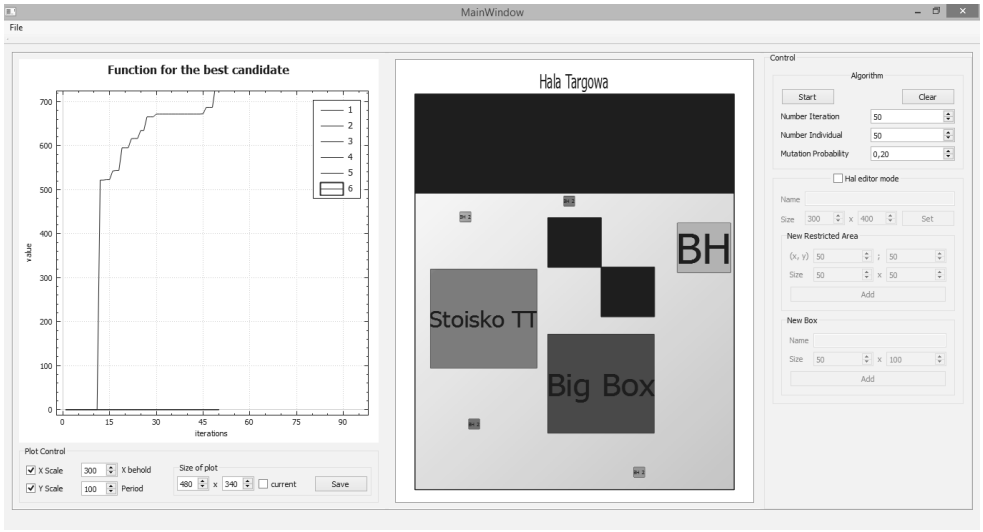
8.5. Opis wykonanej aplikacji

Aby przeprowadzić symulacje i zaobserwować wyniki działania algorytmu opisanego w sekcji 8.3 zaprojektowano prostą aplikację. Aplikacja ta dostarcza wygodnego interfejsu do wprowadzania danych wejściowych w formie graficznej oraz reprezentacji wartości funkcji celu dla najlepszego osobnika (główne okno aplikacji przedstawiono na rysunku 8.1). Dodatkowym atutem aplikacji jest możliwość jej uruchamiania pod systemami Linux, jak i Windows. Jest to zasługą wykorzystania bibliotek Qt, które to posiadają wsparcie multiplatformowe.

8.5.1. Wprowadzanie początkowej konfiguracji hali

Przed wystartowaniem algorytmu genetycznego konieczne jest wprowadzenie następujących danych wejściowych:

- wymiary oraz nazwa hali,
- wymiary oraz położenia obszarów zabronionych,
- wymiary oraz nazwy pawilonów.



Rys. 8.1: Interfejs graficzny uruchomionego programu.

Dostępne są 2 sposoby wczytania tych informacji: z pliku o formacie XML oraz przez wbudowanych edytor graficzny. Pierwsza z tych opcji zapewnia sprawdzanie poprawności danych pod względem strukturalnym poprzez walidację wczytanego pliku za pomocą dostarczonego schematu (pliku XML Schema). Graficzny edytor jest alternatywą do tej opcji. Uaktywnia się po zaznaczeniu pola wyboru „Hall editor mode". Znajdują się w nim 3 pola, na których można wprowadzić odpowiednio nazwę i wymiary hali, położenie i wymiary obszarów zabronionych („Restricted Areas"), wymiary pawilonów („Box"). Po kliknięciu przycisku „Set" lub „Add" na podglądzie graficznym ukazuje się wstępne rozlokowanie elementów hali (widok z góry). Po odznaczeniu pola wyboru „Hall editor mode" można wystartować algorytm przyciskiem „Start" lub usunąć bieżącą konfigurację przyciskiem „Clear". Dostępne dla użytkownika parametry symulacji to liczba iteracji („Number Iteration"), liczba osobników („Number Individual"), prawdopodobieństwo mutacji („Mutation Probability").

8.5.2. Obserwowanie wyników działania aplikacji

Po uruchomieniu algorytmu na podglądzie graficznym pojawia się otrzymane rozmieszczenie pawilonów w hali. Czarnym kolorem zaznaczone są obszary zabronione. Możliwa jest ręczna korekcja ustawienia za pomocą myszki (należy przeciągnąć odpowiedni element w pożądane miejsce). Wykres z lewej strony prezentuje wartość funkcji celu dla najlepszego osobnika w kolejnych iteracjach algorytmu. Można zmienić jego nazwę bądź też opisy osi klikając na nie dwukrotnie. Aby powiększyć lub pomniejszyć fragment wykresu należy kliknąć na oś, wzdłuż której nastąpi ta akcja i skorzystać z rolki myszy. Skalowanie wykresu w trakcie działania algorytmu wobec osi X i Y zapewniają pola wyboru „X Scale" i „Y Scale". W polu „X Behold" można ustawić liczbę pamiętanych próbek, „Pe-

riod" odpowiada za zakres widoczny na ekranie. Wykres można też zapisać ustalając jego wymiary bądź skorzystać z aktualnie wyświetlonego fragmentu (przycisk „Save").

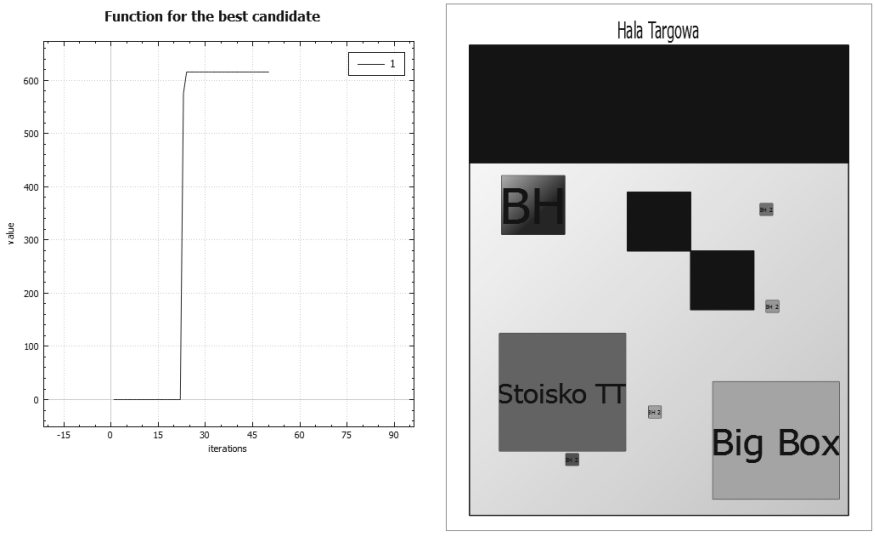
8.5.3. Przykład działania

Przykład działania aplikacji zademonstrowano dla następujących parametrów wejściowych:

- wymiary hali: 300 m (szerokość), 400 m (wysokość),
- liczba pawilonów: 7,
- liczba obszarów zabronionych: 3,
- parametry algorytmu:
  - liczba iteracji: 50,
  - liczba osobników: 50,
  - prawdopodobieństwo mutacji: 0.2.

Wynik działania dla powyższych parametrów pokazano na rysunku 8.2. Obszar hali jest reprezentowany jako kontur, w którym znajdują się wszystkie rozważane elementy: obszary zabronione i pawilony handlowe. Obszary zabronione są widoczne jako czarne prostokąty, natomiast pawilony mają zróżnicowane kolory (na wydruku - odcienie szarości) oraz nazwę.

Na wykresie (po lewej) przedstawiono wynik funkcji przystosowania dla najlepszego z osobników w danej iteracji. Okazuje się, że pomimo zróżnicowanej instancji problemu, algorytm już w około 22 kroku potrafi znaleźć rozwiązanie, które następnie stara się optymalizować.



Rys. 8.2: Wynik działania aplikacji.

8.6. Podsumowanie

Wykonana praca pokazała, że możliwe jest sprowadzenie zagadnienia CSP do postaci akceptowalnej przez algorytm genetyczny. Metody ewolucyjne służą do rozwiązywania szerokiego grona problemów, szczególnie dla takich, których formalna definicja jest ciężka do uzyskania. Niektóre problemy nie posiadają dedykowanych algorytmów, które pozwoliłyby na ich rozwiązanie. Jednakże, wykorzystując algorytmy genetyczne możliwe jest uzyskanie suboptymalnych wyników dla danego problemu. Wymagane jest jedynie odpowiednie sprowadzenie reprezentacji problemu do zagadnienia rozwiązywanego przez algorytm. Przykładem takiego postępowania jest niniejsza implementacja algorytmu genetycznego dla problemu rozmieszczania pawilonów w hali. Można zdefiniować formalny opis tego zagadnienia, jednakże ograniczenia CSP zostały w prosty sposób przełożone na funkcję przystosowania. Wiele problemów podlega pod definicję CSP, zatem, aby rozwiązać dane zagadnienie za pomocą algorytmu genetycznego wymagane jest jedynie zdefiniowanie odpowiedniej funkcji celu oraz sposób reprezentacji osobnika.

Literatura

[1] P. Hofstedt. Constraint-based object-oriented programming. *IEEE Software*, 27(5):53–56, 2010.

[2] B. Sun, W. Wang, Q. Qi. Satellites scheduling algorithm based on dynamic constraint satisfaction problem. *Computer Science and Software Engineering, International Conference on*, 4:167–170, 2008.

[3] K. Mizuno, Y. Fukui, S. Nishihara. Urban traffic signal control based on distributed constraint satisfaction. *47th Hawaii International Conference on System Sciences*, 0:65, 2008.

[4] C.H. Cheng, B.R. Feiring, T.C.E. Cheng. The cutting stock problem – a survey. *International Journal of Production Economics*, 36(3):291–305, 1994.

[5] E. Tsang. Foundations of constraint satisfaction, 1993.

[6] F.H.D. Van Batenburg, A.P. Gultyaev, C.W.A. Pleij. An APL-programmed genetic algorithm for the prediction of RNA secondary structure . *Journal of Theoretical Biology*, 174(3):269–280, 1995.

[7] H. Iba, T. Higuchi, H. De Garis, T. Sato. Evolutionary learning strategy using bug-based search. *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'93, strony 960–966, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[8] Y. Li, K.H. Ang, G.C.Y. Chong, W. Feng, K.C. Tan, H. Kashiwagi. Cautocsd-evolutionary search and optimisation enabled computer automated control system design. *International Journal of Automation and Computing*, 1(1):76–88, Październik 2004.

[9] Z. Cai, Q. Cao, Sh. Du, X. Ji. The genetic algorithms in optimization of silicon clusters. *Natural Computation (ICNC), 2012 Eighth International Conference on*, strony 1247–1250, Maj 2012.

[10] J.A. Vinoth, K. Batri. Layout problem optimization in vlsi circuits using genetic algorithm. *International Journal of Science, Engineering and Technology Research (IJSETR)*, 3(4):711–716, Kwiecień 2014.

[11] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and MAchine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[12] A. Stachurski, A.P. Wierzbicki. *Podstawy optymalizacji*. Oficyna Wydawnicza Politechniki Warszawskiej, 2001.

[13] J. Stadniczki. *Teoria i praktyka rozwiązywania zadań optymalizacji z przykładami zastosowań technicznych*. Wydawnictow Naukowo-Techniczne, 2006.

[14] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictow Naukowo-Techniczne, 1999.

ROZDZIAŁ 9

PROJEKT JĘZYKA DZIEDZINOWEGO ZANURZONEGO W SKŁADNI LaTeX

M. Kolasa, M. Patro

"Czy w dziejach ludzkości istnieje odkrycie bardziej rewolucyjne niż wynalezienie pisma? Prostota jego elementów, łatwość rozprzestrzeniania się na rozległych obszarach oraz trwałość utrwalonych dzięki niemu informacji sprawiły, że pismo odegrało fundamentalną rolę w historii świata." [1]

"Traktowane jako wynalazek pismo jest zawsze związane z językiem, jest środkiem utrwalenia go, jest narzędziem utrzymania tego, co płynne, przejściowe, wynika z najwyższych zdolności człowieka. Dalszym krokiem jest problem przetwarzania danych. Poddaje się obróbce lub przetwarzaniu dane utworzone ze znaków pisma, to znaczy przeobraża się je zgodnie z instrukcją, co wiedzie do uogólniającej abstrakcji." [2]

Od początku rozwoju cywilizacji człowiek zajmował się gromadzeniem zdobytej wiedzy. Początkowo wiedza ta była na tyle ograniczona, że wystarczającym narzędziem do jej utrzymania były międzypokoleniowe przekazy ustne. Wraz z postępem jej zakres zwiększał się, co doprowadziło do opracowania metod zapisu informacji, w tym pisma – od postaci piktograficznej do pisma alfabetycznego.

Ważnym krokiem na ścieżce rozwoju było opracowanie zaawansowanych formalizmów (matematycznych, fizycznych, chemicznych itp), które pozwoliły opisywać świat w sposób bardziej uporządkowany. Dzięki nim poznawanie świata, przetwarzanie informacji i rozwijanie badań stało się efektywniejsze niż kiedykolwiek wcześniej

Pojawienie się języków programowania w zastosowaniu do zapisu i przetwarzania informacji nastąpiło stosunkowo niedawno. Zdarzenie to zapoczątkowało etap projektowania języków przeznaczonych do zapisania posiadanej wiedzy (wyrażeń matematycznych, reguł, algorytmów itp.) w sposób zrozumiały dla komputerów.

Języki programowania są językami zbyt ogólnymi, aby można było za ich pomocą opisywać bardziej rozwinięte abstrakcje. Do opisywania zagadnień wysoce wyspecjalizowanych i ukierunkowanych na konkretne zadania i problemy

wdraża się języki dziedzinowe, nazywane też językami dedykowanymi bądź specjalizowanymi (ang. *Domain Specific Languages*, DSL).

W niniejszym rozdziale przedstawiono zagadnienia związane z tworzeniem i wykorzystaniem języków dziedzinowych. Na początku objaśniono, czym są te języki. Następnie omówiono mechanizmy ich tworzenia oraz przedstawiono dziedziny ich zastosowań. Na koniec zaprezentowano własny projekt języka dziedzinowego zanurzonego w składni LaTeX oraz implementację jego prekompilatora. Prekompilator ten jest odpowiedzialny za generowanie i wstawianie do dokumentów obrazów przedstawiających konfigurację manipulatorów o $n \leq 7$ stopniach swobody dla zadanych parametrów Denavita-Hartenberga.

9.1. Języki dziedzinowe

9.1.1. Podstawy

Realizacja języków dziedzinowych różni się w zasadniczy sposób od tradycyjnych języków programowania. Języki te są dostosowane do konkretnej dziedziny i oferują znacznie większą siłę wyrazu i prostotę użytkowania w konkretnych zastosowaniach [3, 4]. Ponadto są one zazwyczaj niewielkie. Zawierają tylko ograniczone zestawy notacji i abstrakcji z danej dziedziny. Typowym przykładem użycia języka dziedzinowego jest tworzenie formuł w arkuszu kalkulacyjnym [5].

Języki dziedzinowe opracowywane jako zupełnie nowe, bez związków z innymi językami, nazywane są zewnętrznymi. Ich opracowanie wiąże się z takimi etapami, jak: analiza leksykalna, interpretacja, kompilacja i generowanie kodu. Czasami jednak języki dziedzinowe powstają na bazie składni istniejącego języka (np. języka programowania). Język dedykowany traktowany jest wtedy jako podjęzyk rozszerzający język ogólny o wyrażenia specjalizowane. Języki korzystające z infrastruktury języka bazowego nazywane są wewnętrznymi. Są one zazwyczaj realizowane w postaci bibliotek dla języka bazowego [6, 5].

Tworzenie języków specjalizowanych jest opłacalne, gdy ułatwiają on rozwiązywanie problemów, a także wtedy, gdy dany problem pojawiają się często. Rozwój języków dziedzinowych jest trudny, ponieważ wymaga jednocześnie wiedzy na temat konkretnej dziedziny, jak procesu rozwoju języka. Niewiele osób posiada wystarczająco duże kompetencje w tych obszarach, dlatego często decyzja o rozwoju języków dziedzinowych odraczana jest w nieskończoność [5, 3].

Zaletami wykorzystywania języków specjalizowanych są, między innymi:

- możliwość tworzenia zapisów w składni zbliżonej do naukowego formalizmu stosowanego przez ekspertów w danej dziedzinie,
- możliwość odpowiedniego raportowania błędów,
- możliwość analizy, optymalizacji i transformacji specyficznych dla domeny,
- możliwość wykorzystania istniejących języków, bibliotek,
- zmniejszenie czasu potrzebnego na wykonanie projektu.

Do wad stosowania języków specjalizowanych zalicza się:

- koszty projektowania, wdrażania i utrzymania,

- trudność przy rozszerzaniu,
- koszty kształcenia użytkowników,
- problem z określeniem zakresu dziedziny,
- ograniczone zastosowania.

9.1.2. Rozwój i wykorzystanie

Rozwój języków dziedzinowych zazwyczaj obejmuje następujące etapy: decyzja, analiza, projektowanie, realizacja i wdrożenie. Etap decyzja jest jedyny, w którym rozważa się przyczyny wdrożenia języka specjalizowanego, z uwzględnieniem celów długoterminowych wraz z czynnikami ekonomicznymi. W etapie analizy identyfikowana jest dziedzina problemu i gromadzona jest wiedza specjalistyczna. Etap ten wymaga zaangażowania ekspertów z rozważanej dziedziny, a także wzięcia pod uwagę dostępnej dokumentacji oraz kodu. Etap projektowania określa sposób projektowania języka: wewnętrzny lub zewnętrzny. W następującej fazie realizacji wybiera się odpowiednie podejście do realizacji, np: interpreter, prekompilator. Język może być zrealizowany jako pakiet bibliotek dla bazowego języka, source-code do samodzielnej kompilacji lub skrypt instalacyjny [5].

Decyzja o przyjęciu nowego języka dedykowanego zazwyczaj nie jest łatwa. Inwestycja w rozwój musi odpłacić się bardziej ekonomicznym wdrażaniem i konserwacją oprogramowania. Względy krótkoterminowe i brak doświadczenia mogą łatwo spowodować odroczenie lub wycofanie się z projektu.

Aby ułatwić podjęcie decyzji można skorzystać z kryteriów opisanych następującymi wzorcami: notacja, automatyzacja zadań, reprezentacja struktury danych, translacja struktury danych, system front-end, interakcja, AVOT [3]. Zasadniczo są to ogólne modele, łączące ze sobą takie kwestie, jak:

- ulepszenie ekonomi wytwarzania oprogramowania,
- umożliwienie programowania końcowemu użytkownikowi,
- umożliwienie analizy, weryfikacji, optymalizacji, transformacji.

Analiza wiąże się z identyfikacją dziedziny problemu i akwizycją potrzebnej wiedzy. Etap ten wymaga eksperckiego doświadczenia, dokumentacji i kodu, które można uzyskać z danej dziedziny wiedzy. Większość działań w czasie analizy jest nieformalna, ale czasami stosuje się formalne metody analizy:

- DARE (ang. *Domain Analysis and Reuse Environment*),
- DSSA (ang. *Domain-Specific Software Architectures*),
- FODA (ang. *Feature-Oriented Domain Analysis*),
- ODM (ang. *Organization Domain Modeling*).

Wyniki formalnej analizy dziedziny, choć mogą być bardzo zróżnicowane, dają pewnego rodzaju obraz pozyskanej wiedzy specjalistycznej [3].

Projektowanie w najprostszym przypadku może polegać na wykorzystaniu istniejącego języka. Jedną z zalet takiego podejścia jest dostarczenie rozwiązania

po części już znanego użytkownikom. Samo projektowanie w tym trybie może być realizowane na kilka sposobów. Można konstruować funkcje specyficzne dla dziedziny z istniejących już części języka lub je rozkładać na części możliwe do realizacji przez inne języki. Różnica pomiędzy tymi podejściami zależy od sztywności granicy pomiędzy językiem dziedzinowym a resztą wykorzystanych języków. Podejścia te są często wykorzystywane w przypadkach, gdy składnia języka jest już powszechnie znany, np: w językach wykorzystujących wyrażenia arytmetyczne.

Innym podejściem jest rozszerzenie istniejącego języka o całkowicie nowe funkcje. W większości zastosowań tego modelu istniejące funkcje języka bazowego pozostają dostępne. Wyzwaniem jest integracja funkcji dziedzinowych z resztą języka w jednolity sposób.

Całkowicie innym podejściem jest stworzenie języka dedykowanego, którego konstrukcja nie ma żadnego związku z istniejącymi rozwiązaniami. W praktyce tego rodzaju rozwiązanie jest bardzo trudne i skomplikowane do scharakteryzowania. Oprócz pamiętania o przejrzystości i prostocie projektant jednocześnie musi uwzględniać specyfikę dziedzinową języka i fakt, że użytkownicy nie muszą być programistami. Ponieważ najlepiej sprawdzają się już dojrzałe sposoby notacji, projektując język dziedzinowy należy starać się je raczej wykorzystać niż ulepszać.

Po ustaleniu relacji pomiędzy istniejącymi językami należy zająć się opracowaniem specyfikacji projektu. Rozróżniane są formalne i nieformalne projekty. W nieformalnym przypadku specyfikacja jest zazwyczaj jakąś formą języka naturalnego. Specyfikacja formalnego projektu będzie zapisana zgodnie z jakąś metodyką oraz technikami modelowania. Najczęściej stosowane notacje formalne, do specyfikacji składni i atrybutów gramatycznych, obejmują wyrażenia regularne i gramatykę.

Oczywiście, nieformalne podejście będzie prawdopodobnie najłatwiejsze dla większości ludzi. Jednak formalne podejście nie powinno być dyskryminowane. Nieformalne projekty językowe mogą zawierać niedokładności, które powodują problemy w fazie realizacji. Natomiast formalne projekty mogą być realizowane automatycznie przez narzędzia, co znacznie skraca i ułatwia wdrożenie [3].

Realizacja i wdrożenie następują już po zaprojektowaniu języka. Do narzędzi i metod stosowanych na tym etapie należą:

- interpreter,
- kompilator (generator aplikacji),
- preprocesor,
- techniki osadzania (ang. *embedding*),
- tworzenie rozwiązań hybrydowych.

Należy zauważyć, że interpreter i kompilator są często stosowanymi narzędziami, zarówno do przetwarzania języków dziedzinowych, jak i ogólnych. Specyfika języków dziedzinowych sprawia jednak, że można do nich stosować bar-

dziej efektywne metody realizacji (np. przetwarzanie wstępne (ang. *preprocesing*) czy osadzanie).

W metodzie osadzania realizacja języka następuje poprzez rozszerzenie istniejącego języka (po zdefiniowaniu dodatkowych abstrakcyjnych typów danych i operatorów można wykorzystać je razem z całą składnią języka bazowego). Zwolennicy metody osadzenia często krytykują stosowane w niej tradycyjne podejście do projektowania ze względu na duże nakłady pracy niezbędne do opracowania rozszerzonej składni. Z drugiej strony źle zaprojektowana semantyka języka może rodzić różne późniejsze problemy oraz ograniczać możliwość rozszerzania języka o nowe funkcje.

W językach dziedzinowych wykorzystywane są metody stosowane przy formalnym definiowaniu języków programowania (abstrakcyjna maszyna stanu, specyfikacja algebraiczna, denotacja semantyki, semantyka operacyjna itp.). Dobrze opracowany model powinien wspierać modularność i rozszerzalność specyfikacji. Ponadto najbardziej produktywny rozwój języków dziedzinowych odbywa się za pomocą programowych narzędzi, które automatycznie generują odpowiednie interpretery i kompilatory.

9.1.3. Wsparcie narzędziowe

Tworzenie języków dziedzinowych i ich wykorzystanie często jest wspierane przez narzędzia programowe. Poniżej wymieniono niektóre z tych narzędzi.

ANTLR jest narzędziem do generowania parserów, dostarczającym struktur do implementacji tłumaczy. W ANTLR wygenerowany parser może być zaimplementowany jako tłumacz w jednej z dwóch form: wykonujący akcje semantyczne i generujący program z użyciem szablonów. Proces tłumaczenia ma dwie fazy: pierwszą, w której dane wejściowe są analizowane (parsowane) i przekazywane jako argumenty do szablonów oraz drugą, w której generowany jest kod, a połączenia szablonów są mapowane do pojęć języka. ANTLR zapewnia niezbędne biblioteki do generowania parsera dla danej gramatyki. Aby wygenerować program docelowy, ANTLR wspiera wykorzystanie StringTemplate – silnika do generowania tekstu przy użyciu szablonów [7].

RUBY jest dynamicznym, obiektowo zorientowanym językiem programowania ogólnego przeznaczenia. W Ruby języki dziedzinowe są realizowane dzięki takim cechom, jak: abstrakcje lambda (bloki kodu), oceny, dynamiczne wpisywanie, odbicia i elastyczna składnia. W Ruby bloki kodu są zamknięte, co może być wykorzystywane do kodowania informacji specyficznych dla dziedziny [7].

Stratego/XT to programowa platforma transformacji zawierająca: język Stratego (do realizacji programu poprzez określenie transformacji przepisywania) oraz zestaw narzędzi XT (do zapewnienia infrastruktury do realizacji tych przekształceń). Stratego/XT realizuje transformacje poprzez reprezentowanie programów w postaci abstrakcyjnych drzew składni (zwanych *Annotated Terms*, ATerms) oraz stosowanie strategii i warunków przypisujących do nich reguły. W Stratego/XT

języki dziedzinowe realizowane są z wykorzystaniem potoków transformacji składających się z trzech etapów: parsowania (wykonywanego przez parser), transformacji (wykonywanego przez program transformujący z wykorzystaniem języka STRATEGO) i *pretty printing* (wykonywanego podczas parsowania ostatecznej formy ATerm do programu docelowego) [7].

Converge to nowoczesny język o bogatej składni, łączący w sobie cechy języka Python (typy danych) i szablonów Haskell. Języki dziedzinowe w Converge realizowane są z wykorzystaniem obiektów CTMP (ang. *compile-time meta-programming*). CTMP może być traktowane równoważnie do makr, gdyż zapewnia użytkownikowi mechanizm interakcji z kompilatorem (tj. budowę dowolnych fragmentów kodu programu) [7].

Eclipse Modeling Project ujednolica platformy związane z modelowaniem, narzędzia i standardy implementacji. Projekt ten składa się z [8]:

- Eclipse Modeling Framework – platforma modelowania i obiektów generacji kodu, pozwalająca na specyfikacje meta modeli i zarządzanie instancjami modeli,
- Graphical Editing Framework – pozwala na stworzenie edytora graficznego na podstawie istniejącego modelu aplikacji,
- Graphical Modeling Framework – zapewnia składniki generowania i wykonywania infrastruktury dla rozwoju edytorów graficznych opartych na EMF i GEF.

Microsoft DSL Tools to rozwiązanie firmy Microsoft pozwalające zaprojektować język dziedzinowy i wygenerować wszystko, co jest potrzebne użytkownikowi do tworzenia modeli bazujących na tym języku. Stosowane tu określenie „model domeny" jest odpowiednikiem metamodelu, składającego się z hierarchii klas i związków (prostych odniesień lub wbudowanych związków). Model posiada właściwie dwie role (źródłową i docelową) i może być supertypem. W DSL Tools do definiowania modeli domen służy DSL Designer – narzędzie do projektowania z graficznym interfejsem użytkownika [8].

Bison/Flex to para narzędzi służących do generowania skanerów, czyli programów rozpoznających wzory leksykalne w tekście. Dokładniej mówiąc: flex służy do generowania analizatorów leksykalnych, zaś bison do tworzenia analizatorów składni. Połączenie obu narzędzi daje możliwość zarówno analizy leksykalnej, jak i parsowania składni tworzonych języków, zarówno specyficznych dla danej domeny, jak i bardziej ogólnych (narzędzia o funkcjach podobnych oferowanych przez bison nazywane są też kompilatorami kompilatorów).

analiza leksykalna (ang. *lexical analysis*) – rozpoznawania różnych rodzajów elementów w strumieniu wejściowym;
analizator leksykalny lub *skaner* (ang. *scanner*) – program odczytujący dane wejściowe i zapewniający rozróżnianie występujących w nich składników (tzw. tokenów, ang. *tokens*);
rozbiór (ang. *parsing*) - rozpoznawanie struktury wyższego poziomu w se-

kwencji elementów w strumieniu wejściowym, czyli bloków, wyrażeń arytmetycznych, instrukcji przypisania itp.;
analizator składni lub *parser* (ang. *parser*) - program dokonujący rozbioru.

Flex generuje analizator leksykalny na podstawie opisu stworzonego w składni opartej na parach wyrażeń regularnych oraz fragmentach kodu. Opis dotyczy elementów (np. słów kluczowych, liczb i znaczników), które mają być rozpoznawane. Kod zaś ma być uruchomiony po napotkaniu rozpoznanego elementu. Wynikiem działania generatora jest kod zawierający odpowiednie funkcję analizujące składnią i wykonujące kod im odpowiadający [9].

Bison generuje parser konwertując zadaną gramatykę bezkontekstową na deterministyczny parser LR lub LALR [10]. Sposób tworzenia parserów jest analogiczny do sposobu wykorzystanego w programie flex. Programista ma możliwość zapisania odpowiednich przejść gramatyki oraz dodawania fragmentów kodu wykonujących się w trakcie analizy danego słowa dla wybranej gramatyki.

9.1.4. Obszary zastosowań

Języki dziedzinowe kojarzone są głównie z informatyką i obszarami z nią spokrewnionymi z uwagi. Powodem jest programowe wsparcie, jakie towarzyszy używaniu tych języków. Należy sobie jednak zdawać sprawę, że języki dziedzinowe nie są zamknięte tylko w sferze informatyki. Ich zastosowanie rozpina się na wiele gałęzi nauki i techniki. Graficzne języki dziedzinowe stosuje się zazwyczaj do wizualnego projektowania jakichś modeli (danych, procesów itp.). Tekstowe języki dziedzinowe najczęściej służą do definiowania reguł, algorytmów i procedur.

Języki te wykorzystywane są także w [11]: systemach wspomagania projektowania i wytwarzania, systemach monitoringu i planowania, narzędziach do modelowania i symulacji, podczas budowy graficznego interfejsu użytkownika, przy projektowaniu schematów i struktur danych itd. Za pomocą języków dziedzinowych można opisywać sekwencje genów [12], tworzyć muzykę [13], dokonywać obliczeń symbolicznych [14].

Przykłady konkretnych języków dziedzinowych wraz z obszarem ich wykorzystania przedstawiono w tabeli 9.1. Jest to zaledwie fragment dostępnych rozwiązań. Przy dokładniejszych poszukiwaniach może okazać się, że każda dziedzina wiedzy, niekoniecznie naukowa, posiada swój własny, specyficzny język.

9.2. Projekt języka dziedzinowego

Projektowanie języków dziedzinowych i ich wykorzystanie nie musi być wcale trudne. Aby to udowodnić postanowiono stworzyć własny język dziedzinowy, umożliwiający zapis konfiguracji manipulatorów sztywnych zgodnie z transformacjami Denaviť a-Hartenberg'a [16]. Język ten może zostać wykorzystany na wiele sposobów. W niniejszym rozdziale opisano jego zastosowanie polegające na definiowaniu parametrów niezbędnych do wygenerownia rysunków ze szkieletami ramion manipulatorów i dołączaniu wygenerowanych rysunków do dokumentów napisanych w systemie LaTeX.

Tab. 9.1: Przykłady języków dziedzinowych 3R [15, 12, 3].

Język	Dziedzina
RobotML	Modelowania programów robotów
BRIDE	Graficzny DLS dla programistów ROS
GenoCAD	Projektowanie syntetycznych sekwencji DNA
Make	Budowanie oprogramowania
SQL	Zapytania baz danych
VHDL	Projektowanie hardware'u
BoX	Prettyprinting
CodeBoost	Optymalizacja języka C++
GAL	Tworzenie sterowników urządzeń wideo
LaCon	Tłumaczenie modelu danych
Cubix	Wirtualne magazynowanie danych

9.2.1. Składnia języka

Projektowany język dziedzinowy musi mieć składnie zgodną z istniejącym już językiem LaTeX(czyli ma to być język wbudowany). Co więcej, wstawianie fragmentów zredagowanych w tym języku do dokumentów nie powinno zaburzać procesu ich kompilacji. Z drugiej strony język dziedzinowy powinien być deterministyczny i bezkontekstowy [9], przez co jego składnia może być w prosty sposób interpretowana przez parsery typu LR tworzone przez generator Bison.

Zgodnie z tymi założeniami stworzono język charakteryzujący się następującą składnią (zapisaną za pomocą formalizmu opisanego w książce [9]):

- language → robots
- robots → robots robot
- robots → .
- robot → TBEGIN TSRC intructions TEND
- instructions → instructions instruction
- instructions → .
- instruction → segment
- instruction → viewer
- instruction → angletype
- segmet → TFLOAT TSPACER TFLOAT TSPACER TFLOAT TSPACER TFLOAT
- viewer → TFLOAT TSPACER TFLOAT TSPACER TFLOAT
- angletype → TRAD
- angletype → TDEG

Elementy oznaczone dużymi literami w składni języka to terminale — wyrażenia, które są nierozkładalne, znak "." oznacza wyrażenie puste. Terminale posiadają swoją jednoznaczną reprezentację:

- TBEGIN — `\begin{robotics}`,

- TSCR — ciąg znaków z podanym adresem zapisu pliku,
- TEND — `\end{robotics}`,
- TFLOAT — liczba zmiennoprzecinkowa pojedynczej precyzji,
- TSPACER — przecinek,
- TRAD — `rad`
- TDEG — `deg`

Język wymusza na użytkowniku podanie poprawnego początku i zakończenia środowiska. Następnie na początku poprawnie zdefiniowanego środowiska musi znajdować się ścieżka zapisu wygenerowanego pliku. Wnętrze może zawierać opcjonalnie zadeklarowane przez użytkownika komendy związane z ustawieniem perspektywy (`\viewer{}`), kolejnych segmentów robota (`\segment{}`) oraz notacji związanej z zapisem kątów w stopniach (`\type{deg}`) lub radianach (`\type{rad}`).

Dodatkowo, ponieważ wykorzystywany jest LaTeX stworzono własny pakiet, którego celem jest wykrycie nazwy obrazka oraz w miejsce utworzonego środowiska wstawianie komendy `\includegraphics{sciezka}`. Do składni dodano również nieinterpretowany w trakcie przetwarzania języka dodatkowy argument przy rozpoczęciu odczytu ustawień związanych z robotem (TBEGIN), który umożliwia zmianę ustawień dotyczących obrazka z wykorzystaniem pakietu `includegraphics`. Przykładowy opis manipulatora składającego się z trzech segmentów widoczny jest we fragmencie kodu 9.1.

Listing 9.1: Przykładowy manipulator opisany w składni języka.

```
\begin{robotics}[width=14.0cm]{./img/manipulator_1.jpg}
        \type{deg}
        \viewer{1000,1000,1000}
        \segment{90,0,2,0}
        \segment{90,0,1.5,0}
        \segment{90,0,1,0}
\end{robotics}
```

Widać, iż niektóre elementy składni języka są opcjonalne. Z tego powodu należy zdefiniować domyślne ustawienia przyjmowane w razie braku preferowanych ustawień użytkownika:

- ustawienia `includegraphics` — domyślnie obraz ma mieć szerokość całej strony i dopasowaną wysokość,
- kąt domyślnie zapisywany jest w stopniach,
- brak zadanej perspektwy powoduje ustawienie wartości 1000,1000,1000,

9.2.2. Implementacja

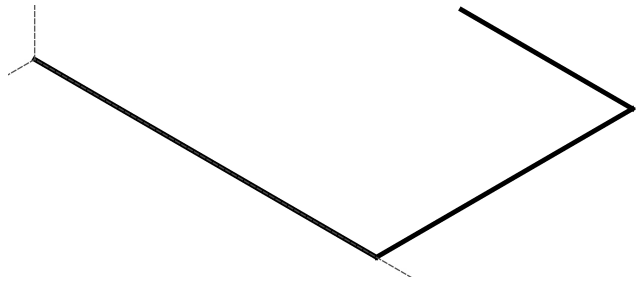
Do stworzenia analizatora leksykalnego i parsera języka wykorzystano narzędzia Bison oraz Flex. Otrzymano w ten sposób kod w języku *C++* realizujący analizę języka oraz tworzenie obiektów opisujących w sposób jednoznaczny zadany manipulator. W celu wizualizacji danych w postaci rysunków postanowiono wykorzystać biblioteki Qt `http://qt-project.org/`.

Proces tworzenia rysunków z otrzymanego pliku posiadającego opis manipulatorów w składni stworzonego jest złożony. Pierwszym etapem jest analiza leksykalna polegająca na przetwarzaniu ciągu danych wejściowych, znajdowaniu odpowiednich wyrażeń regularnych, oraz zapamiętywaniu ich w odpowiedni sposób i przekazywaniu do następnego etapu. Analiza leksykalna języka wykorzystuje oprogramowanie Flex i jest przedstawiona we fragmencie kodu 9.2.

Listing 9.2: Składnia leksykalna do programu Flex.

```
"\\begin{robotics}"     { BEGIN(FROBOTICS); startNumber =
    lineNumber; return TOKEN(TROBOTICS); }
[\n]                    lineNumber++;
.                       {  }

<DISPOSABLE>"]"         { BEGIN(FROBOTICS); }
<DISPOSABLE>[\n]        { lineNumber++; }
<DISPOSABLE>.           { }

<FROBOTICS>"["          { BEGIN(DISPOSABLE); }
<FROBOTICS>"{"          { BEGIN(SRC); }
<FROBOTICS>[\n]         { lineNumber++; }
<FROBOTICS>.            { }

<ROBOTICS>"\\end{robotics}" { BEGIN(INITIAL); return TOKEN(
    TEROBOTICS); }
<ROBOTICS>"\\segment{"      { BEGIN(STEP); }
<ROBOTICS>"\\viewer{"       { BEGIN(STEP); }
<ROBOTICS>"\\type{"         { BEGIN(TYPE); }
<ROBOTICS>[\n]              { lineNumber++; }
<ROBOTICS>.                 { }

<SRC>"}"                   { BEGIN(ROBOTICS); }
<SRC>[^}]+                 { SAVE_TOKEN; return TSRC; }

<STEP>"-"                  { return TOKEN(TMINUS); }
<STEP>[0-9]+               { SAVE_TOKEN; return TINT; }
<STEP>"."                  { return TOKEN(TPOINT); }
<STEP>","                  { return TOKEN(TSPACER); }
<STEP>"}"                  { BEGIN(ROBOTICS); }
<STEP>[\n]                 { lineNumber++; }
<STEP>.                    { }

<TYPE>"}"                   { BEGIN(ROBOTICS); }
<TYPE>"rad"                 { SAVE_TOKEN; return TRAD; }
<TYPE>"deg"                 { SAVE_TOKEN; return TDEG; }
<TYPE>.                 { }
```

Przeszukuje ona dane wejściowe ignorując wszystko do natrafienia na poprawne rozpoczęcie nowego manipulatora. Wtedy przechodzi do przeszukiwania części obowiązkowej poprawnie zdefinowanego manipulatora - definicji ścieżki do zapisu pliku wraz z formatem. Jeśli wcześniej znajdują się opcjonalne informacje



Rys. 9.1: Rysunek manipulatora odpowiadający przykładowemu fragmentowi kodu 9.1.

dla paczki `includegraphics` są one pomijane. Gdy zostanie poprawnie odczytana ścieżka do pliku, program przechodzi do analizy opcjonalnych lub zakończenia opisu manipulatora. Sprawdzane jest czy w manipulatorze został zdefiniowana perspektywa, kolejne segmenty manipulatora oraz sposób zapisu kątów. Znalezienie dodatkowej instrukcji powoduje przejście do analizy składni związanej z danym problemem. Natrafienie na zakończenie opisu manipulatora powoduje przejście do stanu początkowego analizatora leksykalnego i przeszukiwanie pliku w poszukiwaniu rozpoczęcia kolejnego opisu.

Kolejnym etapem jest parsowanie przeanalizowanych danych i tworzenie obiektów w C++ reprezentujących manipulatory. Odbywa się to przy pomocy *Bison*'a. Składnia parsera jest opisana analogicznie do składni języka, z uzupełnieniem o kod wykonywany podczas natrafienia na dany przypadek.

Ostatnim etapem jest wizualizacja obiektów. W celu wizualizacji danych w postaci rysunków postanowiono wykorzystać biblioteki Qt. Bibllioteki te dostarczają podstawowe struktury i metody pozwalające wyliczyć położenie i orientację poszczególnych elementów manipulatora w przestrzeni. Niestety Qt nie dostarcza mechanizmów pozwalających zapisać sceny 3D jako obraz. Problem ten rozwiązano poprzez zdefiniowanie odpowiednich operacji przekształcających i rzutowania na płaszczyznę [17]. Po przekształceniu i rzutowaniau obraz zapisywany jest w postaci obrazu 2D.

Wynikiem pracy oprogramowania robtex są rysunki manipulatorów opisanych w zadeklarowanej składni języka, zapisane w miejscach i pod nazwami zadanymi przez użytkownika. Przykładowy rysunek 9.1 przedstawia obiekt uzyskany przy wykorzystaniu przykładu 9.1.

9.2.3. Opis użycia

Wykorzystywanie stworzonego narzędzia jest analogiczne do popularnego przez użytkowników LaTeX'a narzędzia BibTeX, (`http://www.bibtex.org/`). Użytkownik najpierw parsuje dokument wykonanym przez autorów oprogramowaniem robtex, następnie korzysta z standardowych komend LaTeXdo wygenerowania końcowego dokumentu. Przykład poprawnego wywołania w konsoli widoczny jest we fragmencie kodu 9.3. Tak stworzone oprogramowanie umożliwia również stworzenie makr, które umożliwiają przejrzystość pracy dla użytkownika korzystającego ze standardowych aplikacji do tworzenia plików `.tex`. Przykład

takiego makra dla programu `Texmaker` widoczny jest we fragmencie kodu 9.4. Wykorzystywanie makra powoduje, że dokument tworzony jest oraz wyświetlany za jednym kliknięciem przycisku.

Listing 9.3: Przykład komend generujących dokument z wykorzystaniem robtex.

```
./robtex testowyplik.tex
pdflatex testowyplik.tex
```

Listing 9.4: Makro dla programu *Texmaker*.

```
sciezka_do_programu/robtex %.tex | pdflatex -synctex=1
        -interaction=nonstopmode %.tex | evince %.tex
```

Niezbędne jest, skopiowanie pliku `robotics.sty` do katalogu z wszystkimi pakietami związanymi z środowiskiem LaTeXlub do katalogu wraz z plikiem `.tex`.

9.3. Podsumowanie

Rozdział ten poświęcono językom dziedzinowym. Opisano w nim czym one są, jak i dlaczego powstają oraz gdzie są wykorzystywane. Ze względu na duży rozmiar tematu nie udało się go w pełni wyczerpać. Zarysowano jedynie jego zakres i wskazano kilku przykładowych przypadków użycia języków dziedzinowych, w tym przykład własnego rozwiązania.

Stworzony pakiet `Robotics` jest przykładem prostego języka dziedzinowego, interpretowanego przez dwa niezależne interpretatory. Dzięki swej konstrukcji daje on duże możliwości rozwoju oraz inwencji przy tworzenia podobnych rozwiązań. Może być traktowany jako dowód, że język dziedzinowy nie jest językiem zamkniętym, a współistnienie z innymi językami może być pożądaną cechą.

Ponieważ rozwój technologi i powiększanie się zasobów ludzkiej wiedzy postępuje obecnie w ogromnym tempie, można spodziewać się, że kolejne dziedziny staną się coraz bardziej zależne od komputerów i rozwiązań informatycznych, co napędzać będzie rozwój języków dziedzinowych.

Literatura

[1] F. Comte, E. Bekier. *Najsłynniejsze święte księgi świata: leksykon*. Vademecum - Opus. Opus, 1994.

[2] H. Kaufmann, M. Kęsy. *Dzieje komputerów*. Biblioteka Problemów. Państwowe Wydaw. Naukowe, 1980.

[3] M. Mernik, J. Heering, A.M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, Grudzień 2005.

[4] D. Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 2001.

[5] R. Jha, A. Samuel, A. Pawar, M. Kiruthika. A domain-specific language for discrete mathematics. *International Journal of Computer Applications*, 70(15):6–19, Maj 2013.

[6] A. van Deursen, P. Klint, J. Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Not.*, 2000.

[7] N. Vasudevan, L. Tratt. Comparative study of dsl tools. *Electronic Notes in Theoretical Computer Science*, 264(5):103–121, Lipiec 2011.

[8] T. Özgür. Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development. 2007.

[9] A.V. Aho, R. Sethi, J.D. Ullman. *Kompilatory. Reguły, metody i narzędzia*. WNT, 2002.

[10] J.E. Hopcroft, R. Motwani, J.D. Ullman. *Wprowadzenie do teorii automatów, języków i obliczeń*. WNT, 2005.

[11] Paul Hudak. Domain specific languages. *Handbook of Programming Languages, Vol. III: Little Languages and Tools, Chapter 3*, strony 39–60. MacMillan, Indianapolis, 1998.

[12] M.L. Wilson, S. Okumoto, J. Peccoud. Development of a domain-specific genetic language to design Chlamydomonas reinhardtii expression vectors. *Bioinformatics Advance Access*, 2013.

[13] A. Blackwell, N. Collins. The programming language as a musical instrument. *Proc. 17th Psychology of Programming Interest Group (PPIG) Workshop*, strony 120–130, Brighton, UK, Czerwiec 2005.

[14] R. Zitko. Sneg - mathematica package for symbolic calculations with second-quantization-operator expressions. 2011.

[15] T. Buchmann, J. Baumgartl, D. Henrich, B. Westfechtel. Towards a domain-specific language for pick-and-place applications. *CoRR*, 2014.

[16] K. Tchoń, A. Mazur, I. Dulęba, R. Hossa, R. Muszyński. *Manipulatory i roboty mobilne*. Akademicka Oficyna Wydawnicza PLJ, 2000.

[17] Reprezentacja przestrzeni trójwymiarowej na płaszczyźnie. `http://smurf.mimuw.edu.pl/external_slides/Reprezentacja_przestrzeni_trojwymiarowej_na_plaszczyznie/Reprezentacja_przestrzeni_trojwymiarowej_na_plaszc.html`, 2014.

ROZDZIAŁ 10

WYKORZYSTANIE LOGIKI TEMPORALNEJ DO WERYFIKACJI MODELU

F. Sobczak, M. Błędowski

10.1. Wprowadzenie

Obserwując rozwój naszej cywilizacji bez trudu można zauważyć jak daleko sięga informatyzacja i automatyzacja różnego rodzaju procesów i zjawisk. To dzięki nim zmienia się technologiczne oblicze otaczającego nas świata. Jednak żaden postęp w tych kierunkach nie byłby możliwy bez wcześniejszego opracowania metod formalnych, pozwalające modelować i opisywać rzeczywistość w postaci nadającej się do maszynowego przetwarzania.

Modelowanie polega na tworzeniu dokładnego opisu rzeczywistości, z pominięciem czynników niemających wpływu na badane zjawiska w danym kontekście (jak np. własności dynamicznych manipulatora przemysłowego podczas budowy modelu opisującego jego czynności w postaci następujących po sobie zdarzeń). Na bazie takiego opisu można uruchamiać symulacje, wyznaczać bieżące sterowania oraz dokonywać predykcji przyszłych zachowań różnorodnych systemów. Modelując jakiś system można opisać zależności panujące między jego zmiennymi oraz dopuszczalne zachowania. Opis ten można analizować pod różnymi kątami, zaczynając od weryfikacji jego poprawności, tj. zgodności modelu systemu z jego specyfikacją. U podstaw tych analiz leżą różne aparaty matematyczne (np. automaty skończone, sieci Petriego, logika temporalna).

W niniejszym rozdziale zostaną omówione cechy wybranych metod formalnych. Przedstawiony zostanie również przykład ich praktycznego wykorzystania do opisu systemu sterującego światłami na prostym skrzyżowaniu dróg.

10.2. Metody modelowania

Metody modelowania pozwalają opisać w sposób formalny i jednoznaczny wszystkie istotne cechy badanych obiektów i zjawisk. Opisane dalej automaty skończone, sieci Petriego i logika temporalna należą do uznanych i często stosowanych metod w tym obszarze.

10.2.1. Automaty skończone

W metodzie automatów skończonych (ang. *finite-state machine*, FSM) stan obiektu jest przedstawiany jako unikalna kombinacja wartości zmiennych, a jego zachowanie jako tranzycje – przejście między stanami. Wejściem automatu jest ciąg elementów należących do pewnego akceptowanego alfabetu. Maszyna zaczyna działanie w określonym stanie początkowym i interpretuje symbole wejściowe korzystając przy tym z funkcji przejścia wywołujących tranzycje (przejścia do następnych stanów). Jeśli po przeczytaniu całego ciągu wejściowego automat zakończy działanie w stanie oznaczonym jako akceptujący (końcowy), to słowo należy do języka rozpoznawanego przez maszynę. Automaty można podzielić na dwie kategorie:

- deterministyczne - przeczytanie danego elementu w konkretnym stanie powoduje przejście do określonego stanu następnego,
- niedeterministyczne - przeczytanie danego symbolu w konkretnym stanie może skutkować przejściem do różnych stanów następnych.

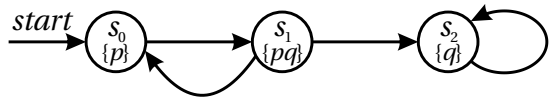
Każdy niedeterministyczny automat można sprowadzić do deterministycznego, akceptującego ten sam alfabet, poprzez zastosowaniu procesu determinizacji automatu skończonego. Należy jednak liczyć się ze znacznym wzrostem liczby stanów z n do nawet 2^n w najbardziej pesymistycznym przypadku.

Struktura Kripkego jest rodzajem niedeterministycznego automatu skończonego [1]. Reprezentuje ją graf skierowany, którego wierzchołki przedstawiają stany osiągalne przez system, a krawędzie tranzycje między stanami (rys. 10.1). Formalnie strukturę Kripkego definiuje się jako krotkę [2]

$$M = \langle S, I, \sigma, AP, L \rangle$$

gdzie:

S – to skończony zbiór stanów,
I – jest zbiorem stanów początkowych ($I \subseteq S$),
$\delta \subseteq S \times S$ – to relacja tranzycji taka, że $\forall s \in S \; \exists s' \in S$ takie, że $(s, s') \in R$,
AP – jest zbiorem zdań atomowych,
$L : S \to 2^{AP}$ – to funkcja interpretacji.



Rys. 10.1: Przykładowa struktura Kripkego: $S = \{s_0, s_1, s_2\}$, $I = \{s_0\}$, $\sigma = \{(s_0, s_1), (s_1, s_2), (s_1, s_0), (s_2, s_2)\}$, $AP = \{p, q\}$, $L(s_0) = \{p\}$, $L(s_1) = \{p, q\}$, $L(s_2) = \{q\}$.

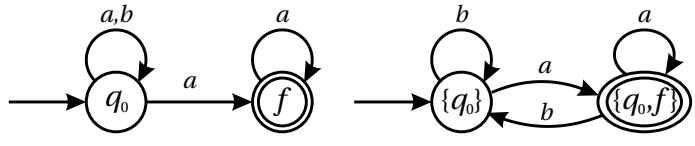
Trójka $\langle S, I, \delta \rangle$ nazywana jest często przestrzenią stanów systemu lub grafem osiągalnym systemu. AP oraz L dołączają do stanów informacje zdefiniowane przez użytkownika.

Struktura Kripkego reprezentuje wszystkie możliwe ścieżki przejść pomiędzy stanami modelowanego nią systemu. Startując ze stanu początkowego można zbudować drzewo obliczeniowe, w którym kolejne węzły wyprowadzane z węzła bieżącego będą reprezentować wszystkie stany, dla których istnieje tranzycja (przejście). Ponieważ w strukturze Kripkego istnieją cykle, budowane drzewo będzie nieskończone.

Automat Büchiego jest rozwinięciem automatu skończonego przyjmującym nieskończone ciągi wejściowe (jest ω-automatem) [3]. Sekwencja wejściowa jest zaakceptowana tylko i wyłącznie jeśli w trakcie działania maszyny, przynajmniej jeden ze stanów akceptujących jest odwiedzany nieskończenie często. Istnieją dwie wersje tego automatu: deterministyczna i niedeterministyczna (rys. 10.2). Zazwyczaj mówi się o wersji niedeterministycznej, opisanej krotką

$$M = \langle Q, \Sigma, \Delta, q_0, F \rangle$$

gdzie:
Q – skończony zbiór stanów,
Σ – skończony zbiór (alfabet),
$\Delta : Q \times \Sigma \to Q$ – relacja tranzycji zwracająca zbiór stanów,
q_0 – stan początkowy,
$F \subseteq Q$ – zbiór stanów akceptujących.



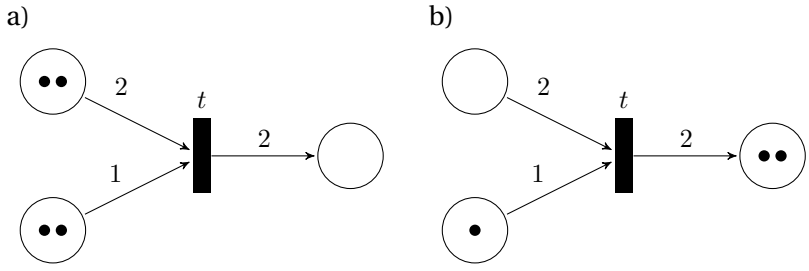
Rys. 10.2: Niedeterministyczny (z lewej) i deterministyczny (z prawej) automat Büchiego.

Sieć Petriego to formalizm uogólniający teorię automatów wykorzystywany do modelowania współbieżnych zdarzeń zachodzących w systemach rzeczywistych. Daną sieć można reprezentować za pomocą trójki [4]

$$M = \langle P, T, F \rangle$$

gdzie:
P – skończony zbiór miejsc,
T – skończony zbiór tranzycji,
$F \subset (P \times T) \cup (T \times P)$ – zbiór ukierunkowanych krawędzi.

Do opisu aktualnego stanu sieci wykorzystuje się dodatkowo tokeny znajdujące się w wierzchołkach grafu. Miejsca poprzedzające tranzycje nazywane są miejscami wejściowymi. Aby tranzycja mogła nastąpić, we wszystkich stanach



Rys. 10.3: Dwa kolejne stany sieci Petriego: a) przed tranzycją, b) po tranzycji.

ją poprzedzających muszą znajdować się tokeny. Ilość tokenów, jaka jest niezbędna do odpalenia (przejścia tokenów do kolejnego stanu) jest zdefiniowana przez wagi krawędziach wejściowych. Ponadto wagi krawędzi wyjściowych definiują ile tokenów pojawi się w wierzchołkach do których prowadzą w przypadku odpalenia tranzycji. Przykład działania sieci przedstawiono na rysunku 10.3.

10.2.2. Logika temporalna

Logika temporalna należy do grupy formalizmów wykorzystywanych do reprezentacji działania systemów. Z jej pomocą tworzy się specyfikacje określające sposób działania obiektu i weryfikuje się modele stworzone za pomocą automatów stanów skończonych. Weryfikacja taka jest możliwa, ponieważ specyfikację opisaną za pomocą formuł logicznych z wykorzystaniem logiki temporalnej można przekształcić na FSM [5, 6]. W takim wypadku stany traktowane są jako punkty w czasie, a poprawność przejść między nimi może być sprawdzana zgodnie z przyjętymi wcześniej logicznymi założeniami.

W procesie weryfikacji konieczne jest uwzględnienie przepływu czasu. Narzędzia dostarczone przez logikę klasyczną (z operatorami: $\neg, \wedge, \vee, \rightarrow$) nie dają możliwości charakteryzowania takich systemów. W większości przypadków wprowadzenie ciągłego czasu nie jest konieczne. Wystarczy dokonać dyskretyzacji i zapewnić, że zdarzenia będą następowały w określonym porządku. Do takich właśnie zastosowań stosuje się logikę temporalną. Biorąc pod uwagę reprezentację czasu logiki temporalne można podzielić na kilka opisanych niżej kategorii.

Liniowa logika temporalna (ang. *linear temporal logic*, LTL) zakłada, że każdy moment ma tylko jedną następującą po nim ścieżkę przepływu czasu. Rozszerza klasyczną logikę o dodatkowe operatory czasowe:

$\Box\phi$ - zawsze zachodzi ϕ,
$\Diamond\phi$ - w bieżącym stanie lub w przyszłości zajdzie ϕ,
$\bigcirc\phi$ - w następnej chwili zajdzie ϕ,
$\phi\mathcal{U}\psi$ - w przyszłości zajdzie ψ, do tego momentu zachodzi ϕ,
$\phi\mathcal{R}\psi$ - ψ zachodzi tak długo, aż zajdzie ϕ.

Ich wprowadzenie umożliwia przedstawianie takich własności, jak np.:

- żywotność – po każdym ϕ nastąpi ψ:
$$\Box(\phi \rightarrow \Diamond\psi),$$
- niezmienność – w pewnym momencie ϕ zawsze będzie prawdziwe:
$$\Diamond\Box\phi,$$
- bezpieczeństwo – ϕ nigdy nie nastąpi:
$$\Box\neg\phi,$$
- uczciwość – ϕ ma miejsce nieskończenie często:
$$\Box\Diamond\phi \rightarrow \psi,$$
- ϕ oscyluje co krok:
$$\Box((\phi \wedge \bigcirc\neg\phi) \vee (\neg\phi \wedge \bigcirc\phi)).$$

Zdanie: *„pracownik nie może być równocześnie kasjerem oraz presezem"* można wyrazić za pomocą liniowej logiki temporalnej w następujący sposób:

$$\Box\neg(pracownik = kasjer \wedge pracownik = prezes).$$

Logika drzew obliczeniowych (ang. *computation tree logic*, CTL) bazuje na rozgałęzionej strukturze czasu (dana chwila może rozwinąć się w przyszłości na wiele sposobów). Do operatorów LTL dochodzą w niej jeszcze dwa operatory ścieżkowe:

- Ap - dla każdej ścieżki zachodzi p (ang. *along All paths*)
- Ep - istnieje taka ścieżka, dla której zachodzi p (ang. *along Existing path*).

Aby przykładowe zdanie z poprzedniej sekcji obowiązywało na wszystkich ścieżkach, wystarczy dodać przed nie operator ścieżkowy:

$$\mathrm{A}\Box\neg(pracownik = kasjer \wedge pracownik = prezes).$$

10.3. Weryfikacja modelu

Jednym z prostszych sposobów weryfikacji poprawności działania modelu jest jego testowanie i symulacja. Metody te pozwalają wykazać obecność błędów, ale nie dają możliwości potwierdzenia ich braku (przez to nie nadają się do wykorzystywania w trakcie procesu weryfikacji). Współcześnie stosuje się metody oparte na weryfikacji zgodności modelu z jego specyfikacją.

Dla modelu M (reprezentowanego przez skończony automat stanów), stanu s (należącego do zbioru stanów początkowych $s \in S$) oraz specyfikacji φ (wyrażonej w języku logiki temporalnej) zadanie weryfikacji modelu polega na sprawdzeniu czy M spełnia/modeluje φ ($M, s \models \varphi$).

Proces weryfikacji modelu rozpoczyna się od przekształcenia zaprzeczenia specyfikacji LTL ($\neg\varphi$) na automat Büchiego $A_{\neg\varphi}$, który akceptuje wszystkie ciągi spełniające formułę $\neg\varphi$. W następnym kroku automat $A_{\neg\varphi}$ łączony jest z modelem podlegającym weryfikacji, tworząc $A_{M,\neg\varphi}$. Zgodnie z intuicją każdy ciąg akceptowany przez $A_{M,\neg\varphi}$ przedstawia przypadek, w którym model M narusza

zasady specyfikacji. Przypadek taki nazywany jest kontrprzypadkiem. Jeżeli nie zostanie on odnaleziony, to $M, s \models \varphi$.

Weryfikacja modelu przy użyciu logiki temporalnej może odbywać się:

- podejściem sprecyzowanym – polegające na rzeczywistym stworzeniu automatu Büchiego i postępowaniu zgodnie z przedstawionym powyżej wzorcem. W spotkaniu z kompleksowymi rzeczywistymi systemami metoda ta napotyka jednak poważny problem. Problem eksplozji stanów polega na wykładniczym rozrastaniu się przestrzeni stanów wraz ze wzrostem ilości zmiennych ze względu na kombinatoryczny charakter automatu.
- podejściem symbolicznym – aby załagodzić problem eksplozji stanów inaczej podchodzi się tu do reprezentacji modelu. Zamiast korzystać z pojedynczych stanów i tranzycji przedstawia się go w sposób symboliczny przy użyciu zbiorów stanów i tranzycji. Są one przedstawiane domyślnie, jako rozwiązanie formuły logicznej. Oszczędza to pamięć, jako że stosunkowo proste formuły mogą reprezentować duże zbiory stanów. Do wnioskowania z reguł opisujących przestrzeń stanów wykorzystuje się binarne diagramy decyzyjne BDD.

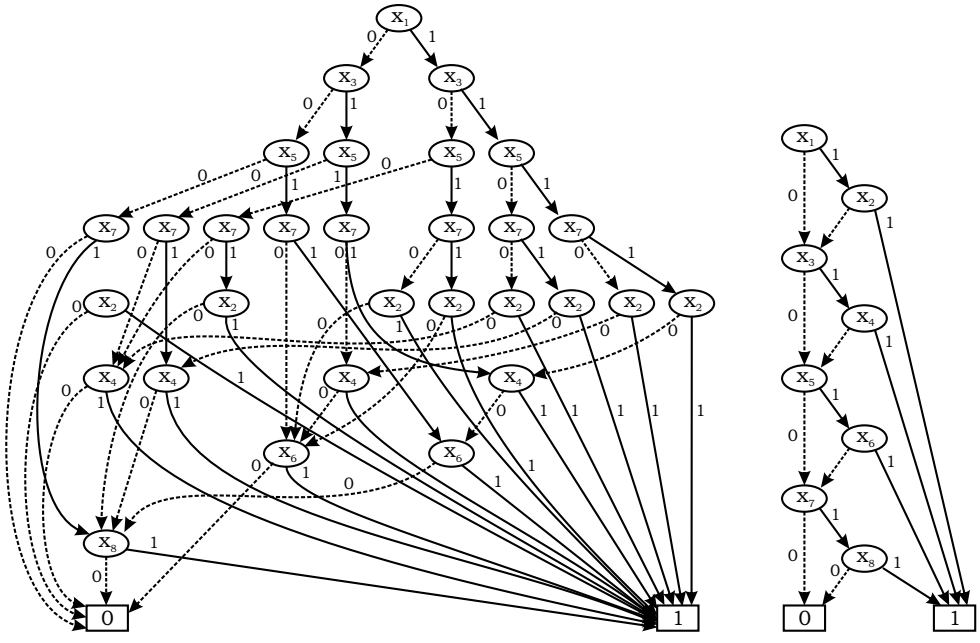
10.3.1. Binarne diagramy decyzyjne

BDD są strukturami reprezentującymi funkcje logiczne. Na bardziej abstrakcyjnym poziomie można o nich myśleć jak o skompresowanej reprezentacji zbiorów, na której można bezpośrednio wnioskować bez konieczności dekompresji. Przy pomocy BDD funkcja logiczna jest przedstawiana jako ukorzeniony, skierowany graf acykliczny składający się z węzłów decyzyjnych oraz dwóch węzłów terminalnych o wartościach 0 i 1. Każdy węzeł decyzyjny N jest oznaczony funkcją logiczną V_N i ma dwa węzły potomne, z których jeden odpowiada przyporządkowaniu V_N wartości 0, a drugi 1.

BDD jest uporządkowany, gdy różne zmienne pojawiają się w tej samej kolejności na wszystkich ścieżkach rozpoczynających się w korzeniu. BDD jest zredukowany jeśli połączy się wszystkie izomorficzne podgrafy oraz wyeliminuje wszystkie węzły, których potomkowie są izomorficzni. Powszechnie przez BDD określa się zredukowany, uporządkowany binarny diagram decyzyjny ROBDD.

Rozmiar BDD zależy nie tylko od funkcji, którą modeluje, ale również od kolejności uszeregowania zmiennych. Przykład wpływu różnego uszeregowania na rozmiar grafu zobrazowano na rysunku 10.4. Przedstawia on wynik modelowania funkcji $f(x_1, ..., x_8) = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8$ [5].

Weryfikacja modelu w oparciu o BDD przebiega w sposób analogiczny do opisanego powyżej. Negację specyfikacji LTL ($\neg\varphi$) oraz model M należy przedstawić w postaci BDD, a następnie policzyć ich koniunkcję ($BDD_M \wedge BDD_{\neg\varphi}$). Poprzez przeszukiwanie grafu w głąb poszukiwana jest ścieżka prowadząca do węzła terminalnego o wartości 1. W wypadku nieznalezienia takiej ścieżki, model spełnia specyfikację.



Rys. 10.4: Dwa przykłady BDD różniące się uszeregowaniem zmiennych: niewłaściwe uszeregowanie (z lewej), właściwe uszeregowanie (z prawej).

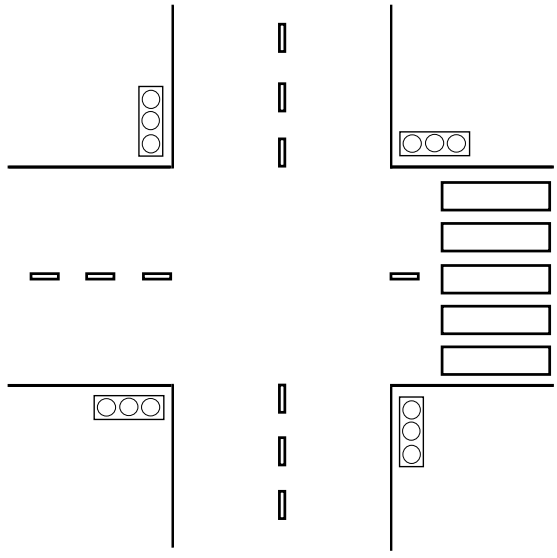
10.4. Przykład weryfikacji modelu

Ze względu na mnogość metod modelowania i weryfikacji systemów przebieg tych procesów postanowiono pokazać na powszechnym oraz zrozumiałym przykładzie systemu sterowania sygnalizacją świetlną na skrzyżowaniu drogowym. Sam problem wydaje się dość prosty, lecz jego waga może być ogromna. W rzeczywistych wdrożeniach takich systemów każda pomyłka może skutkować utratą życia uczestników ruchu.

10.4.1. Opis problemu

Przed rozpoczęciem modelowania należy dokładnie zapoznać się z problemem, poczynając od ustalenia warunków brzegowych. W pierwszym kroku zdecydowano więc, że modelowane będzie skrzyżowanie będące przecięciem dwóch jednopasmowych dróg z czterema sygnalizatorami i jednym przejściem dla pieszych. Schemat takiej krzyżówki pokazano na rysunku 10.5.

Następnie należy określiono tryb przepływu samochodów przez skrzyżowanie. Możliwe jest puszczenie ruchu tylko z jednej odnogi lub równoległe włączenie dwóch zielonych świateł i zezwolenie na jednoczesny przejazd aut z dwóch stron. Przyjęto założenie, że kierowcy mają możliwość jazdy w każdą z trzech stron. W tym wypadku udostępnienie ruchu tylko z jednej odnogi wydaje się być najefektywniejszym rozwiązaniem.

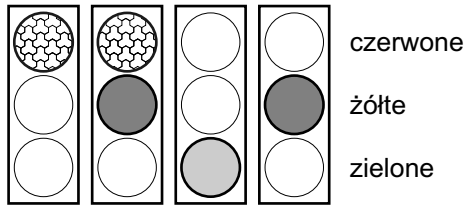


Rys. 10.5: Schemat skrzyżowania.

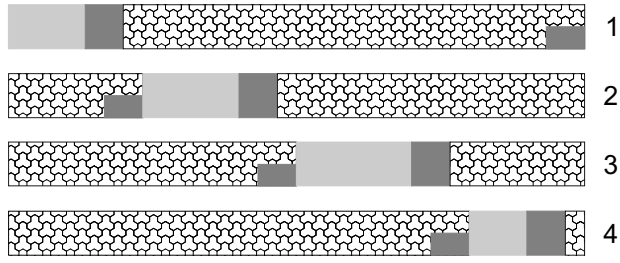
Aby zoptymalizować działanie skrzyżowania, a więc zmaksymalizować przepływ aut, należy dopasować czasy aktywności świateł na poszczególnych sygnalizatorach bazując na rzeczywistych danych. W zależności od konieczności można ustawić tryby zależne od pory dnia lub pomyśleć o czujnikach wykrywających zbliżanie się pojazdu i włączających zielone światło w przypadku stwierdzenia bezkolizyjności tej decyzji. Oprócz dążenia do efektywnego rozładowania ruchu należy zwracać również uwagę na kwestie bezpieczeństwa. Problem modelowania jest więc szerszy niż mogłoby się to początkowo wydawać. Przykładem dodatkowych trudności jest tzw. czynnik ludzki: kierowcy często przejeżdżają na żółtym i nawet czerwonym świetle. W tym celu przy projektowaniu należy uwzględnić opóźnienie przełączenie światła czerwonego na jednej z odnóg w stosunku do wyłączenia światła zielonego na drugiej. Pozwoli to uniknąć stłuczek, które nie zostałyby uwzględnione w idealnym systemie.

Schemat kolejności zmiany świateł w pojedynczym sygnalizatorze przebiega w sposób pokazany na rysunku 10.6. Wykorzystanie semafora czterostanowego pozwala na określenie następnego stanu, w odróżnieniu od sygnalizatora trójstanowego, który po świetle żółtym przełącza się zarówno na zielone i czerwone. Na rysunku 10.7 pokazano wzajemne ustawienia wszystkich czterech semaforów. Jak widać, uwzględniono wcześniej wspomniane przesunięcie czasowe między przełączaniem się świateł na różnych odnogach oraz dopilnowano, by w żadnym momencie nie było włączone więcej niż jedno zielone światło.

Dodatkowo poza sterowaniem ruchem samochodowym skrzyżowania, światła muszą umożliwić pieszym przejście przez jezdnię. Pasy i światła dla pieszych mogą pojawiać się na jednej lub wszystkich z dróg. Zielone światło dla niezmotoryzowanych może włączać się, gdy ruch jest zatrzymany na całym skrzyżowaniu lub jedynie na odnodze, na której znajdują się pasy. Załącza się ono na skutek żą-



Rys. 10.6: Schemat kolejności zmiany świateł pojedynczego semaforma.



Rys. 10.7: Sekwencja zmiany świateł czterech semaforów.

dania wygenerowanego przez wciśnięcie guzika lub w określonym na stałe momencie w każdym cyklu działania krzyżówki. W modelowanym przypadku występuje jedno przejście dla pieszych, włączające się na żądanie pieszego jedynie gdy ruch na stowarzyszonej odnodze jest zatrzymany.

10.4.2. Modelowanie skrzyżowania

Skrzyżowanie przedstawiono jako skończony automat stanów. Do stworzenia modelu skrzyżowania wykorzystano program NuSMV (`http://nusmv.fbk.eu/NuSMV/`). Jest to narzędzie pozwalające na formalną weryfikację automatów stanów skończonych. W procesie tym sprawdzana jest ich zgodność z napisaną w języku logiki temporalnej specyfikacją. Specyfikacja może być napisana w LTL lub CTL, jednak w wypadku LTL w trakcie sprawdzania zostaje ona automatycznie przekonwertowana na CTL. Język programistyczny wykorzystywany w programie stworzony został z myślą o budowaniu FSM. Umożliwia tworzenie hierarchicznych modułów i procesów opisujących systemy synchroniczne lub asynchroniczne. W procesie weryfikacji program bazuje na binarnych diagramach decyzyjnych, a więc wykorzystywane są skończony typy danych jak: zmienne logiczne, typy wyliczeniowe itp.

Automaty stworzone w NuMSV mają postać modelu Kripkego. W trakcie weryfikacji program upraszcza problem przedstawiając całe grupy stanów w postaci symbolicznej. Następnie na powstałych zbiorach przeprowadza przegląd zupełny. W przypadku natrafienia na niespełniający specyfikacji przypadek, podaje go na wyjściu.

Implementacja

Stworzony model składa się z trzech modułów. Jego główna część zawiera się w module opisującym działanie pojedynczego sygnalizatora. Na jego podstawie przybliżony zostanie schemat projektowania automatu w NuSMV [7]. Pozostałe moduły to moduł opisujący działanie świateł dla pieszych oraz moduł skrzyżowania zawierający więcej obiektów typu sygnalizator oraz światła dla pieszych.

Poniżej we fragmentach pokazany został kod modułu sygnalizatora. Dużymi literami zaznaczono sekcje składające się na cały komponent. Na początku zadeklarowana została nazwa modułu wraz ze zmiennymi, od których jest on (moduł) zależny. W tym przypadku jest to zmienna `position` informująca, na której z odnóg skrzyżowania znajduje się semafor.

```
MODULE semaphore(position)
```

W sekcji `VAR` pojawiają się zmienne stanowe powiązane z modułem. Jak widać sygnalizator może przyjmować jedną z czterech wartości skojarzonych z kolorami świateł. Zakres czasu odpowiada maksymalnej liczbie jednostek czasowych, przez które może świecić się dany kolor. W każdej zmianie stanu modelu zmienia się wartość czasu. Jedynie w niektórych chwilach czasowych modyfikowane będą zmienne stowarzyszone z kolorami.

```
VAR
  state : {red, yellow, green, red_yellow};
  time : 0..100;
```

W segmencie `DEFINE` definiowane są wyrażenia, które po napotkaniu w kodzie są zastępowane formułami pojawiającymi się po prawej ich stronie. Definicje późniejsze mogą być zależne od poprzednich, lecz nie na odwrót. Projektant może sam ustalić czasy: świecenia świateł zielonych, przejściowych oraz czas opóźnienia wyłączenia świateł czerwonych. Czas załączenia świateł czerwonych uzależniony jest od czasu trwania całego cyklu oraz świecenia sygnalizacji zielonej.

```
DEFINE
-- czasy zalaczenia swiatel zielonych
  t_g_1 := 6;
  t_g_2 := 4;
  t_g_3 := 5;
  t_g_4 := 4;
-- czasy stanow przejsciowych
  t_y := 2;
  t_ry := t_y;
-- czas opoznienia wylaczenia swiatel czerwonych
  t_delay := 1;
-- czas trwania cyklu
  t_cycle := t_g_1 + t_g_2 + t_g_3 + t_g_4 + 4*t_y + 4*t_delay;
-- czas zielonych ze wzgledu na pozycje
  t_g :=
    case
      position = 1 : t_g_1;
      position = 2 : t_g_2;
      position = 3 : t_g_3;
```

```
      position = 4 : t_g_4;
    esac;
-- czas czerwony ze wzgledu na czas zielony
  t_r := t_cycle - t_g - 2 * t_y;
```

W sekcji `ASSIGN` zakodowane są faktyczne tranzycje między kolejnymi stanami. Wyrażenie `init` pozwala na zdefiniowanie od jakiego stanu ma rozpocząć się działanie modułu. I tak, w zależności od pozycji, semafory będą rozpoczynały pracę święcąc na czerwono lub zielono i będą miały różne wewnętrzne liczniki czasu. Każdy sygnalizator ma inny czas własny, aby umożliwić działanie skrzyżowania zgodne z pokazanym na rysunku 10.7 diagramem. Gdy załącza się dane światło jednocześnie ustawiany jest wewnętrzny czas danego modułu.

```
ASSIGN
  init(state) :=
    case
      position = 1 : green;
      position = 2 : red;
      position = 3 : red;
      position = 4 : red;
    esac;
-- czas startowy - czas trwania pierwszej zmiany swiatel
  init(time) :=
    case
      position = 1 : t_g_1;
      position = 2 : t_g_1 + t_delay;
      position = 3 : t_g_1 + t_y + t_g_2 + 2*t_delay;
      position = 4 : t_g_1 + t_y + t_g_2 + t_y + t_g_3
                     + 3*t_delay;
    esac;
```

Również w sekcji `ASSIGN` znajduje się schemat przebiegu kolejnych tranzycji systemu, zapisany za pomocą wyrażenia `next`. W tej części zaobserwować można funkcjonowanie czasu w module. Co zmianę stanu wartość zmiennej czasowej maleje o jedną jednostkę. Gdy jej wartość osiągnie 1 następuje zmiana świateł oraz ustawienie czasu świecenia dla następnego koloru.

```
  next(state) :=
    case
-- jesli czas=1 nastepuje zmiana stanu
      (state = yellow & time = 1) : red;
      (state = red & time = 1) : red_yellow;
      (state = red_yellow & time = 1) : green;
      (state = green & time = 1) : yellow;
-- jesli czas!=1 kolor zostaje bez zmian
      (state = red & !(time = 1)) : red;
      (state = yellow & !(time = 1)) : yellow;
      (state = green & !(time = 1)) : green;
      (state = red_yellow & !(time = 1)) : red_yellow;
      TRUE : state;
    esac;
  next(time) :=
```

```
    case
      (time > 1) : time - 1;
-- czasy ustawiane sa dla nastepnego koloru
      (time = 1 & state = red) : t_ry;
      (time = 1 & state = red_yellow) : t_g;
      (time = 1 & state = green) : t_y;
      (time = 1 & state = yellow) : t_r;
      TRUE: time;
    esac;
```

W podobny sposób zaimplementowano działanie świateł dla pieszych oraz moduł całego skrzyżowania.

Specyfikacja

Specyfikację w programie przedstawia się wykorzystując klasyczne operatory logiczne znane z innych języków programowania oraz stosując dodatkowe operatory temporalne:

- G – $\Box\phi$ - zawsze zachodzi ϕ,
- F – $\Diamond\phi$ - w bieżącym stanie lub w przyszłości zajdzie ϕ,
- X – $\bigcirc\phi$ - w następnej chwili zajdzie ϕ.

Formuły specyfikacji są dopisywane na końcu pliku zawierającego opis działania FSM.

Poprawność funkcjonowania skrzyżowania można sprawdzić w pokazany poniżej sposób.

- Zielone światło dla pieszych nigdy nie będzie włączone, gdy światło na stowarzyszonej z nim odnodze nie będzie czerwone.

```
LTLSPEC G !(pd_lt_sm2.state = on & !(sm2.state = red));
```

- Zielone światło nigdy nie będzie jednocześnie świecić się na więcej niż jednym sygnalizatorze.

```
LTLSPEC G !(sm1.state = green & (sm2.state = green |
            sm3.state = green | sm4.state = green));
LTLSPEC G !(sm2.state = green & (sm1.state = green |
            sm3.state = green | sm4.state = green));
LTLSPEC G !(sm3.state = green & (sm2.state = green |
            sm1.state = green | sm4.state = green));
LTLSPEC G !(sm4.state = green & (sm2.state = green |
            sm3.state = green | sm1.state = green));
```

- Kolejność zmiany świateł w sygnalizatorze będzie odpowiadała założeniom. Równocześnie będzie zapewnione, że do każdej z pożądanych tranzycji dojdzie nieskończenie wiele razy.
  Globalnie prawdą jest, że pojawi się taki moment, w którym kolor bieżący to czerwony, a kolor w stanie następnym to czerwono-żółty.

```
LTLSPEC G F (sm1.state = red &
            (X sm1.state = red_yellow));
```

Globalnie prawdą jest, że nigdy nie pojawi się taki moment w którym kolor bieżący to czerwony, a kolor w stanie następnym to zielony lub żółty.

```
LTLSPEC G !F (sm1.state = red &
              (X sm1.state = green));
LTLSPEC G !F (sm1.state = red &
              (X sm1.state = yellow));
```

Analogiczne sformułowania napisano dla pozostałych tranzycji.

- Zielone światło dla pieszych nigdy nie będzie włączone gdy światło na stowarzyszonej z nim odnodze nie będzie czerwone.

```
LTLSPEC G !(pd_lt_sm2.state = on & !(sm2.state = red));
```

Testowanie

Napisany program można uruchomić w trybie dynamicznym i interaktywnym. W trybie dynamicznym model sprawdzany jest automatycznie. W przypadku gdy system spełnia specyfikację, wyświetlane są komunikaty o poprawności każdej z formuł logicznych. Poniżej przedstawiono dwa przykładowe komunikaty.

```
-- specification  G !(sm4.state = green & ((sm2.state =
green | sm3.state = green) | sm1.state = green)) is true
-- specification  G ( F (sm1.state = red &
       X sm1.state = red_yellow))  is true
```

W przypadku, gdy specyfikacja nie została poprawnie zamodelowana, na wyjściu programu pojawia się komunikat o niespełnieniu formuły wraz z kontrprzypadkiem przedstawionym jako ciąg kolejnych stanów. W poniższym przykładzie pominięto, również wyświetlone, stany oraz zmienne, których naruszenie specyfikacji nie dotyczyło. Otrzymany komunikat jest efektem zwiększenia czasu aktywności światła zielonego na jednym z sygnalizatorów w niewłaściwym miejscu. Skutkowało to tym, że w jednym stanie światło zielone zapalone było na dwóch sygnalizatorach.

```
-- specification  G !(sm3.state = green & ((sm2.state =
green | sm1.state = green) | sm4.state = green)) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
> State: 1.25 <-
  sm3.state = green
  sm4.state = green
```

Program można uruchomić też w trybie interaktywnym umożliwiającym na symulacje modelu z zadanymi przez użytkownika parametrami.

10.5. Wnioski

Na przykładzie zaprezentowanego systemu sygnalizacji świetlnej widać, że logika temporalna nadaje się do weryfikacji modelów. Stworzona za jej pomocą

specyfikacja jest zwięzła i zrozumiała. Opis specyfikacji systemu różni się od jego reprezentacji za pomocą automatu stanów skończonych. Opis taki nakłada na model odgórne, ogólne wymagania co do jego funkcjonowania. Nie wymaga planowania każdej z tranzycji, a jedynie przedstawia konieczne dla poprawnego działania warunki.

Wykorzystany program NuSMV okazał się wystarczającym narzędziem do rozwiązania problemu stworzenia i weryfikacji modelu. Używany w nim język, dla osoby mającej kontakt z innymi językami programistycznymi, wydaje się intuicyjny i prosty w obsłudze. Sam proces weryfikacji również nie jest skomplikowany. W trybie dynamicznym do weryfikacji modelu wystarczy wpisanie jednej komendy. W trybie interaktywnym poza weryfikacją można przeprowadzać również symulacje ze sprecyzowanymi przez użytkownika parametrami. Wadą programu jest konieczność zawarcia całego kodu automatu i specyfikacji w jednym pliku źródłowym. Możliwość zawarcia pojedynczych modułów oraz specyfikacji w osobnych plikach uczyniłaby pracę z NuSMV bardziej przejrzystą. Na koniec warto wspomnieć, że program wydany jest na licencji publicznej GNU Lesser General Public License, co zachęca do darmowego wykorzystywania go do celów niekomercyjnych.

Literatura

[1] S. Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[2] E.M. Jr. Clarke, O. Grumberg, D.A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[3] J.R. Buechi. On a Decision Method in Restricted Second-Order Arithmetic. *International Congress on Logic, Methodology, and Philosophy of Science*, strony 1–11. Stanford University Press, 1962.

[4] M. Huth, M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004.

[5] K.Y. Rozier. Linear symbolic model cheking. *Computer Science Review*, 2010.

[6] T. Wahl. Temporal logic model checking. *Oxford University*, 2009.

[7] R. Cavada, A. Cimatti, G. Keighren, E. Olivetti, M. Pistore, M. Rover. NuSMV 2.5 Tutorial. `http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf`.

ROZDZIAŁ 11

WIZUALIZACJA ZJAWISK TEMPORALNYCH

D. Rolla, R. Tomaszewski

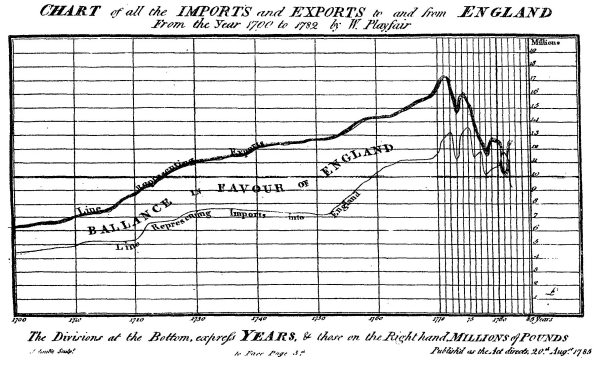
Większość zjawisk zachodzących w otaczającym nas świecie można określić temporalnymi – zjawiska te rozpoczynają się w pewnej chwili, trwają przez jakiś czas i kończą się. Podczas przetwarzania opisów tych zjawisk oraz ich wizualizacji zależność od czasu często rodzi problemy. W niniejszym rozdziale podjęto próby przeanalizowania tych problemów oraz przedstawienia propozycji metod wizualizacji wyników analiz temporalnych.

11.1. Wprowadzenie

Wizualizacja danych to metoda ułatwiająca ich analizę oraz wyciąganie poprawnych wniosków. Dzięki wizualizacji danych szybciej i łatwej można zrozumieć naturę badanych zjawisk oraz określać zależności i cechy dla nich charakterystyczne. Okazuje się, że wykonanie wysokiej jakości wizualizacji jest problemem nietrywialnym i wymaga dogłębnego przestudiowania tematu [1, 2].

Historia zbierania i prezentacji danych liczbowych w formie tabel zaczęła się w okolicach drugiego wieku naszej ery. Jednakże graficzne sposoby przedstawiania danych rozwinęły się dopiero w siedemnastym wieku za sprawą Kartezjusza (fr. *René Descartes*), francuskiego filozofa i matematyka. Zaproponował on ideę wyrażania zależności między liczbami za pomocą układu współrzędnych. W osiemnastym wieku William Playfair jako pierwszy przedstawił zmianę pewnych wartości w czasie za pomocą linii ciągłej umieszczonej w kartezjańskim układzie współrzędnych. Dzięki niemu zawdzięczamy także powstanie wykresu słupkowego i kołowego. Kolejnym przełomem w wizualizacji danych była książka Jacques'a Bertina zatytułowana *Semiologia grafiki*. Publikacja ta dała podwaliny pod dalszy rozwój metod wizualizacji, ponieważ Bertin odkrył, że percepcja wzrokowa człowieka działa zgodnie z pewnymi zasadami, które mogą być wykorzystywane podczas prezentacji danych.

W roku 1977 profesor Uniwersytetu Princeton John Turkey zaproponował nowe statystyczne podejście EDA (ang. *exploratory data analysis*) polegające na analizowaniu cech charakterystycznych prezentowanych danych (np. badanie rozkładów zmiennych, przeglądanie macierzy korelacji). W 1983 roku Edward



Rys. 11.1: Wykres przedstawiający angielski import i eksport w latach 1700-1782 [3].

Tufte opublikował książkę, w której zauważał, że pomimo dostępnych efektywnych sposobów prezentowania danych ludzie wykorzystują je bardzo nieefektywnie. W 1999 panowie Stuart Card, Jock Mackinlay i Ben Shneiderma podsumowali w książce *Readings in Information Visualization: Using Vision to Think* wszystkie dotychczasowe dokonania w dziedzinie wizualizacji danych. Badania dotyczące wizualizacji danych prowadzone w XXI wieku zorientowane ogniskują się na ludzkiej percepcji. Ich wyniki dobrze przedstawił Colin Ware w publikacjach *Information Visualization: Perception for Design* i *Visual Thinking for Design*.

11.2. Metody wizualizacji danych

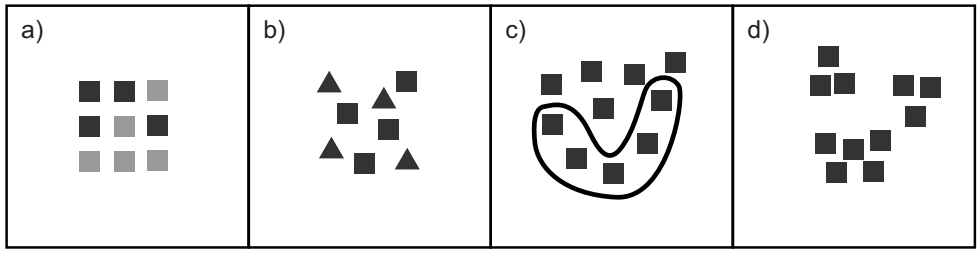
11.2.1. Podejście tradycyjne

Dane można wizualizować na wiele sposobów. Jednak nie wszystkie są równie efektywne. Aby osiągnąć maksymalny przekaz należy zagłębić się w sposób działania ludzkiego umysłu. Dzięki zrozumieniu pewnych mechanizmów tworzenie wizualizacji może stać się bardziej przejrzyste i intuicyjne.

Za odbiór wrażeń wzrokowych odpowiada ośrodek wzroku znajdujący się w obrębie kory płata potylicznego. W skład kory wzrokowej wchodzi również asocjacyjna kora wzrokowa, która bodźce wzrokowe analizuje i kojarzy z sygnałami pochodzącymi z innych ośrodków mózgu. Proces widzenia jest procesem niezwykle szybkim i efektywnym, niewymagającym wielkiego wysiłku. Natomiast procesy myślenia odbywają się w korze mózgowej w płacie czołowym. Jest on odpowiedzialny m.in. za planowanie, rozpoznawanie, pamięć, ocenę emocji i sytuacji, a także podejmowanie decyzji. W porównaniu do ośrodka wzroku układ ten działa stosunkowo wolno i mało efektywnie.

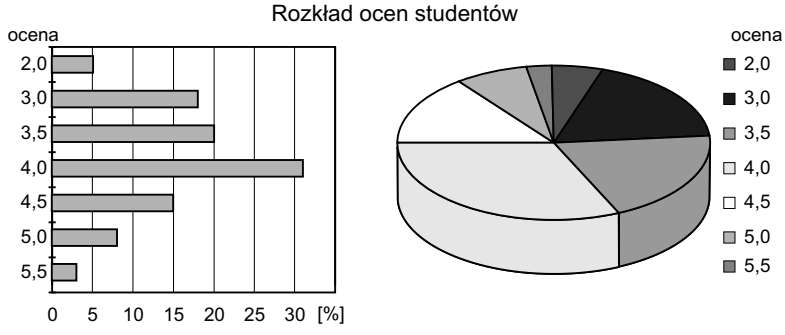
Najbardziej popularne metody wizualizacji danych w całości odnoszą się do pracy płata czołowego. Stephen Few [1] zaproponował, aby tak wizualizować dane, żeby część procesów ich analizy wykonywała się już w ośrodku wzroku.

Przykładowo, obiekty o tych samych kształtach lub kolorach będą instynktownie klasyfikowane przez ludzi do tej samej grupy, gdy zostaną w pewien sposób zakreślone lub znajdą się blisko siebie (rys. 11.2). Takie przedstawienie powiązań pomiędzy obiektami daje dużo lepsze efekty niż na przykład ich prezentacja w formie tabelarycznej.



Rys. 11.2: Grupowanie obiektów a) grupy takich samych kolorów, b) grupy takich samych kształtów, c) grupa zakreślona krzywą, d) grupy obiektów będących blisko siebie.

Kolejne zjawiska przeanalizujmy na postawie dwóch wykresów przedstawionych na rysunku 11.3. Na obu wykresach przedstawiono dokładnie te same dane. Wydawać by się mogło, że oba dobrze wizualizują zależności między danymi. Zastanówmy się jednak, jak dokładnie odwzorowane są proporcje. Na wykresie kołowym wyrażone są one za pomocą powierzchni (kątów) poszczególnych kawałków. Niestety w przeciwieństwie do linii prostych, ludzki umysł ma ograniczone możliwości porównywania kątów. Potrafi bardzo sprawnie porównywać długości prostych obiektów. Natomiast w przypadku długości i kątów krzywych, a także odległości między nimi, zaczyna zawodzić. Dlatego porównywanie ze sobą poszczególnych danych na wykresie kołowym jest kłopotliwe i może powodować wyciąganie błędnych wniosków. Efekt ten pogarsza jeszcze fakt, że wykres przedstawiony jest w wersji trójwymiarowej, co wprowadza dodatkowe zniekształcenia. Ostatecznie ciężko oszacować prawdziwe wartości danych. Dodatkowo poszczególne kawałki wykresu nie są opisane, legenda znajduje się obok. Wymaga to cią-



Rys. 11.3: Przykłady wykresów: słupkowego i kołowego.

głego wyszukiwania opisu danych na podstawie koloru, co może uniemożliwić analizę danych osobom cierpiącym na zaburzenia rozpoznawania barw.

Zupełnie inaczej sytuacja wygląda w przypadku wykresu słupkowego. Ponieważ wszystkie słupki wyrównano do tej samej krawędzi i ustawiono równolegle, bardzo łatwo można je porównać. Dzięki temu, że wszystkie słupki są tego samego koloru, można całkowicie skupić się na ocenie ich długości. Ponadto każdy z nich jest bezpośrednio opisany odpowiednią etykietą. Nie musimy zatem wyszukiwać nazw poszczególnych danych.

Dzięki zastosowaniu tych kilku prostych rozwiązań można znaczną część analizy danych przekierować do ośrodka wzroku, zwalniając ośrodek myślenia na potrzeby głębszej analizy i jednocześnie przyspieszając cały proces. Należy jednak pamiętać, iż ludzki umysł jest w stanie jednorazowo przeanalizować i przyswoić ograniczoną ilość informacji. Przy wizualizacji zbyt wielu danych z pewnością część z nich zostanie przez odbiorca pominięta.

Podczas tworzenia wizualizacji należy pamiętać jeszcze o jednym. W wyniku wad wzroku ludzie mogą widzieć nieostro. Dlatego wizualizacje ze zbyt drobnymi szczegółami mogą okazać się niemożliwe do zaakceptowania. Podobnie postępować należy przy wadach polegających na zaburzeniu rozpoznawania barw (tritanopia, deuteranopia, protanopia, monochromatyzm, daltonizm). Próby przekazania informacji w oparciu jedynie o kolory mogą wtedy okazać się nieskuteczne.

11.2.2. Podejście temporalne

Żeby zrozumieć ideę danych temporalnych zacznijmy od analizy klasycznej bazy danych. Przechowuje ona pewne informacje, które są aktualne w danej chwili. W pewnym sensie dane nie zależą zatem w żaden sposób od czasu. Przykładem takiej bazy danych może być baza pensji pracowników pewnej firmy o prostym schemacie:

imię i nazwisko	pensja
Jan Nowak	1500 zł
Małgorzata Malinowska	2800 zł

Taka baza zawiera pewną porcję informacji o każdym pracowniku (w tym przypadku aktualną wysokość pensji). Nie wiemy jednak, jak wyglądała wysokość pensji w przeszłości ani jak będzie zmieniać się w przyszłości. Aby dało się takie historie zmian wartości monitorować oraz dokonywać predykcji, należałoby w schemacie bazy danych uwzględnić czas:

imię i nazwisko	pensja	okres
Jan Nowak	1300 zł	2012 - 2013
Jan Nowak	1500 zł	2013 - *INF*
Małgorzata Malinowska	2800 zł	2008 - *INF*
Krzysztof Niziński	1800 zł	2010 - 2011

Dodanie do tabeli dodatkowej kolumny informującej o okresie otrzymywania danej pensji pozwala zapisać pełną historię wynagrodzeń pracowników. Widać,

że Jan Nowak w latach 2012-2013 zarabiał 1300 zł, a w roku 2013 dostał 200 zł podwyżki. Zapis *INF* oznacza, że wraz z upływem czasu dane są wciąż aktualne. Widać również, że w latach 2010-2011 w firmie pracował Krzysztof Niziński i otrzymywał pensję w wysokości 1800 zł. W tradycyjnej nietemporalnej bazie danych informacja o wypłacie pana Nizińskiego zostałaby usunięta w momencie zwolnienia z pracy. Informację o okresie ważności danych w literaturze określa się mianem ***valid time***.

Baza danych ewoluuje w czasie. Nowe dane są do niej dopisywane, natomiast niektóre z aktualnych danych tracą ważność i mogą być usunięte. Temporalna baza danych daje możliwość śledzenia i zapamiętywania tych zmian. Wymaga to wprowadzenia dodatkowej informacji o czasie wprowadzania zmian. Informacja taka w literaturze zwana jest ***transaction time***. Najprościej mówiąc jest to czas wprowadzenia informacji o zaistniałym fakcie (zdarzeniu) do bazy danych. Przykładem bazy danych z czasem wprowadzenia jest lista pensji pracowników wraz z momentem wprowadzenia zmian:

imię i nazwisko	pensja	transaction time
Jan Nowak	1300 zł	2012
Jan Nowak	1500 zł	2013
Małgorzata Malinowska	2800 zł	2008
Krzysztof Niziński	1800 zł	2010
Krzysztof Niziński	0 zł	2011

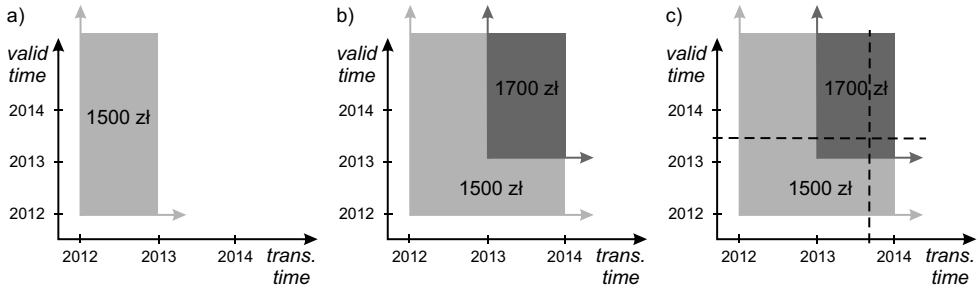
Złożeniem temporalnej bazy danych zależnej od *transaction time* oraz od *valid time* jest baza bitemporalna (*bitemporal database*). Przechowuje ona zarówno informację o okresie ważności danych, jak i momencie ich wprowadzenia do bazy. Obrazując to na konkretnym przykładzie: Jan Kowalski przyjmuje się do pracy w roku 2012 i otrzymuje pensję równą 1500 zł. Zatem baza danych wygląda następująco:

imię i nazwisko	pensja	*valid time*	*transaction time*
Jan Kowalski	1500 zł	2012 - *INF*	2012

Moment wprowadzenia informacji do bazy (*transaction time*) to 2012 rok, natomiast planowany okres otrzymywania wynagrodzenia (*valid time*) zaczyna się w 2012 roku. Nie wiemy jednak, kiedy pan Kowalski zmieni warunki zatrudnienia, dlatego zakładamy, że górna granica *valid time* wynosi *INF*. W 2013 roku pracownik otrzymuje podwyżkę. Stąd w bazie danych pojawi się nowy wpis:

imię i nazwisko	pensja	*valid time*	*transaction time*
Jan Kowalski	1500 zł	2012 - 2013	2012
Jan Kowalski	1700 zł	2012 - *INF*	2013

Zauważmy jednak, że wraz z umieszczeniem nowego faktu w bazie poprzednia informacja straciła ważność i należy zmienić jej okres ważności z *INF* na rok 2013. Spróbujmy umieścić te informacje na wykresach (rys. 11.4).



Rys. 11.4: Bitemporalne wykresy wartości płacy Jana Kowalskiego a) sytuacja znana w 2013 roku, b) sytuacja znana w 2014 roku, c) wypłata Jana Nowaka w VI 2013 na podstawie danych z X 2013.

Takie przechowywanie i wizualizacja danych pozwala na łatwe analizowanie aktualnej sytuacji oraz historii zmian. Aby sprawdzić zarobki Jana Kowalskiego znane w październiku 2013 roku wystarczy przeciąć wykres poziomą linią w odpowiednim miejscu na osi *transaction time* (rys. 11.4 c). Przemieszczając się wzdłuż przerywanej linii można zaobserwować, że w okresie 2012-2013 zarabiał on 1500 zł, natomiast teraz otrzymuje kwotę 1700 zł.

Z wykresu można również odczytać odpowiedź na pytanie: Ile według bazy zarabiał pan Kowalski w czerwcu 2013 roku? Otóż według założeń w latach 2012-2013 miał on zarabiać 1500 zł. Natomiast od roku 2013 stało się wiadome, że jego wynagrodzenie wynosi 1700 zł. Odpowiedź na bardziej precyzyjne pytanie: Ile zarabiał Jan Kowalski w roku czerwcu 2013 na podstawie bazy danych z października 2013 roku znajduje się na przecięciu obu widocznych linii przerywanych.

11.3. Metody zapisu danych temporalnych w plikach XML

Pliki XML są bardzo często wykorzystywane do przechowywania wszelkiego rodzaju danych, w tym danych temporalnych. Do zapisu danych związanych z czasem można wykorzystywać typy zdefiniowane w standardzie XML Schema:

- **`xs:date`** - typ opisujący dzienną datę w formacie `"YYYY-MM-DD"` (rok-miesiąc-dzień). Wszystkie te elementy są wymagane. Przykładowa deklaracja daty może wyglądać następująco:

```
<xs:element name="startDate" type="xs:date"/>
<startDate>2014-06-11</startDate>
```

- **`xs:time`** - typ opisujący godzinę w formacie `"hh:mm:ss"` (godzin:minut:sekund). Definiuje się go analogicznie do przypadku `xs:date`

```
<xs:element name="startTime" type="xs:time"/>
<startTime>13:30:00</startTime>
```

- **`xs:dataTime`** - typ ten jest złożeniem obu poprzednich typów. Przyjmuje format: `"YYYY-MM-DDThh:mm:ss"`, gdzie `T` jest separatorem.

```
<startdate>2014-06-11T13:30:00</startdate>>
```

- **`xs:duration`** - typ definiujący okres. Przyjmuje format `"PnYnMnDTnHnMnS"`, gdzie `P` jest stałą wymaganą, `T` separatorem między datą i czasem, natomiast `nY,nM,nD,nH,nM,nS` znaczą kolejno liczbę (`n`) lat, miesięcy, dni, godzin, minut i sekund w okresie. Nie wymaga się, aby wszystkie elementy były podane. Format dopuszcza zarówno deklarację `<period>P7Y</period>` jak również `<period>P2Y1M11DT12H1M9S</period>`.

Zgodnie z propozycją opublikowaną w [4] dane temporalnych można zapisywać w pliku XML z zastosowaniem stempli czasu oznaczającymi ich okres ważności (umieszczanymi w atrybutach węzłów). Poniżej pokazano efekt wstawiania takich stempli dla danych z przytoczonego wcześniej przykładu:

```
<osoba imie="Jan" nazwisko="Kowalski">
    <pensja Od="2012" Do="2013">
        1500
    </pensja>
    <pensja Od="2013" Do="INF">
        1700
    </pensja>
<\osoba>
```

11.4. Opis autorskiego rozwiązania

W analizach funkcjonowania systemów informatycznych często napotyka się zagadnienia związane z raportowaniem stanu jakichś procesów czy też rejestracji i przetwarzania zebranych statystyk. Można do nich zaliczyć monitorowanie ilości stron wydrukowanych przez różnych użytkowników na przestrzeni czasu, z możliwość wyboru jego danego okresu. Aplikacja, którą opisano dalej, miała umożliwiać takie monitorowanie poprzez wczytywanie danych dostarczonych w formacie XML i ich wizualizację na kilku typach wykresów.

11.4.1. Format danych wejściowych

Dane do aplikacji są wprowadzane za pomocą pliku XML o strukturze zdefiniowanej przez następujący schemat dokumentu XML:

```
<xs:complexType name="Type4Task">
       <xs:attribute name="Start" type="xs:dateTime" />
       <xs:attribute name="End" type="xs:dateTime" />
       <xs:attribute name="UserName" type="xs:string" />
       <xs:attribute name="NumberOfPages" type="xs:integer" />
</xs:complexType>

<xs:complexType name="Type4Printer">
        <xs:sequence>
                <xs:element name="Task" type="Type4Task" minOccurs
                    ="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" />
</xs:complexType>
```

```
<xs:complexType name="Type4PrintersData">
        <xs:sequence>
                <xs:element name="Printer" type="Type4Printer"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>

<xs:element name="PrintersData" type="Type4PrintersData">
</xs:element>
```

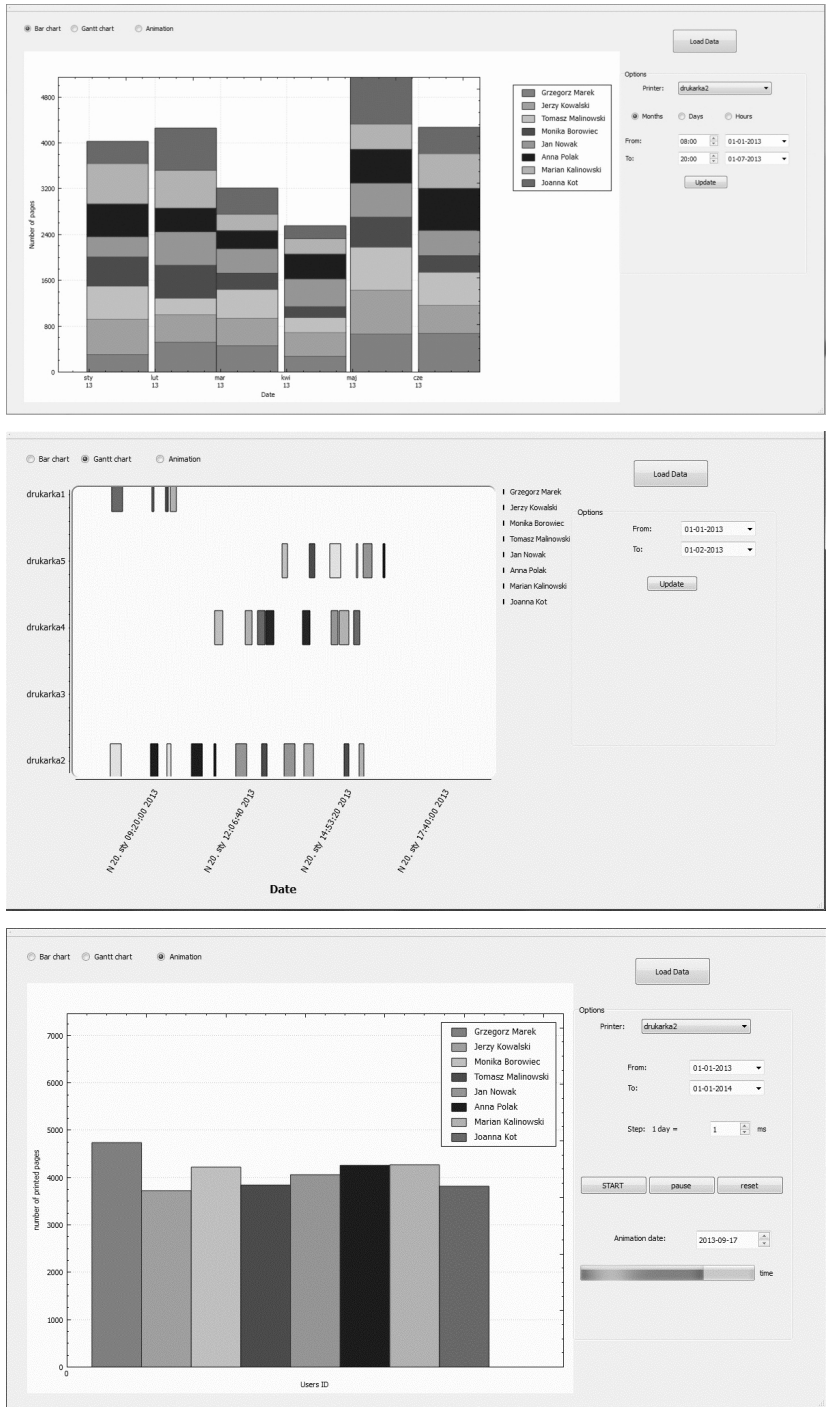
Poniżej opisano znaczenie poszczególnych elementów i atrybutów:

- `PrintersData` – element zawierający nieokreśloną ilość elementów `Printer` zawierających informację o drukarce i jej zadaniach.
- `Printer` – element zawierający nazwę drukarki(atrybut `Name`) oraz nieokreśloną liczbę zadań przechowywanych w elementach typu `Task`.
- `Task` – element zawierający informację o użytkowniku `UserName` który zlecił zadanie, liczbie drukowanych stron `NumberOfPages`, oraz czasach rozpoczęcia `Start` i zakończenia `End` wykonywania zadania przechowywane w formacie `"yyyy-MM-ddThh:mm:ss"`.

11.4.2. Opis interfejsu graficznego

Na rysunku 11.5 przedstawiono przykładowe zrzuty ekranu aplikacji. W prawym górnym rogu każdego przykładu znajduje się przycisk *Load Data* otwierający okno służące do wyboru pliku w formacie XML, z którego zostaną wczytane dane temporalne (na potrzeby testowania i prezentacji projektu dane zostały wygenerowane sztucznie). W lewym górnym rogu znajdują się trzy przyciski pozwalające na wybór sposobu wyświetlania danych, po wyborze typu z prawej strony pojawiają się odpowiednie opcje. Aplikacja pozwala na wizualizację danych temporalnych na 3 różne sposoby:

- **Wykres słupkowy**(przycisk *Bar chart*) – tryb ten pozwala na wizualizację ilości stron drukowanych przez poszczególnych użytkowników o określonych okresach czasu. Menu z prawej strony pozwala na wybór drukarki dla której są prezentowane dane, dokładność(miesiące, dni lub godziny) oraz przedział czasowy z którego dane nas interesują.
- **Wykres gantta**(przycisk *Gantt chart*) – tryb ten pozwala na wizualizację zajętości drukarek przez zadania w określonym czasie. Dodatkowo każde zadanie jest oznaczone kolorem przyporządkowanym do użytkownika a diagram można przybliżać w zależności od potrzeb.
- **Animacja**(przycisk *Animation*) – tryb ten pozwala na wizualizację tego jak zmieniała się ilość drukowanych przez użytkowników stron w określonym okresie czasu. Oprócz tego można ustalić szybkość animacji – tj. ile milisekund animacji będzie trwał dzień pracy drukarki. Poniżej wyświetlana jest aktualna data dla której wyświetlane są dane oraz pasek postępu w stosunku do całego czasu animacji.



Rys. 11.5: Widok interfejsu po wybraniu opcji wykresu słupkowego (u góry), wykresu Gantta (po środku) i animacji (na dole).

11.4.3. Wykorzystane narzędzia programowe

Wizualizacja informacji przy pomocy aplikacji komputerowej powinna bazować na ogólnych zasadach opisanych w rozdziale 11.2. W procesie tworzenia aplikacji bardzo istotnym etapem jest wybór języka programowania i/lub bibliotek spełniających stawiane przez problem wymagania. Wiele języków oraz bibliotek pozwala na tworzenie graficznego interfejsu użytkownika (*Java*, *C#*(zestaw narzędzi *Gtk#*), *C++*(np. biblioteki *wxWidgets*, *GTK+*, *Qt* itd.).

Podczas implementacji autorskiej aplikacji zdecydowano się na użycie biblioteki *Qt* dla języka *C++*, zarówno ze względu na dość dobrą znajomość tego narzędzia przez autorów jak i jej dostępność (licencja GPL) oraz liczne zalety (wieloplatformowość, bardzo dobra dokumentacja (`http://qt-project.org/doc/`), szeroki zakres gotowych klas służących do budowy interfejsu, obsługę procesów, plików, języka XML). Biblioteka ta jest rozwijana od 1991. Obecnie *Qt* jest dostępne na platformy *X11*, *Windows*, *Mac OS X* a także wspiera platformy mobilne takie jak *Android*. *Qt* jest podstawą np. dla unixowego środowiska graficznego *KDE*. Warto zwrócić uwagę, że biblioteka ta pozwala na tworzenie wizualizacji danych zarówno dzięki wbudowanym klasom jak i klasom udostępnionym przez użytkowników na licencji CreativeCommons BY-NC-SA (jak np. biblioteki *QCustomPlot* (`http://www.qcustomplot.com/`) lub *Qwt* (`http://qwt.sourceforge.net/index.html`).

11.4.4. Podsumowanie

Zaimplementowana aplikacja ma możliwość wczytywania danych z pliku XML zgodnego z zaprojektowanym schematem XSD. Podczas jej działania dane są przetwarzane i przechowywane w pamięci operacyjnej. Umożliwia ona dokonywanie wizualizacji liczby wydrukowanych stron (na wykresie słupkowym) lub zajętości drukarek w rozpatrywanych okresach czasu (na diagramie Gantta). Dodatkowo zaimplementowana została możliwość wyświetlenia przyrostów wydruków w formie animowanego wykresu słupkowego. W każdym z typów wizualizacji można wyświetlać dane pochodzące z różnych drukarek oraz ustawiać różne przedziały czasu. Stworzona aplikacja prezentuje przykładowe sposoby przechowywania, przetwarzania oraz wizualizacji zjawisk temporalnych.

Literatura

[1] S. Few. *Data Visualization for Human Perception*. The Interaction Design Foundation, Aarhus, Denmark, 2013.

[2] R.T. Snodgrass. A Case Study of Temporal Data. *Teradata Magazine Online*, Październik 2010.

[3] W. Playfair. *Commercial and Political Atlas*. 1786.

[4] F. Rizzolo, A. A. Vaisman. Temporal XML: Modeling, Indexing, and Query Processing. *The VLDB Journal manuscript*, 17:1179–1212, 2008.

Od redaktora i wydawcy

W niniejszej książce zebrano opracowania studentów II roku studiów magisterskich Wydziału Elektroniki Politechniki Wrocławskiej, kierunku Automatyka i robotyka, specjalności Robotyka, wykonane w semestrze letnim roku akademickiego 2013/2014 podczas realizacji prowadzonego przeze mnie kursu *Komputerowe przetwarzanie wiedzy*. Schemat organizacji treści tych opracowań jest podobny. Rozpoczynają się krótkim przedstawieniem dziedziny w jakiej je osadzono, po czym następuje opis autorskiego rozwiązania. Zakres poruszanych tematów można hasłowo podsumować w następujący sposób:

- gromadzenie wiedzy (klasyfikacja tekstów, audyt w systemach informatycznych),
- semantyczne bazy wiedzy (wykorzystanie reguł, integracja rozproszonych ontologii),
- interakcje człowieka z maszyną (urządzenia wejścia, łączenie świata analogowego z cyfrowym, sterowniki haptyczne),
- rozwiązywanie problemów spełniania ograniczeń (wykorzystanie algorytmów genetycznych),
- języki dziedzinowe (prekompilacja składni LaTeX),
- zagadnienia temporalne (zastosowanie logiki temporalnej, metody wizualizacji).

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

Tomasz Kubik
Wrocław, listopad 2014

ISBN 978-83-930823-6-0
9 788393 082360