Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego

Inżynieria oprogramowania i systemy czasu rzeczywistego: od badań do praktycznych zastosowań

Redakcja naukowa
Lech Madeyski, Piotr Kosiuczenko, Marek Bolanowski

RADA NAUKOWA
POLSKIEGO TOWARZYSTWA
INFORMATYCZNEGO

PTI

Inżynieria oprogramowania i systemy czasu rzeczywistego: od badań do praktycznych zastosowań

Redakcja naukowa

Lech Madeyski, Piotr Kosiuczenko, Marek Bolanowski

Autorzy

Zbigniew Świder, Leszek Trybus – ***ROZDZIAŁ 1****, Maciej Koryl, Paweł Dymora, Mirosław Mazurek –****ROZDZIAŁ 2****, Michał Madera, Tomasz Śliwa –* ***ROZDZIAŁ 3****, Grzegprz Dec, Piotr Woźniak –* ***ROZDZIAŁ 4****, Dominik Strzałka–* ***ROZDZIAŁ 5****, Bartorzsz Trybus, Leszek Trybus –* ***ROZDZIAŁ 6****, Dariusz Rzońca, Andrzej Stec, Bartosz Trybus –* ***ROZDZIAŁ 7****, Paweł Głuchowski –* ***ROZDZIAŁ 8****, Sławomir Samolej, Tomasz Rogalski, Dariusz Rzońca –* ***ROZDZIAŁ 9****.*

Recenzenci

Andrzej Stasiak, Aneta Poniszewska-Maranda, Jacek Kluska, Tomasz Lewowski, Andrzej Stasiak, Lech Madeyski, Gregorz Wójcik, Łukasz Więckowski, Jan Cisek, Marcin Skuba, Lucjan Pelc, Leszek Trybus, Sławomir Samolej, Andrzej Paszkiewicz, Jacek Stój, Marcin Szpyrka.

Redakcja naukowa

Lech Madeyski, Piotr Kosiuczenko, Marek Bolanowski

ISBN: 978-83-946253-4-4

Wydanie: I
Liczba Egzemplarzy: 100. Arkuszy wydawniczych: 8,6. Arkuszy drukarskich: 9,9.
Wydruk: Oficyna Wydawnicza Politechniki Rzeszowskiej
al. Powstańców Warszawy, 12 35-959 Rzeszów

Spis treści

Przedmowa

Inżynieria oprogramowania to bardzo ważna, z praktycznego punktu widzenia, dyscyplina naukowa obejmująca wszystkie aspekty wytwarzania oprogramowania poprzez zastosowanie systematycznego, zdyscyplinowanego, wymiernego podejścia do wytwarzania i utrzymania oprogramowania. W chwili obecnej rynek usług programistycznych jest znaczącą gałęzią gospodarki w Polsce jak i na świecie. Wzrost liczby programistów niestety nie implikuje wzrostu jakości wytwarzanego przez nich oprogramowania. Jest to jedno z wyzwań, które ma wpływ m.in. na pielęgnowalność i wydajność projektowanych aplikacji, czy bezpieczeństwo ich eksploatacji. Wagę tego zagadnienia dostrzegła m.in. Rada ds. Kompetencji Sektora IT (http://radasektorowa.pti.org.pl/), której jednym z zadań jest monitorowanie poziomu kompetencji pracowników z obszaru IT. Podczas jednego z posiedzeń Rady podkreślono wagę wiedzy i kompetencji programistów w zakresie znajomości zasad projektowania systemów informatycznych oraz teoretycznych i praktycznych aspektów inżynierii oprogramowania. Stwierdzono konieczność prowadzenia szeroko zakrojonych prac badawczych w tych obszarach i rozpowszechniania ich wyników w środowisku IT. Niniejsza monografia stanowi odpowiedź na zapotrzebowanie rynku prezentując praktyczne wdrożenia wyników prac badawczych prowadzonych w obszarze inżynierii oprogramowania, w tym także w obszarze systemów czasu rzeczywistego. Rozdziały zawarte w monografii wpisują się ideę łączenia aspektów badawczych z zagadnieniami praktycznego ich wykorzystania. Monografia została podzielona na dwie części.

Pierwsza część monografii ilustruje rezultaty prac w obszarze inżynierii oprogramowania, w szczególności stworzone narzędzia oraz ich praktyczne zastosowania, i składa się z pięciu rozdziałów. W pierwszym rozdziale "Edytor graficzny języka LD dla pakietu CPDev" przedstawiona została najnowsza wersja edytora graficznego dla języka LD normy IEC61131. W rozdziale przedstawiono interfejs użytkownika edytora graficznego języka LD, etapy tworzenia diagramu drabinkowego i kod sterowania w języku LD generowany na podstawie stworzonego schematu. W rozdziale drugim "Implementacja systemu wspierającego zjawisko emergencji w modelowaniu procesów biznesowych" przedstawiono możliwości wykorzystania teorii emergencji w modelowaniu procesów biznesowych. W rozdziale zaprezentowano metody użycia oprogramowania ogólnodostępnego jako elementów składowych spójnego systemu wspierającego realizację procesów w sposób emergentny oraz zaproponowano model komponentowy takiego rozwiązania i podstawowe przypadki użycia, które są realizowane z jego pomocą. Rozdział trzeci "Budowa systemu do obliczeń rozproszonych z wykorzystaniem typowej infrastruktury laboratoryjnej" przedstawia sposób wykorzystania istniejącej infrastruktury obliczeniowej ośrodka akademickiego w celu wykonywania obliczeń rozproszonych. Pokazano, że laboratoria komputerowe wyposażone w znaczną liczbę komputerów stacjonarnych, mogą być wykorzystane do rozwiązywania złożonych obliczeniowo zadań. Przedstawiono system zarządzania zasobami obliczeniowymi oraz omówiono problemy napotkane przy realizacji środowiska rozproszonego. W rozdziale czwartym "Usługa ORM typu REST – model i implementacja" przedstawiono model oraz opis implementacji usługi sieciowej, która udostępnia podstawowe funkcje biblioteki ORM. Usługa ta, na podstawie struktury relacyjnej bazy danych, tworzy abstrakcyjny model wybranych elementów schematu

relacyjnego, a następnie generuje warstwę dostępu do danych w języku Java z użyciem technologii JEE oraz platformy Spring. W rozdziale piątym "Concluder – narzędzie analizy problemu zgodności par" przedstawiono możliwości pracy w programie Concluder, który jest prostym, intuicyjnym, ale bardzo skutecznym narzędziem analizy problemu zgodności macierzy ocen w metodzie porównywania parami.

Druga część monografii dedykowana jest systemom czasu rzeczywistego i została wyodrębniona ze względu na specyfikę i charakterystyczne dla tego obszaru wyzwania. Obejmuje ona projektowanie, analizę oraz zastosowanie systemów czasu rzeczywistego i prezentuje możliwości wykorzystania elementów inżynierii oprogramowania w systemach tej klasy. W rozdziale szóstym "Tryb konfiguracji pewnego rozproszonego systemu sterowania" na przykładzie dwóch prostych projektów napisanych w języku Structured Text normy IEC 61131-3 przedstawiono tryb konfiguracji w środowisku CPDev (Control Program Developer) komputera nadrzędnego w prototypowym rozproszonym systemie sterowania średniej wielkości. W rozdziale siódmym "Wykorzystanie języka SFC pakietu CPDev w procesach sterowania sekwencyjnego" przedstawiono sposób tworzenia programów w języku SFC w pakiecie inżynierskim CPDev dla sterowania sekwencyjnego na przykładzie prostego procesu technologicznego. Opisano także przebieg symulacji programu, tworzenia wizualizacji i uruchamiania wirtualnych sterowników, przy wykorzystaniu programów narzędziowych pakietu CPDev. Rozdział ósmy "Optymalizacja skryptu NuSMV modelującego scenariusz zdarzeń zagrażających bezpieczeństwu na lotnisku" poświęcony jest analizie bezpieczeństwa naziemnego ruchu samolotów. W tym rozdziale przedstawiono wybrane aspekty modelowania czasowo zoptymalizowanego scenariusza i weryfikowania bezpieczeństwa przy pomocy symbolicznego weryfikatora modelowego NuSMV i logiki temporalnej CTL. Rozdział dziewiąty "Wybrane problemy wytwarzania systemów czasu rzeczywistego dla bezpilotowych statków powietrznych" poświęcony jest analizie rozwiązań kilku wybranych problemów projektowania systemów czasu rzeczywistego dotyczących wytwarzania systemów informatycznych dla bezpilotowych statków powietrznych, a w szczególności zwiększeniu przewidywalności systemu w ramach zaistniałych ograniczeń. W rozdziale przedstawione zostały wybrane zagadnienia programowania komputera autopilota, jak również komunikacji w rozproszonym systemie sterowania i nadzorowania.

Mamy nadzieję, że lektura niniejszej monografii pozwoli czytelnikowi zapoznać się z aktualnymi rezultatami badań w obszarze wytwarzania oprogramowania, w tym także aplikacji czasu rzeczywistego. Przedstawione zagadnienia naukowe zostały bardzo szeroko poparte przykładami ich wdrożenia w rzeczywistych aplikacjach i systemach produkcyjnych. Liczymy, że lektura monografii zachęci naukowców i przedstawicieli szeroko pojętego przemysłu do dalszej, jeszcze ściślejszej współpracy w obszarze badań i rozwoju nowych systemów informatycznych.

Lech Madeyski
Piotr Kosiuczenko
Marek Bolanowski

Część 1
Narzędzia inżynierii oprogramowania w zastosowaniach praktycznych

Rozdział 1

Edytor graficzny języka LD dla pakietu CPDev

1. Wprowadzenie

Produkowane elementy aparatury kontrolno–pomiarowej, jak przetworniki pomiarowe, urządzenia wykonawcze, regulatory PID, wyświetlacze HMI, po połączeniu w systemy rozproszone służą zwykle do automatyzacji procesów małej i średniej wielkości. Niezbędne są jednak narzędzia inżynierskie, których podstawowym zadaniem jest kompilacja programu napisanego w jednym z języków normy IEC na pewien kod pośredni (lub wykonywalny) przeznaczony dla konkretnej platformy sprzętowej.

Środowisko stacji inżynierskiej CPDev (*Control Program Developer*, [2,5,6]) jest stale rozwijane w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej i stanowi pakiet inżynierski przeznaczony do programowania rozproszonych systemów sterowania zgodnie z normą IEC 61131-3 [1,3]. Obejmuje ono:

- edytory tekstowe języków ST i IL oraz graficzne języków LD, FBD i SFC,
- kompilator generujący uniwersalny kod binarny, który to po stronie sterownika wykonuje dedykowana maszyna wirtualna VM,
- symulator *off-line* (CPSim) umożliwiający wyświetlanie wartości zmiennych globalnych i lokalnych w celu sprawdzenia poprawności działania programu,
- konfigurator zasobów sprzętowych (CPCon) pozwalający na przypisanie adresów fizycznych do zmiennych globalnych oraz przydzielenie zadań komunikacyjnych dla wymiany danych z rozproszonymi urządzeniami I/O,
- tester *on-line* (CPSim) umożliwiający m.in. odczytanie wartości rzeczywistych sygnałów obiektowych oraz zmiennych globalnych i lokalnych.

Wśród istotnych cech tego środowiska warto wyróżnić dużą uniwersalność w sensie przeznaczenia dla różnych platform sprzętowych (takich jak na przykład 8-bitowe mikrokontrolery AVR czy też 32-bitowe procesory komputerów PC) oraz otwartość dla konstruktorów sterowników, inżynierów wdrażających CPDev oraz użytkowników.

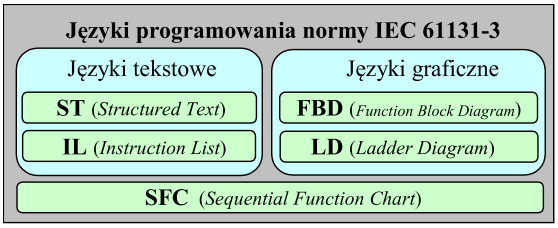
Obecnie pakiet CPDev używany jest głównie przez holenderską firmę Praxis Automation Technology [4], gdzie wybrane części pakietu CPDev zastosowano w systemie służącym do sterowania i monitorowania pracy statków, jak również przez studentów i pracowników Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej podczas zajęć dydaktycznych.

W rozdziale drugim przedstawiono języki programowania normy IEC 61131-3 zaimplementowane w środowisku CPDev, jego składniki i pełnione przez nie funkcje w systemie oraz interfejs użytkownika. Następnie omówiono okno edytora graficznego języka LD (interfejs użytkownika), kolejne etapy tworzenia diagramu oraz przedstawiono kilka reprezentatywnych przykładów schematów drabinkowych wraz z wygenerowanym kodem źródłowym. Na koniec krótko zasygnalizowano dodatkowe funkcjonalności edytora graficznego.

2. Charakterystyka ogólna pakietu CPDev

2.1. Języki programowania normy IEC 61131–3

Norma IEC 61131–3 [3] definiuje pięć języków programowania – LD, IL, FBD, ST i SFC, pozwalając dobrać język w zależności od problemu z uwzględnieniem preferencji projektanta (rys.1). Lista rozkazów IL oraz tekst strukturalny ST są językami tekstowymi, które to stosuje się zwykle w mniejszych aplikacjach oraz do tworzenia własnych bibliotek bloków, natomiast schematy drabinkowy LD, blokowy FBD oraz sekwencyjny SFC są językami graficznymi i występują głównie w aplikacjach średnich i dużych. Wspólnymi elementami języków są nazwy (identyfikatory), typy danych, stałe i zmienne. Do dyspozycji jest 20 typów elementarnych, wśród których najczęściej stosowanymi są: *BOOL*, *INT*, *REAL*, *TIME*. Podstawowymi elementami czytelnego oprogramowania są funkcje, bloki funkcjonalne i programy. Norma IEC 61131-3 definiuje stosunkowo niewielki zestaw standardowych bloków, którymi to są elementy dwustanowe, detektory zbocza, czasomierze i liczniki.



Rys. 1. Języki programowania normy IEC 61131-3

Warto zwrócić uwagę na zalety języków graficznych, których użycie może być prostsze i bardziej wygodne dla projektantów systemów sterowania (w porównaniu z językami tekstowymi). Ponadto, użycie języków graficznych pozwala na utworzenie programu w formie, która jest znacznie prostsza do zrozumienia i ewentualnej modyfikacji przez osoby nie znające szczegółów implementacji układu sterowania. Dodatkowo, opracowane diagramy i schematy graficzne mogą posłużyć jako załączniki do dokumentacji.

Wśród języków normy, tekst strukturalny ST jest językiem wysokiego poziomu wywodzącym się z Pascala czy C, nadającym się szczególnie do zapisu złożonych algorytmów. W wielu pakietach inżynierskich język ST jest więc językiem „domyślnym" do programowania własnych bloków funkcjonalnych – tak też jest w pakiecie CPDev.

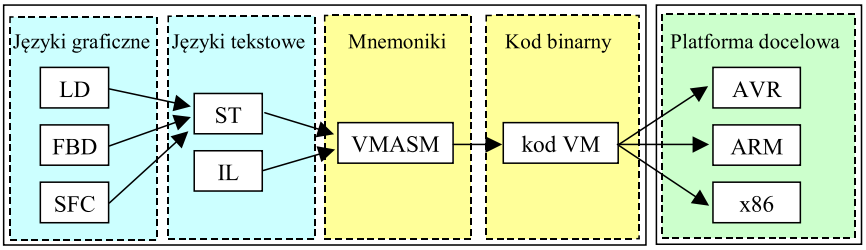
Dlatego też programy napisane w językach FBD i LD są tu konwertowane do ST, a następnie kompilowane do wspólnego kodu pośredniego.

2.2. Składniki środowiska CPDev

W skład standardowego środowiska CPDev [2,5,6] wchodzi pięć głównych modułów, z których po stronie komputera PC (czyli stacji inżynierskiej) to:

- edytory języków tekstowych ST, IL oraz graficznych LD, FBD i SFC,
- kompilator języków ST i IL na kod pośredni dla maszyny wirtualnej,
- symulator oprogramowania CPSim (*off-line* oraz *on-line*),
- konfigurator zasobów sprzętowych i komunikacji CPCon.

Programy wymieniają między sobą dane poprzez odpowiednie pliki. Kompilator CPDev (jest to zarazem nazwa całego środowiska) generuje uniwersalny kod wykonywalny (pośredni), który po stronie sterownika wykonuje maszyna wirtualna VM (*Virtual Machine*), pracująca jako interpreter. Kod wykonywalny jest listą instrukcji elementarnych języka tej maszyny nazywanego asemblerem VMASM (*VM Assembler*).



Rys. 2. Proces tworzenia i wykonywania kodu w pakiecie CPDev

Środowisko CPDev umożliwia tworzenie wszystkich jednostek organizacyjnych oprogramowania POU, czyli programów, bloków funkcjonalnych oraz funkcji. Nie ma również ograniczenia na liczbę programów wykonywanych w urządzeniu docelowym przez maszynę wirtualną. Wszystkie jednostki programowe przygotowane w językach graficznych (LD, FBD, SFC) są wstępnie przekształcane do kodu źródłowego w języku ST (rys.2). Następnie instrukcje języków tekstowych (ST, IL) są tłumaczone do kodu pośredniego w opracowanym na potrzeby platformy języku VMASM. Kod pośredni maszyny wirtualnej jest kompilowany do kodu binarnego, który to jest wykonywany przez maszynę wirtualną (VM) dedykowaną na konkretną platformę sprzętową (z procesorem AVR, ARM, x86 czy też innym). Warto dodać, że kod binarny układu sterowania dla maszyny wirtualnej jest niezależny od platformy docelowej, co umożliwia jego przenoszenie i wykonywanie na różnych sterownikach bez potrzeby ponownej rekompilacji.

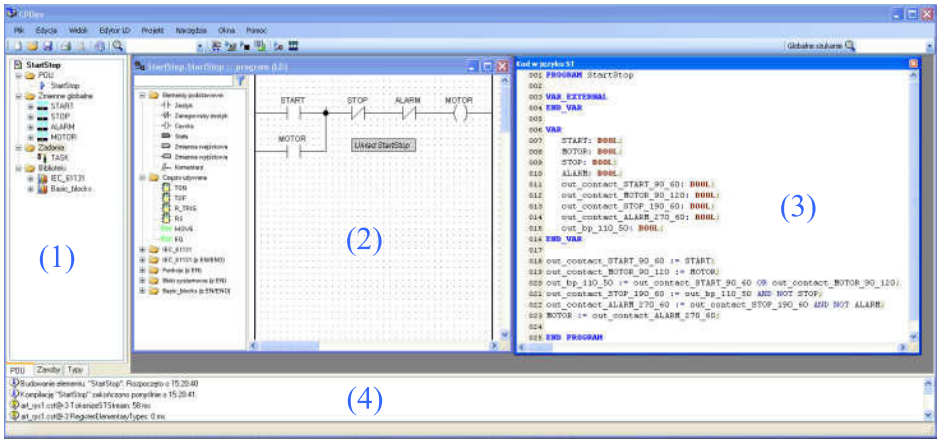
Każda platforma docelowa ma inną specyfikację wejść i wyjść obiektowych, więc konieczne jest skonfigurowanie i przypisanie zmiennych do odpowiadających im sygnałów obiektowych. Odpowiedzialnym za to jest moduł CPCon, w którym możemy przyporządkować poszczególne wejścia i wyjścia procesowe do konkretnych zmiennych globalnych oraz skonfigurować parametry zadań (np. czas cyklu) czy też komunikacji.

2.3. Interfejs programisty

Na rys. 3 pokazano główne okno interfejsu projektanta w pakiecie CPDev, wspólne dla wszystkich dostępnych języków programowania, zawierające:

- drzewo struktury projektu, konfiguracji sprzętowej, zasobów programowych (1),
- program w wybranym języku programowania, np. LD (2),
- okno konwersji schematu LD na kod programu w języku ST (3),
- okno komunikatów, zasobów i typów zmiennych, raportów (4).

Ramki okien można swobodnie regulować, zawartość przewijać, okna umieszczać w dowolnym miejscu lub zwijać do paska narzędziowego.



Rys. 3. Główne okno interfejsu użytkownika

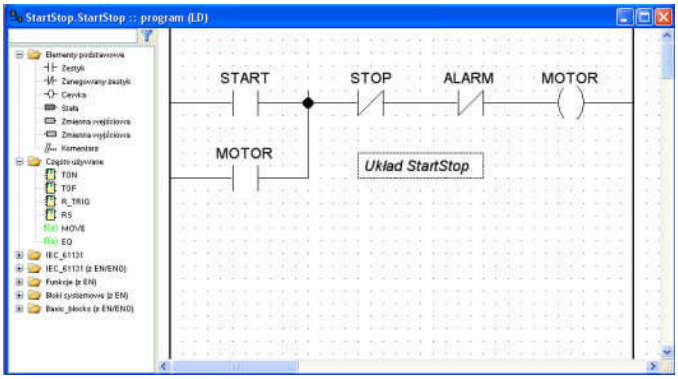
Schematy stworzone w edytorach graficznych LD, FBD i SFC są automatycznie przekształcane do programów w języku tekstowym ST, a następnie mogą być dowolnie modyfikowane i zmieniane. Podobnie bloki funkcjonalne napisane przez użytkownika w ST są automatycznie obrazowane w edytorze FBD i mogą być tam użyte do tworzenia kolejnych schematów blokowych. Dodatkowo, projektant może tworzyć własne funkcje, bloki funkcjonalne i programy gromadząc je w swoich bibliotekach do późniejszego wykorzystania.

3. Edytor graficzny języka LD

3.1. Programowanie w języku LD

Edytor graficzny dla środowiska programistycznego CPDev pozwala na tworzenie programów, funkcji oraz bloków funkcjonalnych w formie schematów drabinkowych. Schemat drabinkowy jest podobny do elektrycznych schematów połączeń stykowych i stanowi graficzną reprezentację równań boolowskich. Schemat LD z obu stron ograniczony jest szynami prądowymi (*power rails*), które reprezentują "zasilanie" elementów umieszczonych na szczeblach (*rungs*). Zestyki (*contacts*) mogą być

utożsamiane z wejściami procesowymi, zaś cewki (*coils*) – z wyjściami. Realizację operacji *OR* można wykonać łącząc zestyki równolegle, zaś operacji *AND* – szeregowo.



Rys. 4. Okno edytora graficznego LD

Podstawowe operacje edycyjne dla schematu LD wykonywane są podobnie, jak dla pozostałych edytorów graficznych dostępnych w środowisku CPDev (rys.4). Inne są jednak elementy, jakie mogą być umieszczane na schematach sterowania.

W lewej części interfejsu edytora graficznego znajduje się okno zawierające rozwijalne drzewo elementów, które można wybierać i umieszczać lub przeciągać do okna schematu na właściwą pozycję. Drzewo elementów zawiera:

- podstawowe elementy schematu: zestyki, cewki, zmienne i stałe,
- kilka najczęściej używanych bloków i funkcji (dla szybszego dostępu),
- standardowe bloki funkcjonalne z biblioteki (zgodnie z normą IEC_61131),
- te same bloki ale z dodatkowym wejściem sterującym *EN* (*Enable*)
- funkcje i bloki systemowe, również z dodatkowym wejściem *EN*.

W przeciwieństwie do standardowych bloków normy IEC 61131, wszystkie bloki i funkcje z wejściem *EN* są wykonywane tylko wtedy, gdy na wejściu EN występuje wartość logiczna *TRUE*. Pozwala to na umieszczanie na schematach drabinkowych LD funkcji i bloków funkcjonalnych typowych dla schematów FBD.

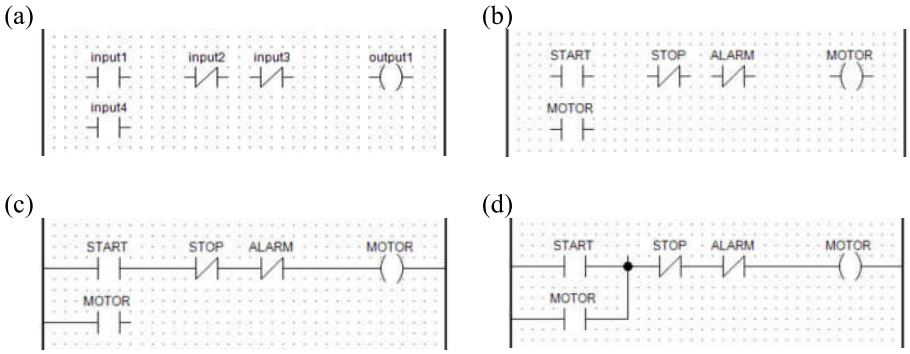
W przykładowym diagramie (rys. 4) na schemacie znajdują się zestyki (*START*, *STOP*, *ALARM*, *MOTOR*), cewka *MOTOR*, szyny prądowe oraz szczeble. Przy pomocy równoległego połączenia szczebli realizowana jest operacja *OR* pomiędzy wartościami zmiennych *START* i *MOTOR*. Na jej wyniku oraz wartościach zanegowanych zmiennych *STOP* i *ALARM* wykonywana jest z kolei operacja *AND*, co pozwala określić docelową wartość zmiennej *MOTOR*.

3.2. Tworzenie diagramu LD

Każdy szczebel diagramu tworzony jest w kolejnych krokach (rys.5) obejmujących:

- umieszczenie na schemacie wymaganych elementów (zestyków, cewek, ...),
- zmianę tymczasowych nazw (*input1*, ...) na docelowe (*START*, ...),
- ustawienie wartości dla stałych i zmiennych wejściowych i wyjściowych,
- wykonanie połączeń pomiędzy elementami (ręcznie myszką lub automatycznie),

– dodanie punktów rozgałęzień (*branch points*) i wykonanie połączeń do nich.



Rys. 5. Tworzenie diagramu: (**a)** umieszczenie elementów, (**b)** wpisanie nazw, (**c)** wykonanie połączeń szeregowych, (**d)** wykonanie połączeń równoległych.

Okno właściwości elementów (rys.6) pozwana na zmianę typu zestyku lub cewki (normalny, zanegowany, ...) na pożądany. Bloki ze standardowej biblioteki IEC_61131 (bez wejścia sterującego *EN*) mogą być umieszczane wewnątrz szczebla (pomiędzy zestykami a cewkami), natomiast bloki funkcjonalne i funkcje (z wejściem sterującym *EN*) muszą być umieszczane na końcu szczebla (zamiast cewki).

Rys. 6. Okno właściwości dla zestyku i cewki

3.3. Generowanie kodu ST i uruchamianie

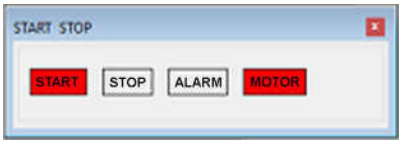
Podczas kompilacji schemat LD jest najpierw tłumaczony na program w języku ST poprzez dodanie pomocniczych zmiennych lokalnych (wykorzystywanych później do śledzenia programu i wyświetlania wartości zmiennych w poszczególnych punktach schematu) oraz zastąpienie połączeń szeregowych i równoległych przez odpowiednie operatory (AND, OR) i wywołania funkcji. Jeśli na schemacie występują błędy (brak połączenia, nieprawidłowy typ, ...), to wyświetlany jest odpowiedni komunikat.

Kod w języku ST

```
001 PROGRAM StartStop
002
003 VAR_EXTERNAL
004 END_VAR
005
006 VAR
007    START: BOOL;
008    MOTOR: BOOL;
009    STOP: BOOL;
010    ALARM: BOOL;
011    out_contact_START_90_60: BOOL;
012    out_contact_MOTOR_90_120: BOOL;
013    out_contact_STOP_190_60: BOOL;
014    out_contact_ALARM_270_60: BOOL;
015    out_bp_110_50: BOOL;
016 END_VAR
017
018 out_contact_START_90_60 := START;
019 out_contact_MOTOR_90_120 := MOTOR;
020 out_bp_110_50 := out_contact_START_90_60 OR out_contact_MOTOR_90_120;
021 out_contact_STOP_190_60 := out_bp_110_50 AND NOT STOP;
022 out_contact_ALARM_270_60 := out_contact_STOP_190_60 AND NOT ALARM;
023 MOTOR := out_contact_ALARM_270_60;
024
025 END_PROGRAM
```

Rys. 7. Program w języku ST otrzymany dla układu StartStop

Automatycznie wygenerowany program dla schematu StartStop rozpoczyna się nagłówkiem *PROGRAM StartStop* (rys.7). Sekcja *VAR_EXTERNAL...END_VAR* w tym przykładzie jest pusta. W sekcji *VAR...END_VAR* umieszczane są deklaracje zmiennych globalnych (*START*, *MOTOR*, ...), a także utworzonych automatycznie zmiennych lokalnych (dla wyjść poszczególnych elementów na schemacie). W tym przykładzie wszystkie zmienne są typu *BOOL*. Tworzone nazwy zmiennych lokalnych składają się z przedrostka (np. *out_contact*), identyfikatora występującego na schemacie (np. *START*) oraz współrzędnych elementu na schemacie (np. *_90_60*). Następnie, począwszy od lewej strony do prawej, analizowane są poszczególne elementy schematu i zamieniane na pojedyncze linie programu w języku ST, używając odpowiednich operatorów i wywołań funkcji bibliotecznych (AND, OR, NOT, ...).



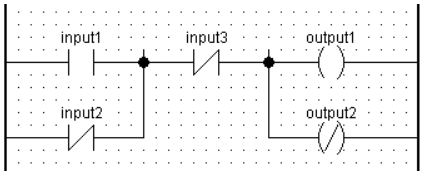
Rys. 8. Panel symulacji dla układu StartStop

Śledzenie wartości poszczególnych zmiennych podczas wykonywania programu dostępne jest w module CPSim. Wartości zmiennych mogą być wyświetlane liczbowo w formie listy, tabeli lub graficznie zebrane w panelu (zawierającym przyciski i lampki w formie prostokątów, rys.8). Zmienne umieszcza się na panelu przeciągając je bezpośrednio z drzewa projektu - dotyczy to zarówno zmiennych globalnych i lokalnych, jak i wyjść funkcji i bloków funkcjonalnych.

3.4. Przykłady schematów drabinkowych

Przykład 1 - proste połączenia szeregowo równoległe.

Na rys.9 przedstawiono przykład diagramu zawierającego proste połączenia szeregowo-równoległe (typowe dla schematów drabinkowych) oraz wygenerowany dla niego fragment kodu w języku ST. Oprócz standardowego zestyku i cewki, na schemacie użyto również ich wersji z negacją. Dodatkowo, na wyjściu zastosowano dwie cewki połączone równolegle.

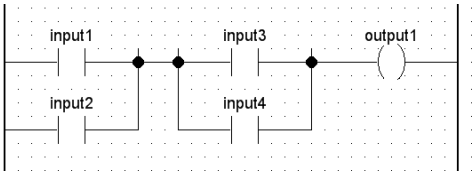



```
out_contact_input1_90_60 := input1;
out_contact_input2_90_120 := NOT input2;
out_bp_110_50 := out_contact_input1_90_60 OR out_contact_input2_90_120;
out_contact_input3_190_60 := out_bp_110_50 AND NOT input3;
out_bp_210_50 := out_contact_input3_190_60;
output1 := out_bp_210_50;
output2 := NOT out_bp_210_50;
```

Rys. 9. Diagram LD i wygenerowany kod w ST dla przykładu 1

Przykład 2 - mieszane połączenia szeregowo równoległe.

Na rys.10 przedstawiono przykład diagramu zawierającego mieszane połączenia szeregowo-równoległe oraz utworzony dla niego kod w ST. Należy zwrócić uwagę, że obliczenia prowadzone są od lewej strony do prawej, a dla połączeń równoległych od góry do dołu.

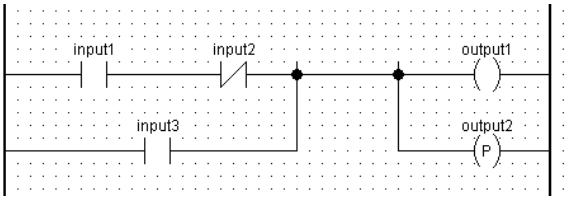



```
out_contact_input1_80_50 := input1;
out_contact_input2_80_100 := input2;
out_bp_100_40 := out_contact_input1_80_50 OR out_contact_input2_80_100;
out_bp_130_40 := out_bp_100_40;
out_contact_input3_210_50 := out_bp_130_40 AND input3;
out_contact_input4_210_100 := out_bp_130_40 AND input4;
out_bp_230_40 := out_contact_input3_210_50 OR out_contact_input4_210_100;
output1 := out_bp_230_40;
```

Rys. 10. Diagram LD i wygenerowany kod w ST dla przykładu 2

Przykład 3 - kolejne połączenia szeregowo równoległe.

Na rys.11 przedstawiono inny przykład diagramu zawierającego mieszane połączenia szeregowo-równoległe. Dodatkowo zastosowano cewkę, której wyjście ustawiane jest zboczem narastającym (przedostatnia linia kodu).

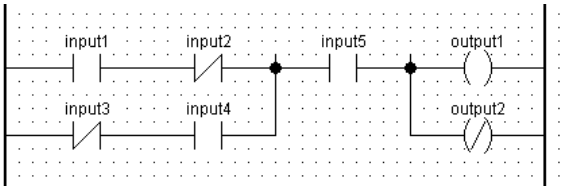



```
out_contact_input1_100_70 := input1;
out_contact_input2_210_70 := out_contact_input1_100_70 AND NOT input2;
out_contact_input3_150_130 := input3;
out_bp_230_60 := out_contact_input2_210_70 OR out_contact_input3_150_130;
out_bp_310_60 := out_bp_230_60;
output1 := out_bp_310_60;
output2 := out_bp_310_60 = TRUE AND out_coil_output2_410_130 = FALSE;
out_coil_output2_410_130 := out_bp_310_60;
```

Rys. 11. Diagram LD i wygenerowany kod w ST dla przykładu 3

Przykład 4 - kolejne połączenia szeregowo równoległe.

Kolejny diagram, przedstawiony na rys.12, zawiera dwie równoległe linie z szeregowo połączonymi zestykami, następnie równolegle, a na koniec szeregowo z sygnałem *input5*. Tutaj dodatkowo użyto cewki z zanegowanym wyjściem.

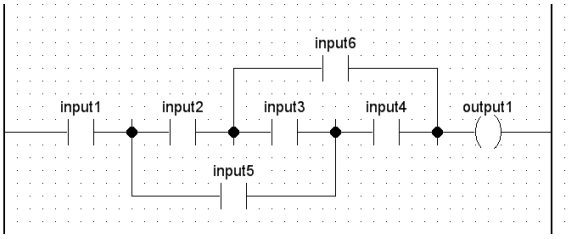



```
out_contact_input1_90_60 := input1;
out_contact_input2_180_60 := out_contact_input1_90_60 AND NOT input2;
out_contact_input3_90_110 := NOT input3;
out_contact_input4_180_110 := out_contact_input3_90_110 AND input4;
out_bp_200_50 := out_contact_input2_180_60 OR out_contact_input4_180_110;
out_contact_input5_280_60 := out_bp_200_50 AND input5;
out_bp_300_50 := out_contact_input5_280_60;
output1 := out_bp_300_50;
output2 := NOT out_bp_300_50;
```

Rys. 12. Diagram LD i wygenerowany kod w ST dla przykładu 4

Przykład 5 - złożone połączenia szeregowo równoległe ("mostek").

Na rys.13 przedstawiono diagram zawierający złożone połączenia szeregowo-równoległe, którego w wielu innych pakietach przemysłowych nie da się narysować.



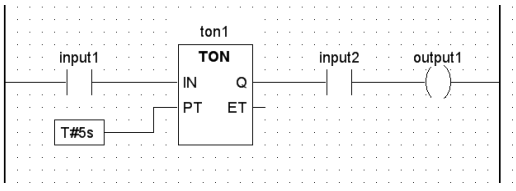

```
out_contact_input1_90_110 := input1;
out_bp_100_100 := out_contact_input1_90_110;
out_contact_input2_170_110 := out_bp_100_100 AND input2;
out_bp_180_100 := out_contact_input2_170_110;
out_contact_input6_290_60 := out_bp_180_100 AND input6;
out_contact_input3_250_110 := out_bp_180_100 AND input3;
out_contact_input5_210_160 := out_bp_100_100 AND input5;
out_bp_260_100 := out_contact_input3_250_110 OR out_contact_input5_210_160;
out_contact_input4_330_110 := out_bp_260_100 AND input4;
out_bp_340_100 := out_contact_input4_330_110 OR out_contact_input6_290_60;
output1 := out_bp_340_100;
```

Rys. 13. Diagram LD i wygenerowany kod w ST dla przykładu 5.

Tutaj analiza schematu odbywa się dwuetapowo. W pierwszym etapie schemat analizowany jest od lewej do prawej i budowana jest wstępna lista połączeń. Następnie schemat analizowany jest od prawej do lewej, uzupełniana jest lista połączeń oraz eliminowane są powtarzające się (zdublowane) połączenia.

Przykład 6 - umieszczenie bloku funkcjonalnego w szczeblu drabinki.

Na rys.14 przedstawiono diagram zawierający blok funkcjonalny TON umieszczony w drabince (w miejscu zestyku). Pierwsze wejście i wyjście bloku musi być podłączone do drabinki, a pozostałe do stałych lub zmiennych odpowiedniego typu. Kolejne wyjścia mogą zostać nie podłączone.

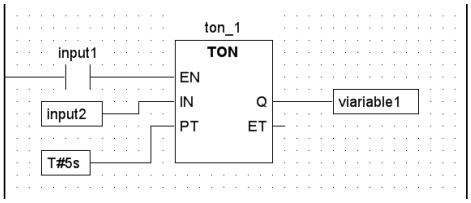


```
out_contact_input1_90_70 := input1;
ton1(IN:=out_contact_input1_90_70,PT:=T#5s,Q=>out_ton1_Q);
out_contact_input2_300_70 := out_ton1_Q AND input2;
output1 := out_contact_input2_300_70;
```

Rys. 14. Diagram LD i wygenerowany kod w ST dla przykładu 6

Przykład 7 - użycie bloku funkcjonalnego z wejściem *EN* (*Enable*).

Na rys.15 przedstawiono diagram zawierający blok funkcjonalny TON z wejściem *EN* (*Enable*). W odróżnieniu od poprzedniego, kod bloku TON jest wykonywany, gdy na wejściu *EN* wystąpi wartość *TRUE* (w kodzie użyto instrukcji *IF...END_IF*). Tutaj też tylko pierwsze wejście (*EN*) może być podłączone do szczebla, a wyjścia bloku mogą być podłączone tylko do zmiennych wyjściowych.

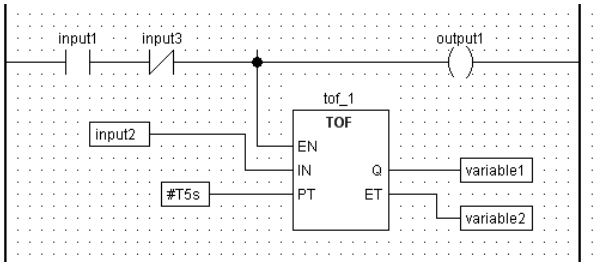



```
out_contact_input1_90_70 := input1;
IF out_contact_input1_90_70 THEN
ton_1(IN:=input2,PT:=T#5s,Q=>out_ton_1_Q);
viariable1 := out_ton_1_Q;
END_IF
```

Rys. 15. Diagram LD i wygenerowany kod w ST dla przykładu 7

Przykład 8 - kolejne użycie bloku funkcjonalnego z wejściem *EN*.

Kolejny przykład zastosowania bloku TOF z wejściem *EN* przedstawiono na rys.16. Tutaj wszystkie wejścia i wyjścia bloku są podłączone.

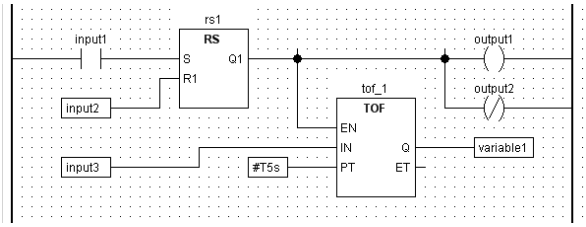


```
out_contact_input1_90_60 := input1;
out_contact_input3_160_60 := out_contact_input1_90_60 AND NOT input3;
out_bp_210_50 := out_contact_input3_160_60;
output1 := out_bp_210_50;
IF out_bp_210_50 THEN
tof_1(IN:=input2,PT:=#T5s,Q=>out_tof_1_Q,ET=>out_tof_1_ET);
variable1 := out_tof_1_Q;
variable2 := out_tof_1_ET;
END_IF
```

Rys. 16. Diagram LD i wygenerowany kod w ST dla przykładu 8

Przykład 9 - użycie dwóch bloków funkcjonalnych - z wejściem *EN* oraz bez.

Przykład zastosowania dwóch bloków funkcjonalnych przedstawiono na rys.17. W linii głównej zastosowano blok RS (bez wejścia *EN*), a jako wyjściowy (równolegle do cewek) blok TOF z wejściem *EN*.

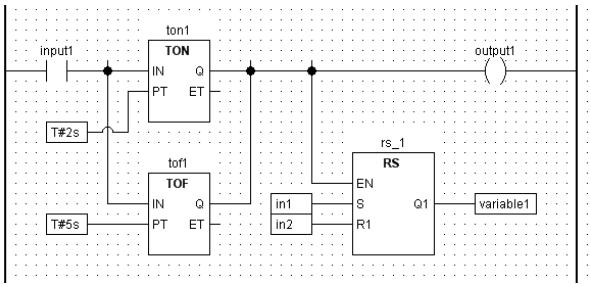



```
out_contact_input1_110_60 := input1;
rs1(S:=out_contact_input1_110_60,R1:=input2,Q1=>out_rs1_Q1);
out_bp_290_50 := out_rs1_Q1;
out_bp_440_50 := out_bp_290_50;
output1 := out_bp_440_50;
output2 := NOT out_bp_440_50;
IF out_bp_290_50 THEN
tof_1(IN:=input3,PT:=#T5s,Q=>out_tof_1_Q);
variable1 := out_tof_1_Q;
END_IF
```

Rys. 17. Diagram LD i wygenerowany kod w ST dla przykładu 9

Przykład 10 - mieszane użycie trzech bloków funkcjonalnych.

Przykład zastosowania trzech różnych bloków funkcjonalnych przedstawiono na rys.18.




```
out_contact_input1_80_80 := input1;
out_bp_100_70 := out_contact_input1_80_80;
ton1(IN:=out_bp_100_70,PT:=T#2s,Q=>out_ton1_Q);
tof1(IN:=out_bp_100_70,PT:=T#5s,Q=>out_tof1_Q);
out_bp_240_70 := out_ton1_Q OR out_tof1_Q;
out_bp_300_70 := out_bp_240_70;
output1 := out_bp_300_70;
IF out_bp_300_70 THEN
rs_1(S:=in1,R1:=in2,Q1=>out_rs_1_Q1);
variable1 := out_rs_1_Q1;
END_IF
```

Rys. 18. Diagram LD i wygenerowany kod w ST dla przykładu 10

3.5. Dodatkowe funkcjonalności

Rozwijanie gałęzi drzewa w celu wyszukania rzadko używanej funkcji lub bloku funkcjonalnego może być uciążliwe. Dzięki mechanizmowi filtrowania elementów drzewa wystarczy jedynie wpisać fragment jego nazwy w polu tekstowym znajdującym się nad drzewem, aby uruchomić automatyczne filtrowanie elementów. Prezentowane są wtedy wszystkie elementy, w których nazwie występuje wpisany ciąg liter. Dla przykładu, po wpisaniu tekstu "AND" pojawia się kilka gałęzi drzewa z elementami, w których nazwach występuje ciąg znaków "AND".

W drzewie elementów w lewym oknie dostępna jest także grupa *Często używane*, w której wyświetlane są elementy często wykorzystywane przez użytkownika przy tworzeniu diagramu. Z menu edytora możliwa jest zmiana algorytmu stosowanego do klasyfikowania elementów wyświetlanych w tej grupie, w tym *Ogólne*, czyli elementy często wybierane przez użytkownika, bądź też *Występujące razem* – to elementy wybierane przez użytkownika i występujące razem na diagramie.

Mechanizm tworzenia automatycznych połączeń umożliwia dodawanie elementów (zestyków, cewek, stałych, funkcji lub instancji bloków funkcjonalnych) bezpośrednio na istniejący szczebel. Przy dodawaniu nowego elementu następuje podzielenie istniejącej linii, a następnie połączenie wejść elementu z poprzednimi wyjściami, a wyjść z następnymi wejściami. W przypadku, gdy element dodawany jest w miejscu, w którym nie istnieje jeszcze połączenie, jest ono tworzone automatycznie i nowy element łączony jest z obydwiema szynami prądowymi.

Dzięki dodatkowym bibliotekom z blokami funkcjonalnymi i funkcjami z wejściem *EN* (*Enable*) możliwe jest tworzenie schematów sterowania przetwarzających nie tylko wartości typu BOOL, ale również *INT*, *FLOAT* czy *TIME* (podobnie jak np. w języku

FBD). Elementy z dodatkowym wejściem *EN* są wykonywane, gdy stan na tym wejściu jest równy *TRUE*. Schemat LD z tymi blokami i funkcjami podlega jednak pewnym ograniczeniom:

- każdy szczebel może zawierać tylko jeden element z wejściem *EN*,
- wejście EN może być podłączone do poprzedniego elementu (np. zestyku),
- pozostałe wejścia i wyjścia muszą być podłączone do zmiennych lub stałych.

4. Podsumowanie

W rozdziale przedstawiono najnowszą wersję edytora graficznego dla języka LD normy IEC61131. Środowisko stacji inżynierskiej CPDev (*Control Program Developer*) obejmuje edytory tekstowe języków ST i IL oraz graficzne języków LD, FBD i SFC, a także kompilator generujący uniwersalny kod binarny, symulator *off-line* oraz *on-line* (CPSim) umożliwiający wyświetlanie wartości zmiennych oraz konfigurator zasobów sprzętowych (CPCon) pozwalający na przypisanie adresów fizycznych i przydzielenie zadań komunikacyjnych dla wymiany danych.

W kolejnych punktach zaprezentowano interfejs użytkownika edytora graficznego języka LD, przedstawiono etapy tworzenia diagramu drabinkowego, generowany na podstawie schematu kod sterowania w języku LD oraz kilka reprezentatywnych diagramów obrazujące możliwości edytora graficznego i generatora kodu.

Przedstawione środowisko stacji inżynierskiej CPDev jest stale rozwijane i modyfikowane. Obecnie pakiet CPDev używany jest głównie przez holenderską firmę Praxis Automation Technology (w systemach służących do sterowania i monitorowania pracy statków) oraz przez studentów i pracowników Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej podczas zajęć dydaktycznych i realizacji prac dyplomowych.

Bibliografia

[1] CoDeSys – Continuous Function Chart – CFC – IEC 61131-3 – programming system – PLC – http://www.3ssoftware.com/index.shtml?en_CoDeSys_CFC.

[2] M. Jamro, D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus: Rozwój środowiska inżynierskiego CPDev do programowania systemów sterowania. [w:] Trybus L, Samolej S. (red.): Projektowanie, Analiza i Implementacja Systemów Czasu Rzeczywistego. WKŁ, Warszawa, 2011.

[3] PN-EN 61131-3 – Sterowniki programowalne. Część 3: Języki programowania. Warszawa, 2004.

[4] PRAXIS Automation Technology, Leiden, The Netherlands – http://www.praxis-automation.nl.

[5] D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus: Środowisko programistyczne dla rozproszonych minisystemów kontrolno–pomiarowych. Metody Informatyki Stosowanej, tom 13, nr 1, s. 199-221, 2008.

[6] A. Stec, Z. Świder, L. Trybus: Charakterystyka funkcjonalna prototypowego systemu do programowania systemów wbudowanych według normy IEC 61131-3. Systemy Czasu Rzeczywistego (red. Z. Huzar, Z. Mazur). Wydawnictwa Komunikacji i Łączności, Warszawa, 179–188, 2007.

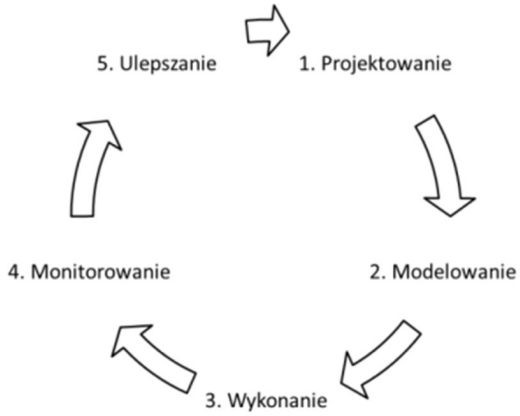
Rozdział 2

Implementacja systemu wspierającego zjawisko emergencji w modelowaniu procesów biznesowych

1. Wprowadzenie

Optymalne podejście do ustalania reguł kierujących procesami biznesowymi realizowanymi w organizacji, pozwalające architektom systemów na zautomatyzowanie procesów biznesowych w sposób najbardziej efektywny jest kluczowym zagadnieniem współczesnej inżynierii oprogramowania i złożonych systemów inżynierskich. Tradycyjne metody zarządzania procesami biznesowymi obejmują pięć etapów życia w cyklu procesu biznesowego: projektowanie procesu polegające na utworzeniu nowego lub zmianie istniejącego model procesu; tworzenie instancji procesu, jego konfigurację i uruchomienie; wykonywanie; monitorowanie realizacji instancji procesu; analizę skuteczności procesu i ulepszanie go (rys. 1).

W ostatnich latach można zaobserwować zasadnicze zmiany w podejściu do sterowania procesem biznesowym. Tradycyjne metody Business Process Management (BPM) są wspierane przez bardziej elastyczne podejście Adaptive Case Management (ACM) [18], w którym definiuje się zbiór możliwych operacji oraz reguły ich użycia, ale nie definiuje się sztywnego modelu procesu.



Rys. 1. Cykl życia procesu BPM

Rozdział przedstawia propozycję wykonania kolejnego kroku na drodze rozwoju dziedziny BPM: zastosowania teorii systemów samoorganizujących się, teorii emergencji, w powiązaniu z nowoczesnymi technikami zarządzania informacją, takimi jak wydobywanie danych (*data mining*), w celu opracowania systemu wspierającego funkcjonowanie organizacji realizującej procesy biznesowe w dynamicznie zmieniającym się środowisku. Jednym z aspektów omawianego rozwiązania jest pozyskiwanie wiedzy na temat przepływu procesu na podstawie rzeczywistych działań jego uczestników i automatyczna konstrukcja modelu procesu (np. Sieć Petri, diagram BPMN) opisującego zależność przyczynową pomiędzy działaniami. Idea eksploracji danych polega na wykorzystaniu właściwości komputera (szybkości, mocy obliczeniowej, pojemności pamięci itd.) do znajdowania prawidłowości ukrytych dla człowieka, z uwagi na ograniczone możliwości przetwarzania, na podstawie informacji zgromadzonej w zbiorach danych lub pojawiającej się w strumieniach danych.

Na początku rozdziału zostaną przedstawione najważniejsze pojęcia związane z zarządzaniem procesami biznesowymi oraz aktualne trendy w tym obszarze. Następnie, wprowadzone zostanie pojęcie emergencji, jako zjawiska, które pozwala w nowy sposób spojrzeć na zarządzanie procesami biznesowymi. Zdefiniowane zostaną warunki, które według niektórych autorów muszą być spełnione, aby zjawisko to mogło zachodzić w grupie współpracujących ze sobą osób. Na tej podstawie, zostanie zaproponowany model systemu informatycznego wspierającego realizację procesów biznesowych w organizacji, którego zadaniem jest aktywne wykorzystywanie potencjału emergencji na drodze realizacji celów biznesowych (*Emergence-Aware System*). Wstępnie zostanie scharakteryzowana odpowiedzialność każdego z komponentów systemu oraz zostanie opisany zbiór narzędzi informatycznych i technologii programistycznych wykorzystanych do budowy rozwiązania. Kolejny rozdział zostanie poświęcony charakterystyce centralnego składnika systemu: modułu przetwarzania wiedzy (*Knowlege Engine*), o kluczowym znaczeniu dla gromadzenia i przetwarzania wiedzy i doświadczenia pojawiających się w organizacji. Dalej, w sposób poglądowy zostanie przedstawiony podstawowy schemat działania systemu, uwzględniający pojawianie się i rozwój zjawiska emergencji. Schemat ten zostanie później rozwinięty w kierunku szczegółowego opisu interakcji pomiędzy modułami systemu za pomocą diagramów sekwencji. Zostanie także przedstawiona wymagana parametryzacja biznesowa systemu obejmująca m.in. konfigurację silnika procesu, modelowanie schematu „wydobywania procesów", czy konfigurację modułu reputacyjnego.

2. Zarządzanie procesami biznesowymi

Jedną z metod opisujących realizowany proces w organizacji jest notacja graficzna, która jest specjalnie sformalizowaną i rozszerzoną wersją diagramu aktywności UML o nazwie *Business Process Model and Notation (BPMN)* [5]. Pozwala definiować działania, które występują w procesie biznesowym i wszystkie możliwe przepływy między nimi, a także zdarzenia zachodzące podczas realizacji procesu. Jednym ze sposobów opisania wymaganych funkcji systemu informacyjnego jest przypadek użycia (PU). Opisuje on w formie scenariusza interakcję między aktorem (zwykle użytkownikiem systemu) a systemem, który składa się z dobrze określonych akcji

przypisywanych każdej ze stron. PU można zdefiniować przy użyciu różnych poziomów abstrakcji, od najbardziej ogólnych, które opisują proces biznesowy, do najbardziej specyficznych, które reprezentują pojedynczą interakcję (działanie) w procesie. W wyniku realizacji celów procesu biznesowego (tj. realizacji kolejnych PU, które obejmuje proces) zmienia się stan systemu. Celem realizacji procesu jest doprowadzenie systemu do określonego, pożądanego stanu końcowego. Najczęściej system uruchamiany i pracujący zgodnie z określonym modelem wymaga dalszej obserwacji i korekty modelu na podstawie badań empirycznych. W dziedzinie zarządzania przepływem danych wyróżnić można inne alternatywne podejścia. Jednym z nich jest podejście *Adaptive Case Management*, w którym problemy rozwiązuje się poprzez ponowne wykorzystanie wcześniejszych rozwiązań dla podobnych problemów, po ich niezbędnym dostosowaniu do nowej sytuacji. Wcześniejsze „zamrożenie" definicji procesów powinno zostać odrzucone, a wiedza, doświadczenie i ciągła poprawa powinny być praktykowane na bieżąco w miarę wzrostu wiedzy i doświadczenia w organizacji.

Wspomniana wcześniej możliwość wykorzystania procesu odkrywczego jest bardzo cenna dla rozwoju następnej generacji systemów informacyjnych PAIS (*Process-Aware Information Systems*), ponieważ podejście takie wyraźnie pokazuje niezgodność między modelami proponowanymi do realizacji tych systemów a rzeczywistością. Z jednej strony modele mają tendencję do nadmiernego uproszczenia, opisując zbyt restrykcyjne systemy, a z drugiej, modele nie potrafią uchwycić ważnych aspektów procesów biznesowych. PAIS [1] definiuje się jako system oprogramowania, który zarządza procesami operacyjnymi obejmującymi osoby, aplikacje i / lub źródła informacji w oparciu o modele procesów. Systemy te opierają się na procesach, które obejmują zasoby ludzkie i oprogramowanie do osiągnięcia konkretnych celów. Systemy takie rejestrują informacje dotyczące wykonywania instancji procesów i wykonywane czynności w dziennikach zdarzeń. Każde zdarzenie reprezentuje realizację instancji aktywności przez dany zasób w pewnym momencie wraz z produkowaną informacją wyjściową. Głównym zadaniem procesu ekstrakcji jest analiza dzienników zdarzeń, zrozumienie i poprawa procesów w oparciu o fakty.

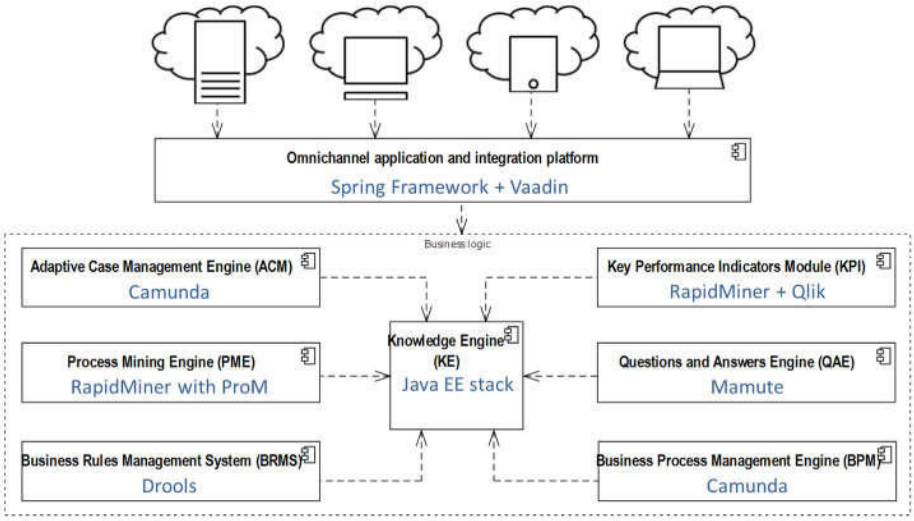
3. Nowe podejście wykorzystujące zjawisko emergencji

Emergencję definiujemy jako zjawisko zachodzące tam, gdzie system wzajemnie powiązanych elementów posiada możliwości, których nie można wyprowadzić opierając się jedynie na możliwościach elementów składowych [12]. System taki wykazuje zdolność działania przewyższającą sumę umiejętności pojedynczych aktorów. Zaobserwowano, że zjawisko emergencji zachodzi tylko wówczas, gdy spełnione są określone warunki. Najczęściej wymienia się ich pięć [14]:

1. system składa się z wystarczająco dużej liczby aktorów ("Wszystko czego potrzebujemy, to tysiące jednostek i kilka prostych reguł działania" [14]),
2. aktorzy otrzymują informację zwrotną z otoczenia,
3. pomiędzy aktorami występuje stała i nieskrępowana możliwość wymiany wiedzy i doświadczeń,
4. aktorzy posiadają możliwość i umiejętność rozpoznawania powtarzających się wzorców,

5. aktorzy nie podlegają odgórnej kontroli; istnieją natomiast mechanizmy sprzyjające wzajemnej kontroli zachodzącej pomiędzy współpracującymi jednostkami.

W rozdziale zaproponowano nowe podejście do budowy systemu informatycznego, pozwalające na wykorzystanie potencjału zjawiska emergencji. Bazując na przedstawionych wymaganiach, przy użyciu podejścia analitycznego stworzono wymagany model systemu, który jest nazywany Emergence-Aware System (EAS). Model komponentów systemu przedstawiono na rys. 2. Oprócz struktury logicznej, diagram zawiera także informację o proponowanej implementacji poszczególnych składników, która szerzej zostanie omówiona w kolejnych rozdziałach.



Rys. 2. Model komponentowy omawianego systemu

Omikanałowa platforma aplikacyno-integracyjna (*Omnichannel application and integration platform*) zapewnia dwukierunkową, asynchroniczną łączność pomiędzy urządzeniami wykorzystywanymi przez użytkowników systemu a opisanymi dalej modułami systemu. Warstwa ta jest także odpowiedzialna za prezentację ekranów użytkownika (*Graphical User Interface, GUI*) oraz zarządzaniem interfejsem programistycznym systemu (*Application Programming Interface, API*). Komponent *Adaptive Case Management Engine (ACM)* jest modułem, który pełni w systemie rolę głównego silnika procesowego, umożliwiającego aktorom realizację ich zadań biznesowych i dającego im możliwość optymalizacji sposobu ich wykonania, jak również dostosowania do zmieniających się warunków w otoczeniu biznesowym. Silnik ACM pozwala aktorom na wytyczenie własnej ścieżki prowadzącej do wyznaczonego celu, jednocześnie rejestrując realizowane kroki i budując bazę wiedzy do późniejszej analizy i wnioskowania z wykorzystaniem opisanego dalej modułu *Knowledge Engine*. Komponent *Process Mining Engine (PME)* jest modułem zapewniającym możliwość odkrywania realizowanych procesów biznesowych na podstawie logów zdarzeń otrzymanych w wyniku aktywności uczestników procesów realizujących swoje

codzienne zadania. Otrzymane modele opisują rzeczywiste procesy zachodzące w organizacji. Komponent *Business Rules Management System (BRMS)* to moduł, który przechowuje, wyzwala i monitoruje wykonanie fragmentów logiki biznesowej, które mogą zostać wydzielone poza bazę kodu oprogramowania i zapisane w specjalnym języku interpretowanym przez oprogramowanie nazywane silnikiem regułowym. W systemach BPM taki rodzaj oprogramowania służy często do określania spełnienia reguł zapisanych w węzłach decyzyjnych występujących w modelu BPMN. Kolejnym składnikiem systemu jest komponent *Key Performance Indicators Module (KPI)*, którego zadaniem jest obliczanie wskaźników efektywnościowych na podstawie danych pojawiających się w systemie. Moduł umożliwia definiowanie potrzebnych wskaźników oraz przeprowadza cykliczne wyliczenia ich aktualnych wartości. Wyniki są prezentowane uczestnikom procesów w postaci pozwalającej na samodzielne wnioskowanie (wykresy, tabele) oraz są wysyłane do modułu *Knowledge Engine*, gdzie stanowią dane wejściowe do dalszych przetwarzań. Komponent *Questions and Answers Engine (QAE)* zapewnia wygodny sposób komunikacji pomiędzy uczestnikami procesów w celu wymiany wiedzy i doświadczeń. Aktorzy realizujący zadania biznesowe mogą w prosty sposób zadawać pytania, udzielać odpowiedzi, formułować propozycje, czy dzielić się spostrzeżeniami. Moduł oferuje specjalny system punktacji, którego zadaniem jest stymulacja aktywnej wymiany doświadczeń. Nagradzanie wartościowych pytań i odpowiedzi realizowane w sposób oddolny przez użytkowników systemu biorących udział w komunikacji prowadzi do ustalania wskaźnika reputacji poszczególnych osób. Wskaźnik ten stanowi istotną informację dla analiz prowadzonych w module *Knowledge Engine*. Komponent *Business Process Management Engine (BPM)* to drugi, obok ACM, element odpowiedzialny za sterowanie przepływem pracy. W tym przypadku jest to silnik pracujący zgodnie z podejściem BPM, a jego zadaniem jest automatyczne wykonywanie procesów odkrytych za pomocą modułu PME wszędzie tam, gdzie uczestnicy procesów lub są zobligowani do pracy na podstawie ustalonego przebiegu procesu lub decydują się na wykorzystanie wsparcia (podpowiedzi) systemu w zakresie realizacji kolejnych kroków. Moduł ten umożliwia także orkiestrację mikro-sekwencji kilku czynności w postaci gotowego podprocesu wykorzystywanego później przez silnik ACM. Kluczowym składnikiem rozwiązania jest moduł *Knowledge Engine (KE)*, którego główną odpowiedzialnością jest integracja pozostałych modułów, prowadzenie analiz prowadzących do odkrywania prawidłowości charakterystycznych dla zjawiska emergencji oraz wykorzystanie zgromadzonej wiedzy do automatyzacji procesów. Integralnym składnikiem modułu jest baza wiedzy wypełniana treścią wraz ze wzrostem doświadczenia powstającego w działającym systemie.

4. Charakterystyka wybranego oprogramowania

Implementacja systemu została wykonana z wykorzystaniem istniejącego oprogramowania ogólnodostępnego. Silnik BPM zintegrowano z oprogramowaniem zapewniającym funkcjonalność odkrywania procesów, dodano moduły odpowiedzialne za obliczanie i wizualizację KPI oraz oprogramowanie wspierające swobodną wymianę wiedzy. Szczególną uwagę zwrócono na zgodność technologiczną poszczególnych składników systemu, możliwość ich powiązania z wykorzystaniem jednolitych technik

integracyjnych oraz możliwość ekspozycji funkcji systemu za pomocą jednego interfejsu użytkownika (GUI) i spójnego interfejsu programistycznego (API). Przyjęto, że integracja pomiędzy modułami będzie realizowana zgodnie z modelem Representational State Transfer (REST) [9], wybrano więc oprogramowanie eksponujące API w tym modelu. Główne składniki systemu opisano poniżej.

4.1. Silnik procesowy Camunda

Camunda [6] jest systemem sterowania przepływem pracy umożliwiającym wykorzystanie zarówno podejścia BPM, jak i podejścia ACM. Camunda implementuje wersję 2.0 standardu BPMN oraz wersję 1.1 standardu CMMN. Dodatkowo, system wspiera ewaluację reguł decyzyjnych modelowanych zgodnie ze standardem *Decision Model and Notation (DMN)*. W pakiecie systemu znajduje się kilka modułów: *BPMN/CMMN/DMN Engine* – właściwy silnik procesowy, *Modeler* – niezależna aplikacja desktopowa umożliwiająca modelowanie procesów, *Tasklist* – aplikacja web-owa dla uczestnika procesów umożliwiająca uruchamianie zadań oraz zarządzanie nimi, *Cockpit* – aplikacja web-owa dla osoby odpowiedzialnej za zarządzanie instancjami procesów (monitorowanie, reagowanie na nieprawidłowości), *Admin* – aplikacja web-owa dla administratora systemu pozwalająca m.in. na zarządzanie użytkownikami, grupami i uprawnieniami do systemu. Camunda jest zaimplementowana w Javie, posiada możliwość integracji z różnymi typami relacyjnej bazy danych oraz dobrze udokumentowany interfejs API zgodny z modelem REST.

4.2. RapidMiner z rozszerzeniem RapidProM

RapidMiner [23] jest rozbudowanym środowiskiem przyspieszającym budowanie narzędzi do analiz danych. Pozwala na elastyczne budowanie definicji przetwarzania danych wykorzystując do tego notację graficzną opartą ma schemacie blokowym. Posiada bogatą bibliotekę gotowych elementów, zwanych operatorami, które można ze sobą łączyć w sekwencje, by następnie skierować strumień danych wejściowych, otrzymując na wyjściu (lub wyjściach) biznesowo użyteczną informację. RapidMiner składa się z dwóch głównych części: *Studio* – aplikacja desktopowa przeznaczona do modelowania procesów przetwarzania, *Server* – środowisko wykonywalne, w którym odbywa się osadzanie i wyzwalanie przygotowanych procesów. Część serwerowa rozwiązania pozwala na ekspozycję interfejsu API w modelu REST. RapidProM [21] jest wersją narzędzia ProM przygotowaną w postaci wtyczki do środowiska RapidMiner. ProM to środowisko oferujące zestaw zaawansowanych narzędzi z obszaru *process mining*, tworzone od wielu lat przez grupę pracującą pod kierownictwem autora najbardziej wpływowego opracowania w tej dziedzinie [2] i zarazem lidera badań nad tym zagadnieniem.

4.3. Qlik

Qlik [22] jest rozwiązaniem z obszaru *business intelligence*, w którym szczególną uwagę poświęcono użytecznej i atrakcyjnej wizualizacji danych. Dwa podstawowe moduły systemu to *Sense* – kompletne środowisko do eksploracji i prezentacji danych

oraz *View* – narzędzie przeznaczone do integracji z aplikacjami w celu wizualizacji znajdujących się tam danych. W przypadku omawianego systemu, wykorzystywana jest możliwość integracji modułu Qlik View z aplikacją RapidMiner, co w efekcie pozwala na prezentację obliczanych w RapidMiner wskaźników KPI oraz innych zagregowanych danych pochodzących przede wszystkim z logów silnika procesowego Camunda. Użytkownicy systemu posiadają dostęp w trybie ciągłym do odpowiednio przygotowanej informacji dotyczącej osiąganych przez nich wyników.

4.4. Silnik Q&A Mamute

Przykładem ogólnodostępnego silnika Questions & Answers jest Mamute [17]. Mamute jest oprogramowaniem w całości napisanym w Javie z warstwą prezentacji opracowaną z wykorzystaniem platformy programistycznej VRaptor [26], w konfiguracji domyślnej współpracującym z bazą danych MySQL, z możliwością podłączenia do innej relacyjnej bazy danych. Narzędzie udostępnia API zgodne z modelem REST.

4.5. Silnik regułowy Drools

Drools [6] jest silnikiem regułowym pracującym w oparciu o algorytm Rete [10]. W skład środowiska promowanego pod hasłem *Knowledge Is Everything* wchodzą przede wszystkim: *Workbench* – aplikacja web-owa do zarządzania silnikiem, *Expert* – właściwy silnik regułowy, *Fusion* – silnik *Complex Event Processing* [16]. Silnik regułowy pozwala na przedstawienie reguł biznesowych w sposób deklaratywny, w odróżnieniu od najczęściej wykorzystywanego podejścia imperatywnego. Do zapisu reguł używana jest własna notacja DRL (*Drools Rule Language*), w której reguła posiada typową strukturę z dwiema sekcjami *when* i *then*. Silnik eksponuje API oparte na modelu REST.

4.6. Platformy programistyczne Spring Framework i Vaadin

Spring [24] jest szkieletem tworzenia aplikacji serwerowych w języku Java. Zawiera gotowe rozwiązania większości problemów technicznych, z jakimi spotykają się programiści aplikacji tego typu. Pozwala na szybkie budowanie gotowych rozwiązań bez konieczności tworzenia tzw. *boilerplate code*, czyli kodu koniecznego do działania aplikacji, ale niestanowiącego jej logiki biznesowej. Spring wspiera większość istotnych elementów architektury systemu: wstrzykiwanie zależności, dostęp do danych i obsługa transakcji, prezentacja GUI, wywołania usług zdalnych, autoryzacja i uwierzytelnianie, obsługa kolejek itd. Zawiera także rozbudowane wsparcie tworzenia aplikacji wielokomponentowych, w tym aplikacji rozproszonych. Biorąc pod uwagę wymagania proponowanego rozwiązania, zaletą szkieletu Spring są rozbudowane mechanizmy integracyjne (przede wszystkim naturalne wsparcie modelu REST), które pozwoliły na zbudowanie spójnego rozwiązania w oparciu o komponenty różnych dostawców. Vaadin [25] to środowisko tworzenia w języku Java graficznego interfejsu użytkownika pracującego w przeglądarce internetowej. Aplikacja zaimplementowana z wykorzystaniem tego oprogramowania pracuje zarówno na serwerze, jak i w przeglądarce, przy czym nie jest wymagane tworzenie kodu w języku JavaScript, ani kodu w języku HTML *(HyperText Markup Language)*. Twórcy szkieletu Vaadin przyjęli

reguły architektoniczne i programistyczne stosowane w szkielecie Spring, co zaowocowało doskonałą integracją obu środowisk oraz jednolitym podejściem do programowania.

4.7. Baza danych PostgreSQL

PostgreSQL [20] jest jednym z najpopularniejszych ogólnodostępnych systemów zarządzania relacyjną bazą danych (*Relational Database Management System, RDBMS*). Posiada dystrybucje dedykowane dla większości obecnie użytkowanych systemów operacyjnych i typów architektury sprzętowej. Większość środowisk programistycznych, w tym opisany wcześniej Spring Framework posiada odpowiednie mechanizmy dostępu do bazy PostgreSQL, co w powiązaniu z jej cechami wydajnościowymi i użytkowymi powoduje, że jest często wybieranym rozwiązaniem zarówno dla zastosowań rozwojowo-badawczych, jak i komercyjnych.

4.8. Serwer usług katalogowych Apache Directory

Ostatnim modułem systemu, który nie bierze bezpośredniego udziału w procesach biznesowych, ale jest wymaganym elementem integracyjnym jest serwer usług katalogowych. Jego głównym zadaniem jest identyfikacja tożsamości użytkowników i zapewnienie jej weryfikacji na żądanie wszystkich pozostałych modułów. Rozwiązaniem, które zostało wybrane jest Apache Directory [4], jedno z bardziej popularnych rozwiązań tego typu, oprogramowanie napisane w Javie i zgodne ze standardami LDAP *(Lightweight Directory Access Protocol)* i Kerberos.

5. Podstawowy schemat działania systemu

Moduł, który został nazwany *Knowledge Engine (KE)* jest centralną częścią systemu, odpowiedzialną za odkrywanie symptomów zjawiska emergencji i umożliwienie wykorzystania jego siły. Komponent otrzymuje informacje wynikające z obliczeń odbywających się w innych częściach systemu, przetwarza je i gromadzi wiedzę w postaci wymaganej do dalszej automatyzacji procesów. Przykładowo, moduł KE otrzymuje z modułu odpowiedzialnego za odkrywanie procesów (PME) informację o modelach BPMN wynikających z rzeczywistej aktywności uczestników procesów, konfrontuje tę informację ze wskaźnikami otrzymanymi z modułu KPI oraz wiedzą na temat reputacji uczestników pochodzącą z modułu QAE. Wynikowa wiedza pozwala na wydanie rekomendacji dotyczącej najkorzystniejszych sposobów kontynuacji procesu.

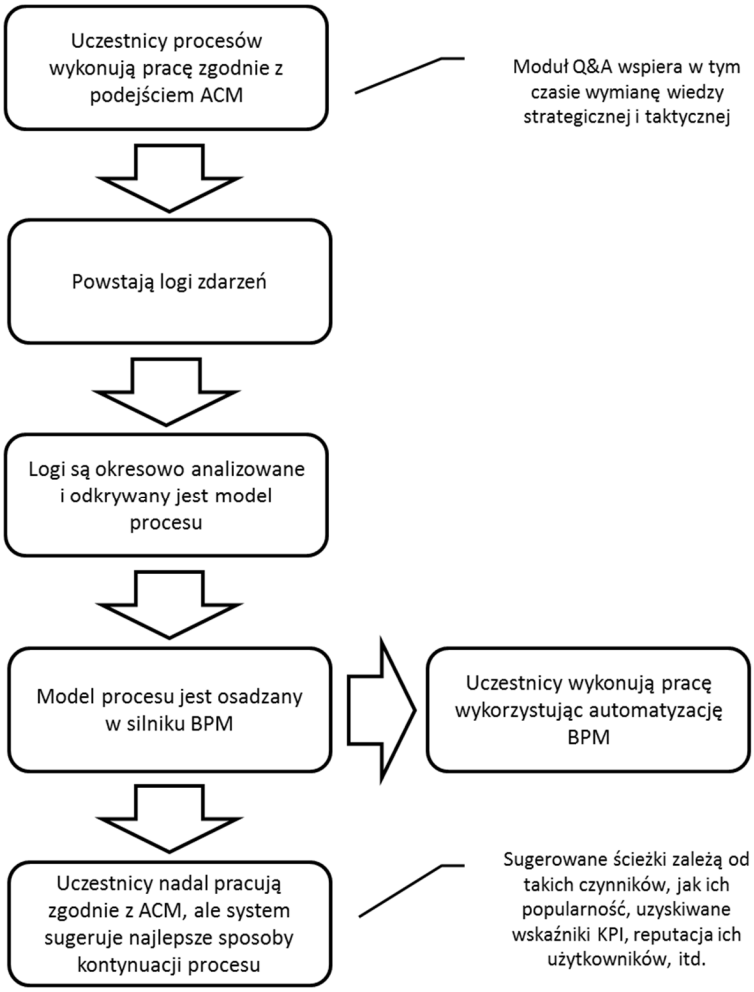
Moduł KE jest najdynamiczniej rozwijającym się składnikiem systemu. Do aktualnie realizowanych zadań należą:

- transformacja danych pomiędzy modelami funkcjonującymi w poszczególnych modułach (np. dane historyczne z API Camunda – log zdarzeń w formacie XES, model procesu w formacie PNML – API Camunda do zarządzania modelem procesu, itd.),
- przechowywanie bazy wiedzy propozycjonalnej tworzonej na podstawie realizowanych zadań,

- przechowywanie bazy wiedzy proceduralnej będącej efektem odkrywania procesów,
- kategoryzacja uczestników procesów ze względu na różne kryteria (reputacja, osiągane wyniki, staż, funkcja, itp.),
- oznaczanie i podpowiadanie najkorzystniejszych ścieżek,
- przyznawanie umownych nagród wynikających z analizy reputacji i wskaźników KPI (nagradzanie reputacji wzmacnia poziom współpracy warunkującej zachodzenie emergencji).

Moduł KE został oprogramowany w języku Java z wykorzystaniem stosu technologicznego, którego podstawowymi elementami są: opisany wcześniej Spring Framework, oraz dodatkowo: Akka – framework umożliwiający programowanie współbieżne w modelu aktorowym [11, 3], Quartz – biblioteka udostępniająca funkcje harmonogramowania przetwarzań. Dane modułu są utrwalane w bazie relacyjnej PostgreSQL. Moduł posiada interfejs GUI wykonany z użyciem Vaadina oraz eksponuje interfejs API modelowany zgodnie z podejściem REST.

Na rys. 3 przedstawiono podstawowy schemat działania systemu. Opiera się on na założeniu, że procesy są realizowane w organizacji przez uczestników charakteryzujących się różnym poziomem wiedzy i doświadczenia. System stosując analizę wielokryterialną w sposób ciągły kategoryzuje uczestników, co pozwala na aktywne wykorzystanie wiedzy pochodzącej od „liderów" poprzez wspieranie wszystkich członków organizacji. Wsparcie jest realizowane m.in. poprzez sugestie najlepszych sposobów kontynuacji procesu albo oddanie do dyspozycji gotowych modeli całych procesów. Ponadto, wszyscy aktorzy na każdym etapie działania otrzymują informację zwrotną, która pozwala im na korektę sposobu postępowania, bądź adaptację do zmieniających się warunków. Szczegóły dotyczące interakcji zachodzących pomiędzy modułami systemu w poszczególnych krokach schematu przedstawiono w rozdziale 7.



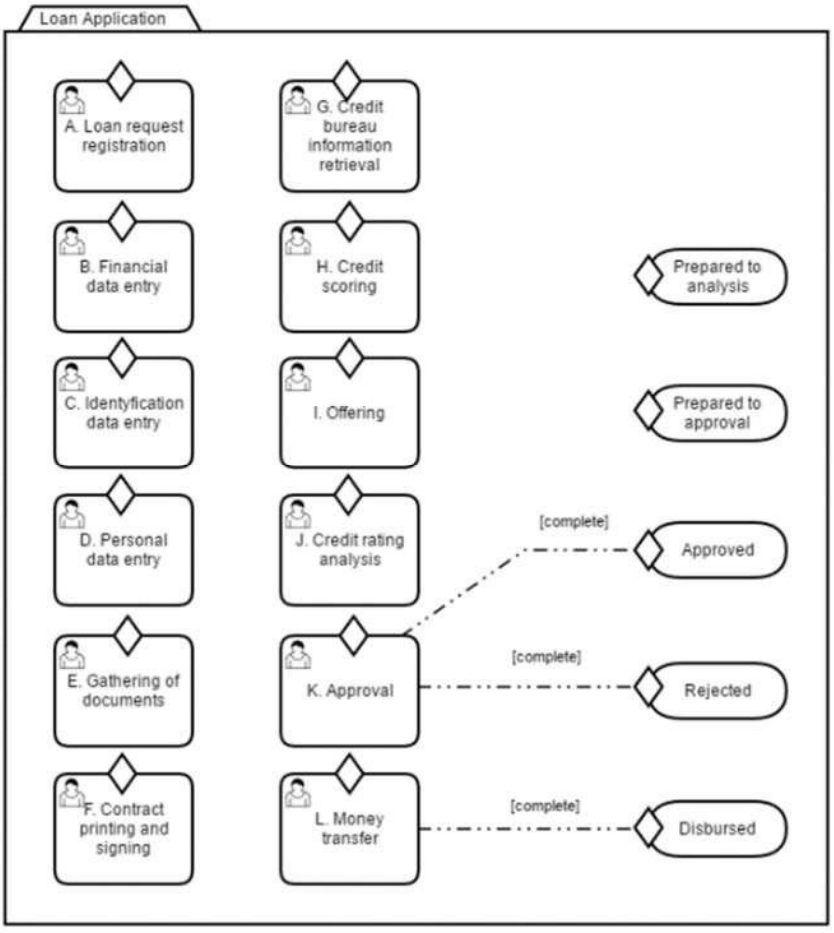
Rys. 3. Podstawowy schemat działania systemu

6. Konfiguracja biznesowa systemu

Omawiany system przed oddaniem do użytkowania wymaga przeprowadzenia konfiguracji biznesowej. W jej zakresie znajdują się m.in.: zdefiniowanie ról pełnionych przez uczestników procesów, rejestracja uczestników, zdefiniowanie czynności (przypadków użycia), modelowanie procedur wydobywania procesów i danych, ustalenie punktacji w module reputacyjnym.

6.1. Apache Directory

Pierwszą czynnością konfiguracyjną jest zdefiniowanie ról i uczestników systemu realizowane w module Apache Directory zgodnie z protokołem aplikacyjnym LDAP [15]. Standard ten umożliwia elastyczne definiowanie modelu danych w postaci wpisów posiadających atrybuty o określonych nazwach. W przypadku bazy użytkowników, wpisy są zwykle organizowane w postaci drzewa katalogowego, bazującego na strukturze hierarchicznej lub lokalizacyjnej danej organizacji. W omawianym systemie baza LDAP przechowuje definicje ról, konta użytkowników systemu oraz przyporządkowanie użytkowników do ról. Dodatkowo znajdują się w niej definicje tych słowników, które posiadają zasięg całego systemu (są wykorzystywane przez wiele modułów).



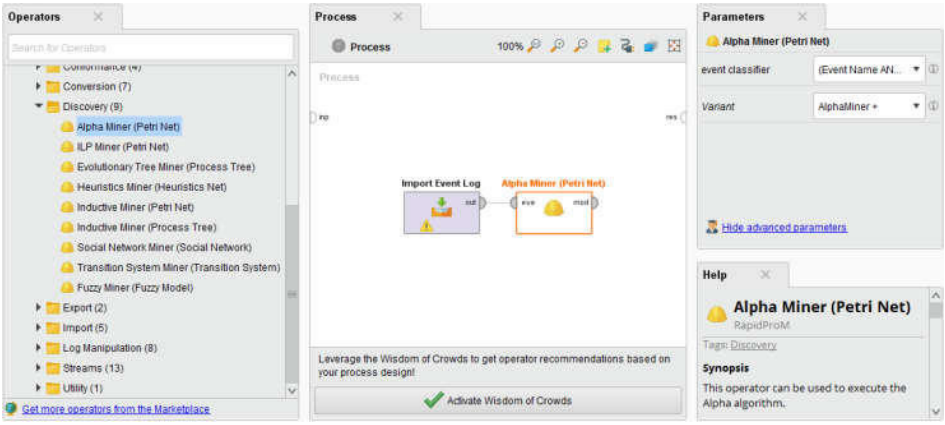
Rys. 4. Model CMMN

6.2. Camunda

Silnik procesowy Camunda pełni dwojaką rolę: jako silnik BPM i jako silnik ACM. Definicje procesów dla silnika BPM powstają automatycznie w wyniku realizacji procedur odkrywających procesy. Istnieje także możliwość definiowania ich poprzez projektowanie za pomocą narzędzia Modeler, co ma miejsce głównie tam, gdzie są wykorzystywane mikro-sekwencje o znanej strukturze. Podstawową czynnością konfiguracyjną jest definiowanie przypadków użycia dla silnika ACM w notacji CMMN. Przykład definicji zestawu PU dla procesu sprzedaży kredytów gotówkowych przedstawiono na rys. 4. Dla każdego z przypadków użycia konieczne jest zdefiniowanie warunków wstępnych, jakie musi spełniać system, aby przypadek został uruchomiony. Warunki takie są ustalane poprzez wprowadzenie wartowników (oznaczonych na diagramie znakiem rombu), sprawdzających zdefiniowane reguły bazujące na danych przetwarzanych w procesie. Możliwe jest także zdefiniowanie kamieni milowych, które są osiągane przez proces po wykonaniu określonych czynności (jak np. kamień oznaczony „Approved") lub po spełnieniu reguły przechowywanej przez wartownika (jak np. dla kamienia „Prepared to analysis"). Ponadto, każdy z PU występujących w modelu musi zostać przypisany do roli (lub kilku ról) jakie są odpowiedzialne za jego wykonanie. Na podstawie tak przygotowanego modelu silnik ACM automatyzuje powoływanie zadań przeznaczonych do realizacji oraz sygnalizuje postęp w realizacji procesu poprzez osiąganie kolejnych kamieni milowych.

6.3. RapidMiner z rozszerzeniem RapidProM

Konfiguracja procedury wydobywania danych odbywa się w programie RapidMiner Studio i sprowadza się do stworzenia modelu przepływu strumieni danych z wykorzystaniem notacji graficznej opartej na schemacie blokowym (rys. 5).



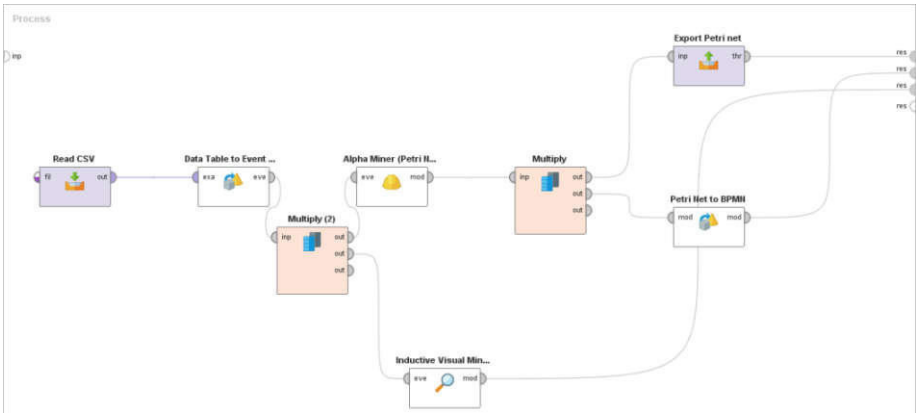
Rys. 5. Modelowanie przepływu w RapidMiner Studio

Najprostszy model przeznaczony do wykorzystania w omawianym systemie, przedstawiony na rys. 6, składa się z trzech operatorów:

- *Import Event Log* – operator realizujący wczytanie logu w standardzie XES [8] do wewnętrznych struktur RapidMiner,
- *Alpha Miner (Perti Net)* – operator realizujący procedurę odkrywania procesu według algorytmu Alpha [2],
- *Export Petri Net* – operator eksportujący odkryty model procesu w postaci sieci Petri zapisanej w formacie PNML [19].

Rys. 6. Prosty model przepływu

Model ten może zostać rozwinięty po to, by uzyskać dodatkowe informacje wyjściowe lub w przypadku, gdy zachodzą inne potrzeby integracyjne. Procedura przedstawiona na rys. 7 oprócz struktury PNML generuje diagram procesu w notacji BPMN oraz diagram odchyleń będący wynikiem pracy narzędzia *Inductive visual Miner*. Ponadto, zmieniona procedura oczekuje danych wejściowych w postaci pliku CSV (*comma-separated values*), zamiast XES.



Rys. 7. Rozbudowana wersja modelu przepływu

6.4. Mamute

Konfiguracja systemu punktacji. Ważnym elementem każdego systemu reputacyjnego jest właściwie skonstruowany system nagród i kar, który z jednej strony musi zachęcać do aktywnego uczestnictwa w procesie wymiany wiedzy, a z drugiej, powinien zapobiegać celowym nadużyciom. Domyślna konfiguracja w systemie Mamute przewiduje następującą politykę [17]:

- autor pytania otrzymuje 5 punktów za każdy głos „za”,
- autor odpowiedzi otrzymuje 10 punktów za każdy głos „za”,
- autor odpowiedzi otrzymuje 20 punktów, jeśli jego odpowiedź zostanie wybrana jako najbardziej poprawna,

- użytkownik, który wybrał odpowiedź jako najbardziej poprawną otrzymuje 5 punktów,
- za każde zadane pytanie, każdą udzieloną odpowiedź oraz każdą zaakceptowaną poprawkę do pytania lub odpowiedzi innego autora, użytkownik otrzymuje 2 punkty,
- użytkownik otrzymuje 1 punkt jeśli jego komentarz otrzymał głos „za”,
- autor pytania lub odpowiedzi z głosem „przeciw” traci 2 punkty,
- użytkownik, który udzieli głosu „przeciw” traci 2 punkty; punkty wracają do użytkownika, jeśli inni użytkownicy również udzielają podobnej dezaprobaty.

Konfiguracja progów uprawnień. Możliwe jest także zarządzanie progami punktacji wymaganej do realizacji poszczególnych akcji w systemie. Zgodnie z ustawieniami domyślnymi, użytkownik musi posiadać na swym koncie:

- 10 punktów, aby zagłosować na pytanie, odpowiedź lub komentarz,
- 10 punktów, aby oznaczyć pytanie lub odpowiedź,
- 20 punktów, aby zamieścić propozycję modyfikacji pytania lub odpowiedzi.
- 20 punktów, aby odpowiedzieć na swoje własne pytanie,
- 100 punktów, aby udzielić głosu „przeciw”,
- 2000 punktów, aby akceptować lub odrzucać zmiany wprowadzone przez innych użytkowników.

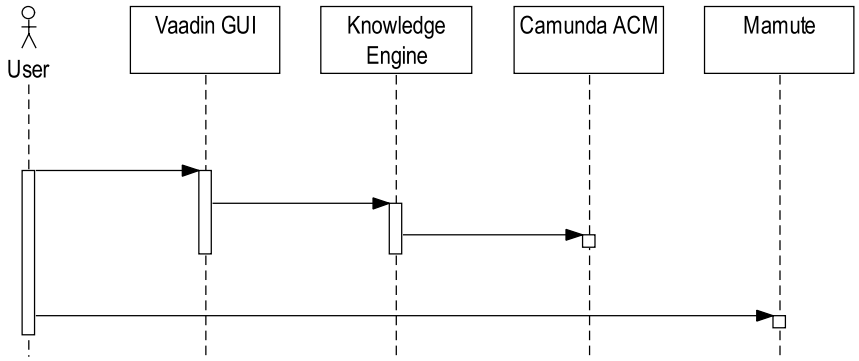
Polityka punktacji zaimplementowana w Mamute jest tak skonfigurowana, aby zabezpieczyć interesy użytkowników, chroniąc ich przed nieuzasadnionym karaniem, szczególnie ze strony uczestników „niezasłużonych”, z niską reputacją, zachęcając jednocześnie do dzielenia się wiedzą i do poszukiwania wiedzy, co jest głównym zadaniem stawianym przed tym modułem systemu.

7. Współpraca modułów systemu

Podstawowy schemat działania systemu przedstawiony w rozdziale 5 opiera się na współpracy wszystkich jego komponentów. Obecnie, za pomocą diagramów sekwencji zostaną przedstawione najważniejsze interakcje, jakie zachodzą w systemie.

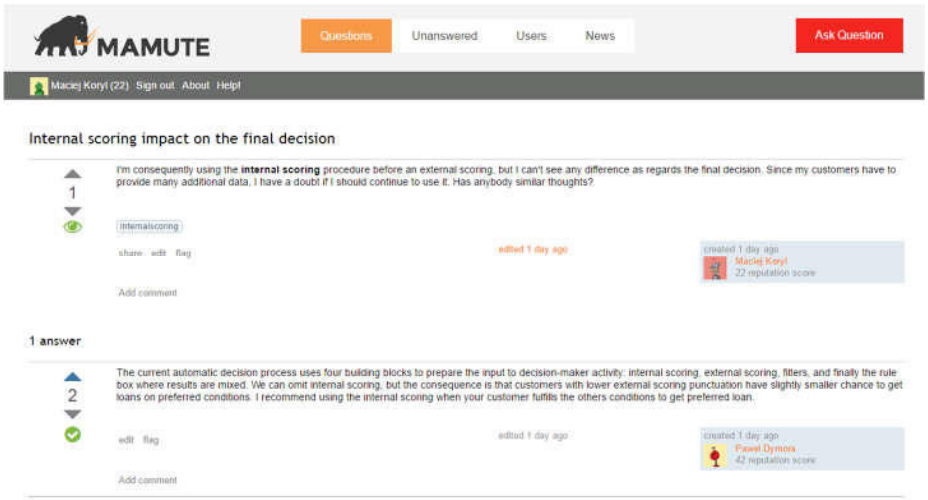
7.1. Aktor pracuje zgodnie z podejściem ACM

Pierwsza sekwencja przedstawiona na rys. 8 dotyczy sytuacji, gdy uczestnik procesu realizuje swoją pracę zgodnie z podejściem ACM. Aktor wchodzi w interakcję z oprogramowaniem za pomocą interfejsu GUI, uzyskując z systemu zbiór zadań przeznaczonych do realizacji na danym etapie procesu. Lista zadań pochodząca z silnika procesowego Camunda może zostać wzbogacona o dodatkowe informacje przygotowane w module KE.



Rys. 8. Aktor pracuje zgodnie z podejściem ACM

Moduł Mamute jest w tym przypadku użycia wykorzystywany jako narzędzie wspierające wymianę wiedzy strategicznej („jak podejść do zagadnienia") i taktycznej („jak to zrobić') pomiędzy uczestnikami procesów zaangażowanymi w realizację zadań biznesowych. Główny ekran użytkownika tego systemu został przedstawiony na rys. 9.

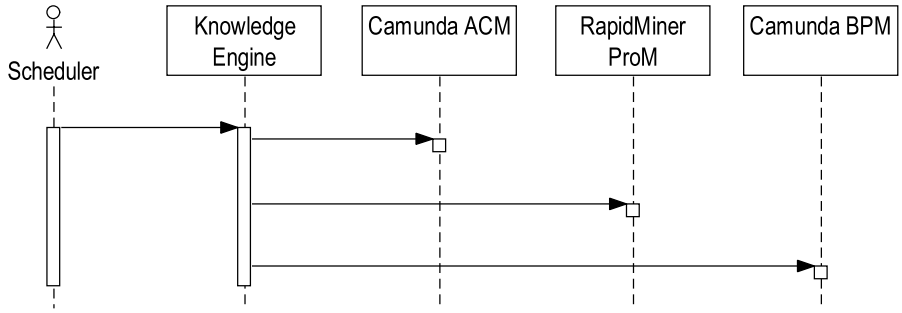


Rys. 9. Wymiana wiedzy w Mamute

Ponieważ oprócz funkcji wymiany wiedzy, Mamute posiada również funkcję obliczającą reputację użytkowników na podstawie wyników z systemu głosowań, system ten realizuje procedury zapobiegające łatwemu i nieuzasadnionemu zdobywaniu punktów. Użytkownik może stracić określoną liczbę punktów w przypadku, gdy system stwierdzi nadużycie. Procedury kontrolne opierają się główne na ustawionych limitach i zależnościach czasowych.

7.2. System realizuje odkrywanie procesu

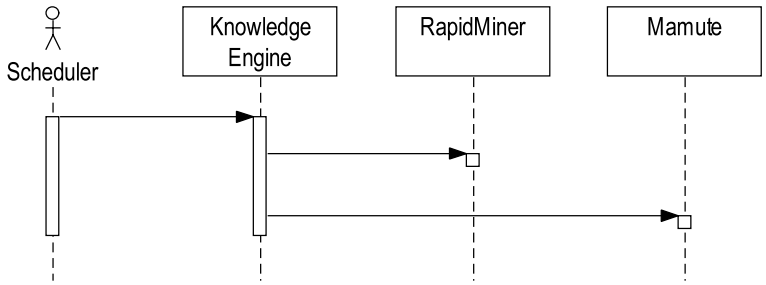
Moduł KE pobiera cyklicznie z zasobów silnika Camunda dane dotyczące historii realizowanych procesów. Dane te przekształca do postaci logów zdarzeń w standardzie XES. Logi te, poprzez wyeksponowane funkcje API silnika RapidMiner są przesyłane do procedury wydobywania procesu realizowanej z wykorzystaniem rozszerzenia ProM. Otrzymany model procesu w postaci PNML jest wprowadzany poprzez funkcje API do struktur silnika BPM w celu wykorzystania jako gotowy modelu procesu instancjonowany przez użytkowników pracujących zgodnie z podejściem BPM lub jako wzór sekwencji do wykorzystania w podejściu ACM. Realizowaną interakcję przedstawiono na rys. 10.



Rys. 10. System realizuje odkrywanie procesu

7.3. System buduje bazę wiedzy

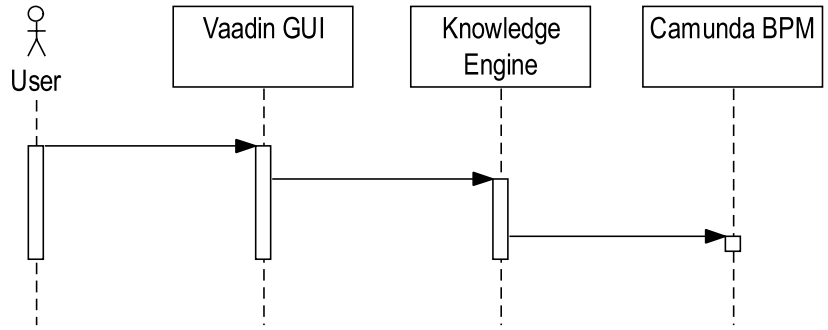
Wykonując analizę danych pochodzących z poszczególnych modułów, system buduje bazę wiedzy, która będzie wykorzystywana do realizacji przyszłych instancji procesów. Na rys. 11 przedstawiono przykład cyklicznego pobierania informacji z modułu Mamute oraz wyliczania wskaźników KPI z użyciem modułu RapidMiner. Proces jest realizowany automatycznie, zgodnie z ustalonym harmonogramem.



Rys. 11. System buduje bazę wiedzy

7.4. Aktor realizuje pracę zgodnie z podejściem BPM

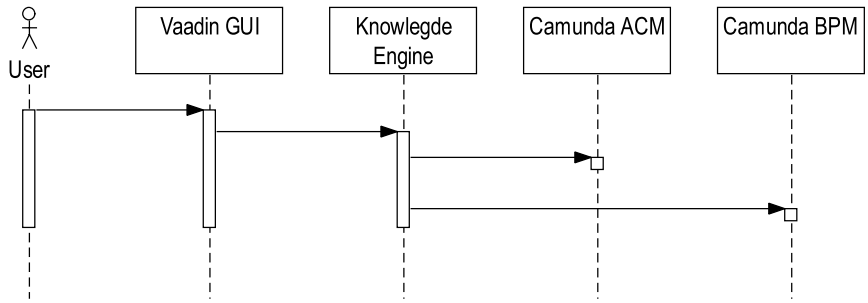
Jednym z efektów odkrywania procesów są gotowe modele całych procesów, które są potrzebne uczestnikom decydującym się na pracę zgodnie z podejściem BPM (lub są do takiego podejścia zobligowani). Moduł KE tworzy w tym celu w silniku Camunda kompletne, gotowe do wykorzystania definicje procesów, które następnie są instancjonowane przez aktorów i używane do pracy (rys. 12).



Rys. 12. Aktor realizuje pracę zgodnie z podejściem BPM

7.5. Aktor realizuje pracę zgodnie z podejściem ACM ze wsparciem BPM

Najbardziej zaawansowanym przypadkiem wykorzystania systemu jest praca polegająca na łączeniu podejścia ACM i BPM. W tym przypadku wykorzystywane są oba silniki procesowe Camunda (rys. 13), a moduł KE wykorzystując zgromadzoną wiedzę realizuje automatyczne przełączenie pomiędzy trybami lub przygotowuje użytkownikowi informację umożliwiającą podjęcie decyzji o najlepszym sposobie kontynuacji procesu.



Rys. 13. Aktor realizuje pracę zgodnie z podejściem ACM ze wsparciem BPM

8. Podsumowanie

W rozdziale przedstawiono model systemu informatycznego, którego zadaniem jest wspieranie automatyzacji procesów biznesowych w organizacji w oparciu o nowe podejście wykorzystujące zjawisko emergencji, czyli samoczynnego wyłaniania się wyższych form organizacyjnych w systemach składających się z dużej liczby współpracujących jednostek. Omówiono realizację systemu przy wykorzystaniu wyselekcjonowanego oprogramowania ogólnodostępnego, które poddano integracji, uzyskując jednolite i spójne rozwiązanie. Kolejnym etapem prac badawczych będzie staranny wybór zbioru przypadków użycia i wyznaczenie celu biznesowego, a następnie konfiguracja i wdrożenie oprogramowania w licznej grupie współpracujących ze sobą uczestników, po to by uzyskać odpowiednio obszerną bazę materiału badawczego pozwalającego na analizę zachodzących zjawisk. Oczekuje się, że przedstawiony system pozwoli potwierdzić hipotezę o możliwości zachodzenia zjawiska emergencji w odniesieniu do procesów biznesowych realizowanych w organizacjach.

9. Bibliografia

[1] M. Dumas, W.M.P van der Aalst, A. H. ter Hofstede: *Process Aware Information Systems: Bridging People and Software Through Process Technology*, Process Aware Information Systems: Bridging People and Software Through Process Technology, Wiley-Interscience, 2005.

[2] W.M.P van der Aalst: *Process Mining. Data Science in Action*, Springer, 2016.

[3] Akka Framework, http://akka.io.

[4] Apache Directory, http://directory.apache.org.

[5] Business Process Model and Notation (BPMN). Object Management Group (OMG), http://www.bpmn.org.

[6] Camunda, https://camunda.org.

[7] Drools, https://www.drools.org.

[8] Extensible Event Stream (XES), http://www.xes-standard.org.

[9] Fielding R. T.: *Architectural Styles and the Design of Network-based Software Architectures*, PhD dissertation, University Of California, Irvine, 2000.

[10] C. Forgy, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, AI, 1982.

[11] C. Hewitt, P. Bishop, R. Steiger: *A Universal Modular Actor Formalism for Artificial Intelligence*. IJCAI'73 Proceedings of the 3rd IJCoAI, 1973, pp. 235–245.

[12] J. H. Holland: *Emergence: From Chaos to Order*, Perseus Publishing, 1999.

[13] J. Jin, Y. Li, X. Zhong, L. Zhai: *Why users contribute knowledge to online communities: An empirical study of an online social Q&A community*, Information & Management, Volume 52, 2015.

[14] S. Johnson: *Emergence: The Connected Lives of Ants, Brains,Cities & Software*, Touchstone, NY, 2001.

[15] Lightweight Directory Access Protocol (LDAP), https://tools.ietf.org/rfc/rfc4511.txt.

[16] D. Luckham: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley, 2002.

[17] Mamute Q&A Engine, http://www.mamute.org.

[18] Motahari-Nezhad H. R., Swenson K. D.: *Adaptive Case Management: Overview and Research Challenges*, IEEE International Conference on Business Informatics, 2013.

[19] Petri Net Markup Language (PNML), http://www.pnml.org.

[20] PostgreSQL, https://www.postgresql.org.

[21] ProM Tool 6.6, http://www.promtools.org.

[22] Qlik, http://www.qlik.com.

[23] RapidMiner, http://www.rapidminer.com.

[24] Spring Framework, http://spring.io.

[25] Vaadin, https://vaadin.com.
[26] VRaptor, http://www.vraptor.org.
[27] G. Wang, K. Gill, M. Mohanlal, H. Zheng, B. Y. Zhao: *Wisdom in the Social Crowd: an Analysis of Quora*, The 22nd International World Wide Web Conference, ACM, 2013.

Rozdział 3

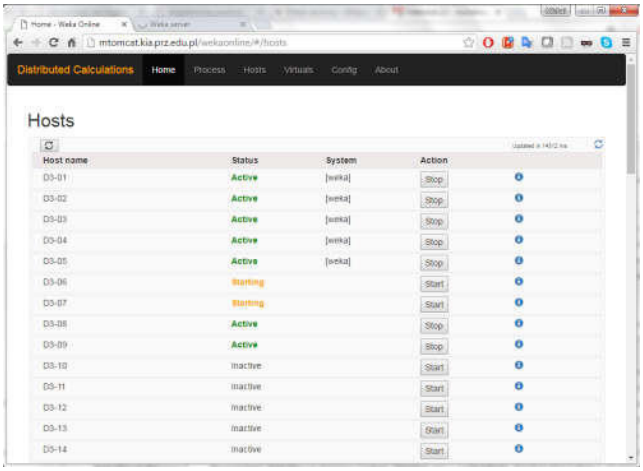
Budowa systemu do obliczeń rozproszonych z wykorzystaniem typowej infrastruktury laboratoryjnej

1. Wstęp

Dla prostych obliczeń często wystarczający jest jeden procesor, podczas gdy do rozwiązywania bardziej złożonych zagadnień jedynym sposobem na osiągnięcie celu jest użycie wielu procesorów. Zagadnienia takie, jak na przykład prognozowanie pogody, inżynieria biomedyczna, optymalizacja zarządzania zasobami przedsiębiorstw, wymaga znacznych zasobów obliczeniowych, przewyższających znacznie możliwości pojedynczych komputerów. W realizacji wymienionych zadań algorytmy inteligencji obliczeniowej, metody eksploracji danych i odkrywania wiedzy implikują potrzebę zwiększonych zasobów mocy obliczeniowej [1], do których jednostki akademickie mogą mieć utrudniony bezpośredni dostęp. Okazuje się jednak, że uczelniane laboratoria komputerowe wyposażone w znaczną liczbę komputerów stacjonarnych, są w stanie zapewnić sporą moc obliczeniową.
Autorzy postawili sobie za cel przygotowanie środowiska, które zapewnia:

- znaczną moc obliczeniową dla obliczeń rozproszonych z wykorzystaniem laboratoriów wyposażonych w komputery o niewielkiej mocy obliczeniowej,
- możliwość zdalnego sterowania zasobami obliczeniowymi,
- możliwość zdalnego zlecania zadań obliczeniowych,
- minimalną ingerencję w konfigurację sprzętowo-programową istniejącej infrastruktury.

W analizowanym przypadku zestaw jednostek obliczeniowych stanowi kilkadziesiąt komputerów stacjonarnych umieszczonych w różnych budynkach uczelni, połączonych szybką siecią LAN. Zarządzanie zasobami obliczeniowymi dokonywane jest zdalnie z opracowanego panelu administracyjnego dostępnego z poziomu przeglądarki internetowej, który pokazano na rys. 1.



Rys. 1. Panel do zdalnego zarządzania jednostkami obliczeniowymi

W niniejszej pracy pokazano metodę konfiguracji środowiska obliczeniowego, która zapewnia spełnienie wyżej wymienionych warunków, w przypadku raczej typowej, uczelnianej infrastruktury laboratoryjnej. Opisano będzie implementację systemu, z możliwością zdalnego administrowania i monitorowania. Podano wyniki eksperymentów z wykorzystaniem zaproponowanego środowiska do obliczeń rozproszonych, na przykładzie rozwiązania pewnych problemów z zakresu inteligencji obliczeniowej, w których wykorzystano pakiet Weka [10].

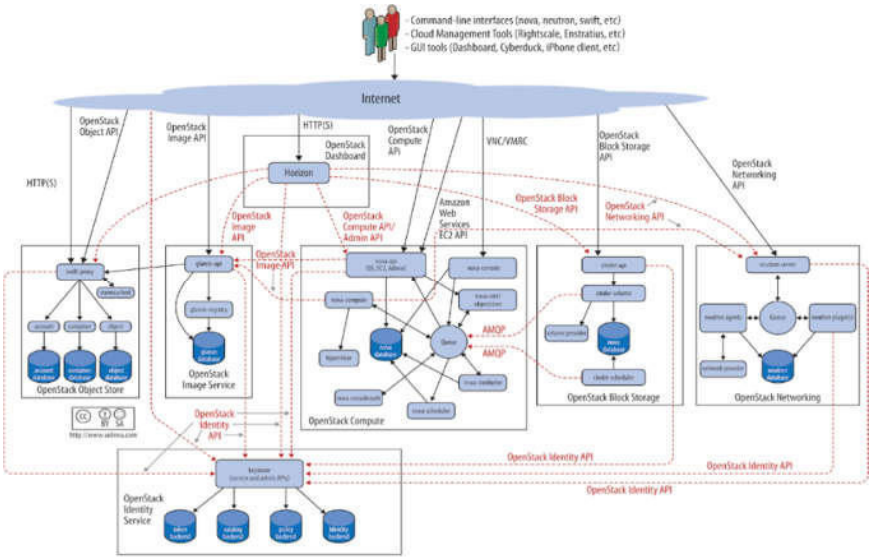
2. Infrastruktura obliczeniowa

2.1. Przegląd istniejących rozwiązań

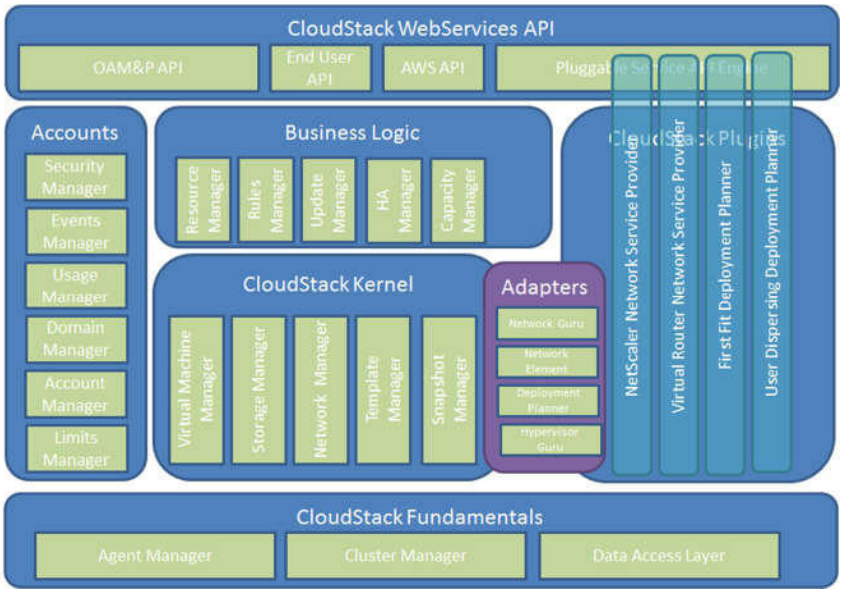
W laboratoriach i salach dydaktycznych z zasady zainstalowane już jest specjalistyczne oprogramowanie do wspierania badań i zajęć dydaktycznych, dlatego ingerencja w systemy powinna być minimalna lub najlepiej żadna. Należy również zachować ostrożność, aby istniejących systemów nie uszkodzić. Dostępne dystrybucje systemu Linux, specjalizowane w dziedzinie obliczeń rozproszonych czy klastrów [2], [3], jako niszowe, bywają rzadko aktualizowane i mogą być wyposażone w przestarzałe narzędzia. Postanowiono więc przygotować klaster obliczeniowy przy użyciu narzędzi dostępnych w aktualnych, nowych dystrybucjach Linuxa takich jak Debian, Fedora czy Ubuntu. Zastosowanie aktualnych dystrybucji udostępniają nowe wersje bibliotek i środowisk wykonawczych, które można wykorzystać do własnych obliczeń, jak np. Python, Ruby, Java lub nawet, jako maszyny wirtualne, całe systemy operacyjne, zarówno Windows jak i Linux. Przy nowo powstałych kompleksowych rozwiązaniach takie podejście ciągle jest zaliczane do atrakcyjnych biorąc pod uwagę wydajność [4].

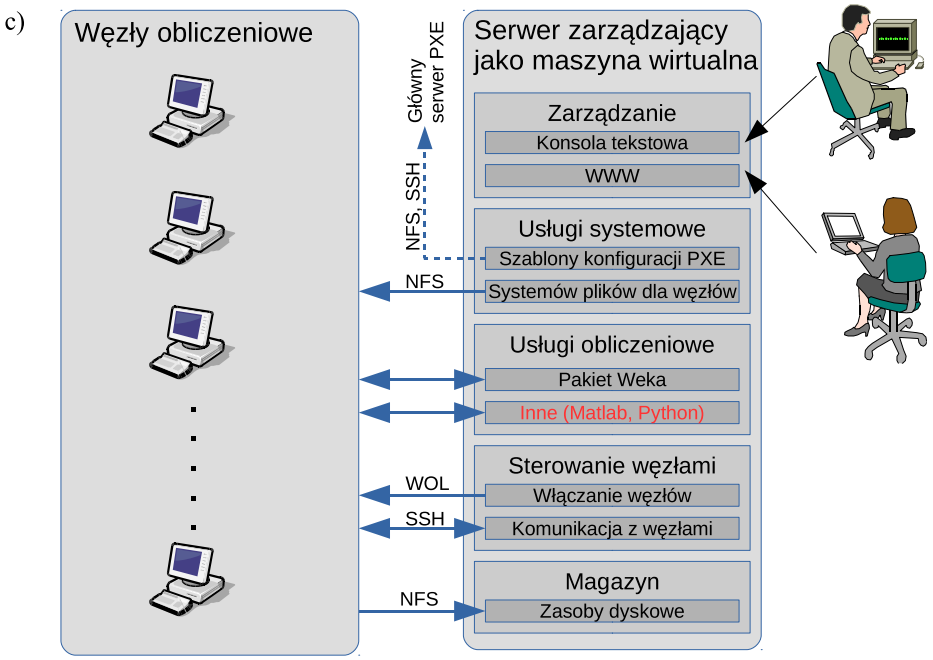
Aktualnie dostępne są zintegrowane rozwiązania open source takie jak OpenStack [5] czy CloudStack [6] oferujące funkcjonalności chmury obliczeniowej, które zapewniają usługi typu SaaS (Software as a Service), IaaS (Infrastructure as a Service), zarządzanie konfiguracją sieci oraz przydziałem zasobów obliczeniowych, bazodanowych i dyskowych. Schematy logiczne powyższych systemów oraz rozwiązanie zaproponowane przez autorów przedstawiono na poniższych rysunkach.

a)



b)





Rys. 2. Architektura logiczna dla obliczeń rozproszonych:
a) chmura OpenStack [8], b) CloudStack WebServices API [7],
c) rozwiązanie opisane w pracy

OpenStack oraz CloudStack są jednak rozwiązaniami o skomplikowanej architekturze, oferującymi szeroką możliwość konfiguracji oraz rozbudowane API, wdrażane często przez *duże organizacje*, takie jak Verizon, Apple, Autodesk, CERN, Volkswagen Group. Istotną cechą jest, aby systemy te instalować *na dedykowanych środowiskach* sprzętowych, aby w przypadku wystąpienia błędu konfiguracji czy problemu w aktualizacji nie spowodować nieprawidłowej pracy innych systemów z którymi sprzęt byłby współdzielony. Z tego powodu autorzy przyjęli własne rozwiązanie, które w niewielki sposób ingeruje w środowisko istniejące wcześniej, skonfigurowane do różnych zadań i działające produkcyjnie. Praca z takim środowiskiem wymaga dużej ostrożności, aby nie zakłócić jego funkcjonowania i nie doprowadzić do utraty danych. Jedynym punktem wspólnym konfiguracji jest praktycznie tylko dostęp serwera zarządzającego do już istniejącej konfiguracji PXE (Preboot Execution Environment).

2.2. Rola WOL i PXE

Węzły obliczeniowe uruchamiane za pomocą WOL (Wake on LAN) i PXE w żaden sposób nie modyfikują istniejących już wcześniej na dyskach systemów operacyjnych przygotowanych i używanych do dydaktyki oraz badań naukowych, a jedyną ewentualną

zmianą jest ustawienie w BIOS możliwości włączenia komputera poprzez WOL oraz domyślnego startu systemu operacyjnego poprzez PXE. Jeśli jako węzły obliczeniowe używane są komputery z wcześniej zainstalowanym na twardym dysku systemem operacyjnym, korzystna bywa także funkcja WOLtoPXE dostępna najczęściej w komputerach wiodących producentów (np. HP, Dell). Niekiedy funkcja ta występuje pod inną nazwą (np. Remote Wakeup Boot Sequence – Remote Server, Remote Wakeup – Boot to NIC), dlatego jej dostępność należy sprawdzić w dokumentacji używanego komputera. Włączenie jej powoduje, że komputer aktywowany standardowym przyciskiem uruchamia się przy pomocy domyślnej procedury ustawionej w BIOS, natomiast włączony poprzez pakiet WOL uruchamia system operacyjny poprzez PXE. Nie jest to jednak funkcjonalność konieczna do użycia komputera laboratoryjnego jako węzła obliczeniowego. W przypadku braku funkcji WOLtoPXE należy komputer ustawić w tryb domyślnego startu za pomocą PXE, natomiast w domyślnej konfiguracji PXE (plik o nazwie default) należy ustawić start z dysku lokalnego. Jeśli komputer ma zostać użyty jako węzeł obliczeniowy, serwer zarządzający klastrem i obliczeniami generuje dla niego specyficzną konfigurację PXE przypisaną do numeru MAC karty sieciowej, która zastępuje konfigurację domyślną. Nowa konfiguracja PXE umieszczana jest w pliku o specyficznej nazwie w formacie **01**-88-99-aa-bb-cc-dd [9], gdzie 88-99-aa-bb-cc-dd jest adresem MAC karty sieciowej węzła obliczeniowego. Po uruchomieniu węzła serwer zarządzający usuwa z katalogu konfiguracji PXE plik 01-88-99-aa-bb-cc-dd, co powoduje powrót do domyślnej konfiguracji.

2.3. Zaproponowane rozwiązanie

Ze względu na prostotę wykorzystano architekturę klastra Beowulf [10] z pojedynczym węzłem zarządzającym. Rozwiązanie dostosowano do istniejącej w jednostce naukowej infrastruktury sieciowej i komputerowej, praktycznie nie ingerując w konfigurację sprzętową i programową. Węzeł zarządzający uruchamia węzły obliczeniowe za pomocą technologii WOL, które z kolei poprzez PXE aktywują swoje systemy operacyjne. Systemy operacyjne węzłów obliczeniowych działają wyłącznie w pamięci RAM i nie wymagają dostępu do lokalnych zasobów dyskowych, a dzięki zastosowaniu AUFS (Advanced Multi Layered Unification FileSystem) już podczas pracy systemu pozwalają na modyfikacje poprzez doinstalowanie nowego oprogramowania koniecznego do obliczeń. Węzły obliczeniowe mogą być uruchamiane ad-hoc, w czasie, kiedy komputery nie są wykorzystywane do zajęć dydaktycznych lub badań naukowych (np. w nocy lub w weekend). Po zakończeniu obliczeń i restarcie, komputer ponownie staje się zwykłym komputerem laboratoryjnym.

3. Istniejące środowisko oraz projekt rozwiązania

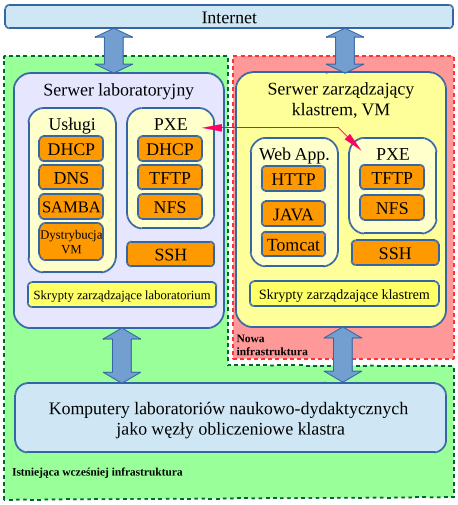
Przyjęto, że istnieje dostęp administracyjny do następującego środowiska sprzętowo-programowego:

- serwer usługowy dla laboratoriów dydaktycznych oparty o system operacyjny Linux, zapewniający takie usługi, jak: DNS, DHCP, PXE,

- dostępnych jest kilka zadaniowych systemów operacyjnych, Samba, SSH, ponadto NFS wraz z repozytorium maszyn wirtualnych,
- stacje robocze w laboratoriach, które pracują w zależności od potrzeb pod kontrolą systemu operacyjnego Windows, uruchamianego z dysku lokalnego lub systemu Linux, uruchamianego przez PXE, w zależności od wymaganych zadań (nauka baz danych, symulacja sieci komputerowych, gospodarz maszyn wirtualnych),
- serwer-gospodarz maszyn wirtualnych, mający dostęp do sieci laboratoryjnych,
- stacje robocze w sieci LAN z dostępem do serwera usługowego oraz serwera-gospodarza maszyn wirtualnych.

Do osiągnięcia wyznaczonych celów przy minimalnej ingerencji w istniejącą infrastrukturę, konieczne było wykonanie następujących prac:

- instalacja serwera zarządzającego klastrem, jako maszyny wirtualnej,
- umożliwienie dostępu do serwera z Internetu,
- umożliwienie bezpośredniego dostępu do serwera dla komputerów laboratoryjnych,
- konfiguracja środowiska PXE na nowym serwerze i integracja z istniejącą infrastrukturą,
- instalacja off-line systemów operacyjnych węzłów obliczeniowych klastra uruchamianych poprzez PXE,
- przygotowanie skryptów zarządzających klastrem,
- przygotowanie i uruchomienie aplikacji webowej umożliwiające kontrolę klastra.



Rys. 3. Schemat logiczny rozwiązania

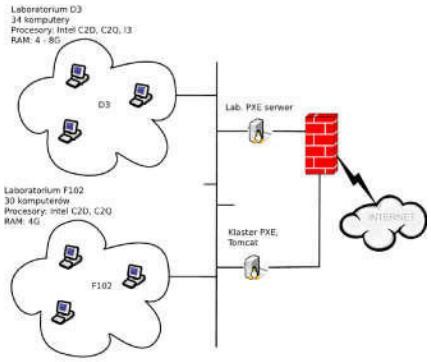
Jak pokazano na rys. 3, serwer pełniący rolę węzła zarządzającego klastrem praktycznie nie ingeruje w pierwotną konfigurację węzłów obliczeniowych. Integracji z istniejącą

infrastrukturą wymaga jedynie połączenie konfiguracji PXE istniejącego serwera usługowego dla laboratoriów z nowym serwerem zarządzającym klastrem.

4. Wdrożenie systemu obliczeń rozproszonych

4.1. Scenariusz

Zgodnie z rys. 3, w jednostce gdzie zrealizowano proponowane rozwiązanie, dostępne są dwa laboratoria naukowo-dydaktyczne (nazwane D3 i F102) wyposażone razem w 64 komputery klasy PC, z procesorami Intel Core2Duo, Core2Quad, i3, od 4 do 8 GB RAM oraz kartami sieciowymi o szybkości transmisji 1Gbps.



Rys. 4. Schemat rzeczywistego wdrożonego rozwiązania

W sumie, dostępnych jest 150 rdzeni procesora i 280 GB pamięci RAM, co daje niemałe zasoby mocy obliczeniowej. W sieci istnieje także infrastruktura PXE udostępniająca systemy operacyjne na potrzeby zajęć dydaktycznych. Komputery laboratoryjne wykorzystano jako węzły obliczeniowe.
Obliczenia mają być prowadzone w czasie, kiedy nie odbywają się zajęcia dydaktyczne, dlatego węzeł (serwer) zarządzający klastrem obliczeniowym powinien być dostępny z Internetu, aby zdalnie sterować obliczeniami. Jako oprogramowanie obliczeniowe użyto popularnego w środowisku naukowym pakietu Weka [10], wymagającego maszyny wirtualnej Java. W opisanej konfiguracji używane adresy IP to:

- adres i maska sieci laboratoryjnej: 10.10.20.0, 255.255.255.0,
- adres routera i istniejącego serwera PXE: 10.10.20.1,
- adres węzła zarządzającego klastrem: 10.10.20.20.

4.2. Istniejąca konfiguracja

Istniejąca infrastruktura PXE oparta jest o serwis dnsmasq, będący lekkim, szybkim i łatwym w konfiguracji serwerem integrującym usługi cache DNS, DHCP oraz TFTP, przez co szczególnie nadaje się do zastosowania w sieciach laboratoryjnych [11]. Włączenie TFTP na potrzeby konfiguracji PXE w dnsmasq jest bardzo proste, zajmując

3 linijki, odpowiadające za włączenie TFTP, wskazanie katalogu domowego TFTP oraz wskazanie domyślnego pliku bootloadera PXE:

```
enable-tftp
tftp-root=/srv/tftpboot
dhcp-boot=pxelinux.0
```

Każdy komputer w laboratorium ma także na stałe przypisaną nazwę i adres IP do adresu MAC. Przykładowo, dla komputerów F102-30 i D3-32 konfiguracja dnsmasq wygląda następująco:

```
dhcp-host=00:22:64:15:68:BC,F102-30,10.10.20.130,net:F102
dhcp-host=74:D4:35:C2:A2:73,D3-32,10.10.20.172,net:D3
```

Na istniejącym serwerze pliki dla PXE rezydują w miejscach:

- /srv/tftpboot – jądra i pliki rozruchowe initrd,
- /srv/nfsroot – pliki systemów operacyjnych, uruchamianych przez sieć,
- /srv/tftpboot/pxelinux.cfg – pliki konfiguracyjne menu PXE.

4.3. Modyfikacje istniejącej konfiguracji

W celu włączenia nowego serwera PXE do istniejącej infrastruktury oraz umożliwienie zarządzania konfiguracją poprzez nowy serwer, należy przeprowadzić kilka modyfikacji, tzn.:

- dodać do istniejącego menu możliwość przełączenia konfiguracji na inny serwer przez zmianę w pliku /srv/tftpboot/pxelinux.cfg/default,
- utworzyć w katalogu /srv/tftpboot/pxelinux.cfg/ nowy plik z konfiguracją automatycznie przekazującą sterowanie PXE do nowego serwera,
- umożliwić zarządzanie konfiguracją PXE dla węzła zarządzającego. Można to osiągnąć za pomocą NFS poprzez udostępnienie w trybie do zapisu katalogu /srv/tftpboot/pxelinux.cfg/ z serwera głównego.

4.4. Serwer pełniący funkcję węzła zarządzającego klasterem

Węzeł zarządzający zainstalowano na maszynie wirtualnej, w której jako system operacyjny pracuje Linux Debian Jessie. Na serwerze powstało środowisko PXE dla węzłów obliczeniowych, skrypty zarządzające klastrem, aplikacja webowa do zdalnego zarządzania klastrem oraz system obliczeniowy Weka, funkcjonujący w trybie obliczeń rozproszonych. Kolejne prace wyglądają następująco:

1) utworzenie maszyny wirtualnej: 2 rdzenie procesora, 2 GB RAM, 2 karty sieciowe, 20 GB pamięci dyskowej,
2) jedna z kart sieciowych włączona do sieci laboratoryjnej, druga do sieci zewnętrznej,
3) instalacja najnowszej dystrybucji Linux Debian z pakietami: SSH, NFS klient i serwer, Java, Tomcat, sudo,
4) zamontowanie w katalogu /mnt/pxelinux.cfg katalogu konfiguracyjnego PXE z serwera głównego,
5) utworzenie i eksport przez NFS katalogu /srv/nfsroot,

6) uruchomienie PXE oraz utworzenie w katalogu /srv/nfsroot/weka bazowego systemu operacyjnego węzła obliczeniowego,
7) w konfiguracji PXE domyślnie uruchamiane są systemy klastra obliczeniowego,
8) utworzenie plików konfiguracyjnych i skryptów zarządzających klastrem:
   - plik machines – zawiera numery i dane sieciowe komputerów, które mogą być używane jako węzły obliczeniowe klastra. W każdej linii znajduje się informacja o innym komputerze w formacie nr,MAC,nazwa,IP,
   - skrypty pomocnicze v-get-machine-ip, v-get-machine-name, v-get-machine-mac przyjmujące jako parametr numer maszyny. Na podstawie danych z pliku machines zwracają odpowiednio IP, nazwę lub MAC dla żądanej maszyny,
   - skrypt v-turn-on,który włącza węzły obliczeniowe wykorzystując polecenie etherwake. Dodatkowo, skrypt ten tworzy także plik konfiguracyjny dla PXE o nazwie 01-MAC, gdzie MAC jest adresem sprzętowym maszyny docelowej. Plik ten jest przetwarzany zamiast pliku default,
   - skrypt v-run-cmd przyjmuje jako pierwszy parametr numer maszyny i poprzez SSH wykonuje polecenie podane jako drugi parametr. Skrypt ten wykorzystywany jest przez inne polecenia i ma postać:

```
#!/bin/bash
export PATH=$PATH:/usr/local/bin
nr=""
cmd=""
if [ $# -gt 1 ]; then
    nr=$1
    cmd=$2
fi
if [ -z "$nr" ] || [ -z "$cmd" ]; then
    fname=`basename $0`
    echo "Wywołanie np: $fname 3 \"lshw -short\""
    exit 0
fi
opts="-o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no"
komp=`v-get-machine-ip $nr`
if [ -n $mac ]; then
    sudo ssh $opts root@$komp "$cmd"
else
    echo "Computer not found for machine number:
$nr"
fi
```

   - skrypt v-turn-off wyłącza podaną jako parametr maszynę, wykorzystując wewnątrz polecenie v-run-cmd, np. v-run-cmd 4 ‘poweroff’, gdzie 4 jest numerem maszyny, natomiast poweroff komendą do wykonania,
   - skrypty pomocnicze: v-check-on, v-check-weka, v-get-number-of-cores, v-get-hardware-info,
   - skrypt clean-pxe-configs uruchamiany cyklicznie (np. co 10 minut) usuwa utworzone przez skrypt v-turn-on pliki konfiguracyjne PXE starsze niż 5 minut,

9) utworzenie grupy użytkowników staff, do której dodano użytkowników mających prawa zarządzania klastrem,
10) konfiguracja mechanizmu sudo nadającego prawa zarządzania klastrem grupie staff. Do pliku /etc/sudoers należy dopisać linię: %staff ALL=NOPASSWD: ALL,
11) instalacja oprogramowania Weka w trybie serwera obliczeń rozproszonych,
12) instalacja aplikacji webowej na serwerze Tomcat ułatwiającej zarządzanie klastrem,
13) aplikacja webowa zarządza klastrem za pomocą opisanych wcześniej skryptów,
14) aby aplikacja mogła wywoływać skrypty zarządzające klastrem, do grupy *staff* należy dodać użytkownika tomcat8, na którego prawach pracuje serwer aplikacji.

4.5. Węzeł obliczeniowy

Węzeł obliczeniowy jest fizycznym komputerem PC skonfigurowanym tak, aby możliwe było jego włączenie funkcją WOL. Domyślnie system operacyjny jest uruchamiany z serwera za pomocą PXE. Po uruchomieniu systemu operacyjnego, automatycznie startuje oprogramowanie obliczeniowe Weka i rejestruje się na serwerze. Węzeł obliczeniowy jest tak skonfigurowany, aby można było na nim poprzez SSH wykonywać polecenia z węzła zarządzającego. Szczegóły konfiguracji systemu operacyjnego węzła obliczeniowego są następujące:

1) w katalogu serwera /srv/nfsroot/weka instalacja systemu operacyjnego Linux Debian [12],
2) wykonując polecenie chroot /srv/nfsroot/weka dodanie potrzebnych pakietów:
   - SSH – umożliwia sterowanie węzłem,
   - Java – uruchamia oprogramowanie Weka,
   - dodanie użytkownika weka i w jego katalogu domowym umieszczenie plików binarnych oprogramowania Weka,
   - utworzenie skryptu weka.sh w katalogu /usr/local/bin w celu uruchomienia oprogramowania Weka:

```
#!/bin/bash
HOST=$1
MASTER="10.10.20.20:8085"
COR=`nproc`
if [ -z "$CORES" ] || [ $CORES -lt 1 ]; then
CORES=1; fi
echo "WEKA run at $HOST, master $MASTER $COR
cores"
cd /home/weka
su weka -c "java -Djava.awt.headless=true -cp
weka.jar weka.Run WekaServer -host $HOST -master
$MASTER -slots $CORES"
```

   - ustawienia nazwy węzła obliczeniowego na podstawie DNS oraz uruchomienie w tle skryptu weha.sh poprzez modyfikację pliku /etc/rc.local.

System przygotowany w przedstawiony sposób jest gotowy do przyjmowania zadań obliczeniowych, które zleca się z poziomu aplikacji Weka GUI, według schematu przedstawionego na rys. 4.

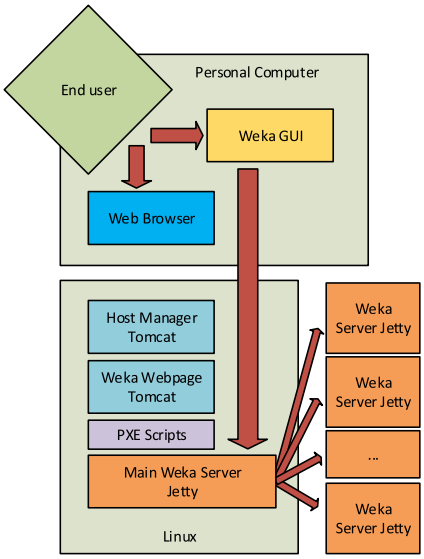
5. Eksperyment

5.1. Założenia

Przygotowane środowisko obliczeniowe przetestowano używając zadania postawionego w [8], które dotyczy oszacowania ceny nieruchomości na podstawie kilkudziesięciu atrybutów. Można to zadanie potraktować jako rozszerzoną wersję często cytowanego zbioru „Boston Housing" [13], służącą zwykle do testowania algorytmów regresji. W analizowanym zbiorze, każdy z 2930 domów, opisany jest 79 atrybutami, wśród których występują 23 nominalne, 23 porządkowe, 14 dyskretnych oraz 20 ciągłych. Należy opracować model, który będzie w stanie trafnie przewidywać cenę nieruchomości.

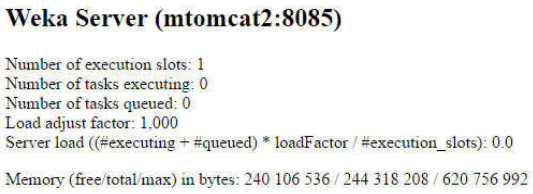
W omawianym eksperymencie do przeprowadzenia obliczeń zostało użyte oprogramowanie Weka 3.7.13 [14], wyposażone w różne algorytmy uczenia maszynowego. Do przeprowadzenia testów wybrano sztuczną sieć neuronową (uczoną metodą wstecznej propagacji błędów) oraz algorytm regresji liniowej. Na potrzeby weryfikacji przydatności opracowanego klastra, przeprowadzono analizę porównawczą czasu obliczeń w środowisku nierozproszonym, oraz z wieloma maszynami połączonymi w klaster obliczeniowy.

Zastosowany pakiet Weka pozwala na dodanie bibliotek umożliwiających wykonywanie obliczeń rozproszonych z użyciem dedykowanego serwera. Proponowaną architekturę prezentuje rys 4.



Rys. 5. Komponenty systemu obliczeniowego

Poszczególne serwery obliczeniowe Weka przez protokół HTTP udostępniają informację o statusie zleconych obliczeń w postaci uproszczonego widoku HTML, co przedstawiono na rys. 6.



Rys. 6. Ekran stanu obliczeń zleconych na serwerze

5.2. Rezultaty

Ze względu na długi czas obliczeń do eksperymentu wybrano tylko 1460 próbek ze zbioru „Ames, Iowa". Próbki zostały wybrane losowo z kompletnego zbioru. Obliczenia były wykonywane na wszystkich 79 atrybutach. Eksperyment w środowisku lokalnym był wykonany na komputerze przenośnym wyposażonym w procesor i7 5600U o częstotliwości 3.2 GHz, natomiast komputery w klastrze obliczeniowym, to jednostki stacjonarne z uczelnianego laboratorium o częstotliwości od 2 do 3 GHz wyposażone w 2 lub 4 rdzenie. Weka podczas obliczeń na komputerze lokalnym, w swoim standardowym module „Explorer" wykorzystuje jeden proces do wyliczenia modelu perceptronu wielowarstwowego. Z tego też tytułu czasy wykonanych obliczeń na komputerze lokalnym rosną w sposób proporcjonalny do liczby części podczas wykonywania testu krzyżowego. Przez czas obliczeń rozumie się sumę czasów potrzebnych na podział zbioru na ilość części walidacji krzyżowej [15], zbudowanie modelu dla każdej z dziesięciu części oraz przetestowanie na wydzielonym zbiorze testowym. W Tabeli 1 zaprezentowano porównanie czasów obliczeń pomiędzy jednym komputerem lokalnym, a rozproszonym środowiskiem obliczeniowym. Z uwagi na zastosowanie we wszystkich eksperymentach tej samej wartości parametru „seed", odpowiedzialnego w Wece za generowanie liczb pseudolosowych, podział zbioru na części oraz wyniki pracy algorytmu będą identyczne dla wszystkich powtórzeń.

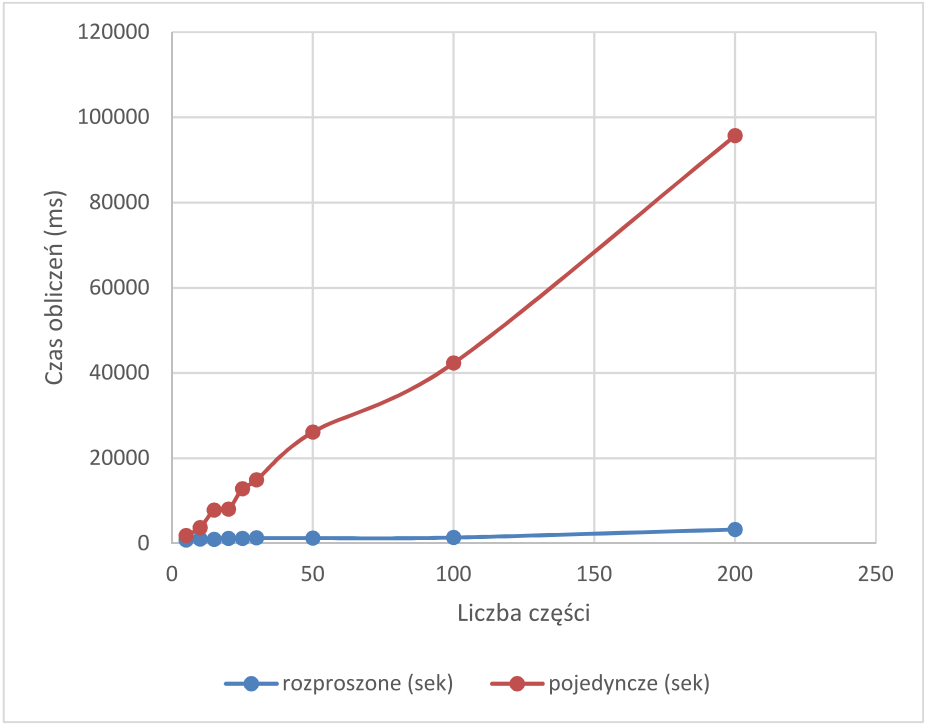
Tabela 1. Zestawienie czasów wykonania obliczeń walidacji krzyżowej dla algorytmu perceptronu wielowarstwowego, w zależności od ilości części dla walidacji krzyżowej oraz środowiska obliczeniowego

Liczba części dla walidacji krzyżowej	Czas obliczeń na lokalnym komputerze (hh:mm)	Czas obliczeń w klastrze (hh:mm)
5	00:31	00:13
10	01:02	00:17
15	02:09	00:16
20	02:14	00:19
25	03:33	00:20
30	04:08	00:21
50	07:15	00:21
100	11:45	00:23
200	26:34	00:55

Widoczne jest wielokrotne skrócenie czasu wykonania, przy zastosowaniu klastra obliczeniowego. Budowanie i testowanie modelu sieci neuronowej w środowisku rozproszonym pozwala na swobodne modelowanie eksperymentu pomimo dużej

złożoności. Przeprowadzenie obliczeń przy podziale na 200 części zajmuje jedną maszynę na ponad dobę, podczas gdy w klastrze można wykonać te obliczenia w przeciągu jednej godziny.

Kolejnym etapem eksperymentu było przeprowadzenie analogicznych obliczeń z zastosowaniem algorytmu regresji liniowej. Został użyty algorytm dostępny w pakiecie Weka, który korzysta z kryterium Akaikego [13] do selekcji modelu. W przypadku regresji liniowej czas budowania modelu jest znacznie krótszy od perceptronu i dla rozpatrywanych danych wynosił od 4-5 sekund. Zależność wykonania czasu obliczeń na lokalnym komputerze oraz w klastrze została zaprezentowana w Tabeli 2. Różnica pomiędzy czasami wykonania na jednym komputerze oraz w klastrze dla regresji liniowej (Tabela 2) jest mniejsza w porównaniu z algorytmem perceptronu wielowarstwowego (Tabela 1).

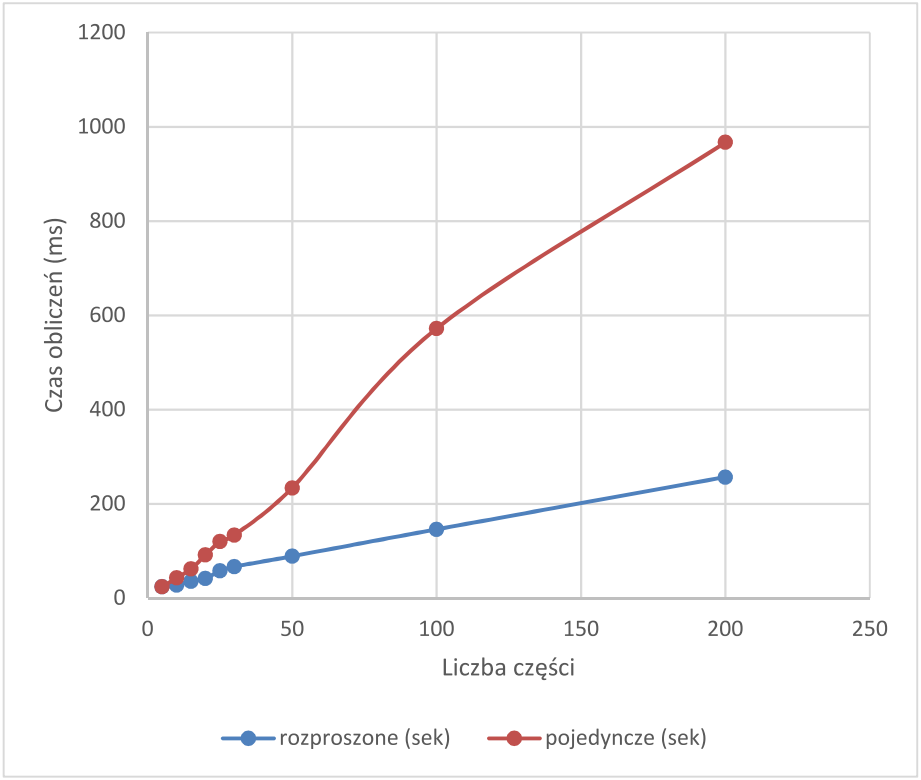


Wykres. 1. Zestawienie czasu obliczeń regresji z użyciem perceptronu na komputerze lokalnym oraz klastrze dla różnej ilości części walidacji krzyżowej

Tabela 2. Zestawienie czasów wykonania obliczeń walidacji krzyżowej dla algorytmu perceptronu wielowarstwowego, w zależności od ilości części dla walidacji krzyżowej oraz środowiska obliczeniowego

Liczba części dla walidacji krzyżowej	Czas obliczeń na lokalnym komputerze (hh:mm:ss)	Czas obliczeń w klastrze (hh:mm:ss)
5	00:00:24	00:00:24
10	00:00:43	00:00:28
15	00:01:02	00:00:36
20	00:01:32	00:00:42
25	00:02:00	00:00:58
30	00:02:14	00:01:07
50	00:03:54	00:01:29
100	00:09:32	00:02:26
200	00:16:07	00:04:17

Wydatek czasowy na transfer danych do maszyn obliczeniowych w stosunku do czasu samych obliczeń jest tu zauważalny i pomniejsza korzyści z przeprowadzania eksperymentu w klastrze.



Wykres. 2. Zestawienie czasu obliczeń regresji liniowej na komputerze lokalnym oraz klastrze dla różnej ilości części walidacji krzyżowej

Zintegrowano skrypty serwera Weka oraz maszyn obliczeniowych tak, aby zapewnić wygodny zdalny interfejs użytkownika. Zważywszy na bogate zasoby oraz jakość algorytmów w pakiecie Weka, takie rozwiązanie może być atrakcyjne dla środowisk akademickich. Pewną niedogodnością jest konieczność oczekiwania na rezultaty pracy przy włączonej aplikacji Weka GUI. Zamknięcie jej w trakcie wykonywania zleconych zadań powoduje, że nie uda się otrzymać rezultatów. Niedogodnością jest również brak mechanizmów przełączania obliczeń zleconych do węzła, który ulegnie uszkodzeniu. W przypadku, gdy którykolwiek z węzłów nie odpowiada, całość zleconych obliczeń jest zakończona niepowodzeniem. Wymienione problemy są związane jedynie z zastosowanym oprogramowaniem pakietu Weka, nie rzutują jednak na całość opracowanego rozwiązania.

W przeprowadzonym eksperymencie wykazano sprawność działania i korzyści wynikające z zastosowania klastra w przypadkach wielu równoległych obliczeń o znacznej złożoności. Jeżeli rozpatrzylibyśmy procesy równoległe, wymagające częstej wymiany informacji pomiędzy poszczególnymi węzłami obliczeniowymi, to proponowane rozwiązanie przegra ze specjalizowanymi, wieloprocesorowymi jednostkami obliczeniowymi. Jeżeli jednak rozważymy tak niezależne zadania jakimi jest budowanie modeli sieci neuronowych dla wielu różnych parametrów, a następnie weryfikacja ich jakości z zastosowaniem walidacji krzyżowej, to proponowane rozwiązanie staje się bardzo konkurencyjne.

6. Podsumowanie

Zaproponowane rozwiązanie dobrze się sprawdza pod względem funkcjonalnym i wydajnościowym. Jego znaczącą zaletą jest minimalna ingerencja w istniejącą infrastrukturę laboratoryjną. Dzięki swej prostocie i uniwersalności, rozwiązanie to może być atrakcyjne dla jednostek akademickich. Dodatkową jego zaletą jest możliwość łatwego uruchamiania eksperymentów obliczeniowych przygotowanych własnoręcznie, bez konieczności posługiwania się specjalistycznym oprogramowaniem do rozpraszania obliczeń. Zastosowany sposób wirtualizacji i możliwość modyfikacji uruchamianego środowiska, nawet podczas jego działania, umożliwia swobodne projektowanie wyrafinowanych systemów do obliczeń rozproszonych.

W opisanym rozwiązaniu wykorzystano pakiet Weka, jednak tylko w charakterze przykładu. Należy zaznaczyć, że równie łatwo można użyć jednego z wielu systemów umożliwiających pracę w klastrze obliczeniowym, takich jak Matlab [16], R [17], Statistica [18] czy TensorFlow [19]. W następnej kolejności rozważane jest opracowanie zintegrowanego, gotowego do pobrania pakietu instalacyjnego proponowanego rozwiązania oraz rozszerzenie panelu administracyjnego o dodatkowe funkcjonalności. Planuje się również zastosowanie przygotowanego klastra do uruchamiania obliczeń w zakresie Deep Learning, z użyciem bibliotek TensorFlow [17].

Bibliografia

[1] K. M. Tolle, D. S. W. Tansley, and A. J. G. Hey, "The Fourth Paradigm: Data-Intensive Scientific Discovery [Point of View]," Proc. IEEE, vol. 99, no. 8, pp. 1334–1337, Sierpie 2011.

[2] "KestrelHPC download | SourceForge.net."
[Online]. Available: https://sourceforge.net/projects/kestrelhpc/. [Accessed: 11-Apr-2017].

[3] "Pelican Hpc | Computer Cluster | Booting," Scribd.
[Online]. Available: https://pl.scribd.com/document/103716505/Pelican-Hpc. [Accessed: 11-Apr-2017].

[4] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita, "Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf," Procedia Comput. Sci., vol. 53, pp. 121–130, Stycze 2015.

[5] "Open source software for creating private and public clouds.," OpenStack. [Online]. Available: https://www.openstack.org/. [Accessed: 14-Jul-2017].

[6] "Apache Cloudstack," Apache Cloudstack. [Online]. Available: https://cloudstack.apache.org/. [Accessed: 14-Jul-2017].

[7] "Development 101 - Apache Cloudstack - Apache Software Foundation." [Online]. Available: https://cwiki.apache.org/confluence/display/CLOUDSTACK/Development+101. [Accessed: 14-Jul-2017].

[8] "OpenStack Docs: Design." [Online]. Available: https://docs.openstack.org/arch-design/design.html. [Accessed: 14-Jul-2017].

[9] "PXELINUX - Syslinux Wiki." [Online].
Available: http://www.syslinux.org/wiki/index.php?title=PXELINUX. [Accessed: 14-Jul-2017].

[10] T. Sterling, D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer, "Beowulf: A Parallel Workstation For Scientific Computation," in In Proceedings of the 24th International Conference on Parallel Processing, 1995, pp. 11–14.

[11] "Dnsmasq - network services for small networks." [Online].
Available: http://www.thekelleys.org.uk/dnsmasq/doc.html. [Accessed: 11-Apr-2017].

[12] "Diskless Debian Linux booting via dhcp/pxe/nfs/tftp/aufs | Debian Addict." [Online]. Available: http://debianaddict.com/2012/06/19/diskless-debian-linux-booting-via-dhcppxenfstftp/. [Accessed: 11-Apr-2017].

[13] D. Harrison and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," J. Environ. Econ. Manag., vol. 5, no. 1, pp. 81–102, Mar. 1978.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explor Newsl, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[15] J.-H. Kim, "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap," Comput. Stat. Data Anal., vol. 53, no. 11, pp. 3735–3745, Sep. 2009.

[16] "Parallel Computing Toolbox - MATLAB." [Online].
Available: https://www.mathworks.com/products/parallel-computing.html. [Accessed: 11-Apr-2017].

[17] "Omówienie rozwiązania R Server — analizy danych oparte na języku R | Microsoft." [Online]. Available: https://www.microsoft.com/pl-pl/cloud-platform/r-server. [Accessed: 11-Apr-2017].

[18] "Products — Statistica." [Online]. Available: http://statistica.io/products/. [Accessed: 11-Apr-2017].

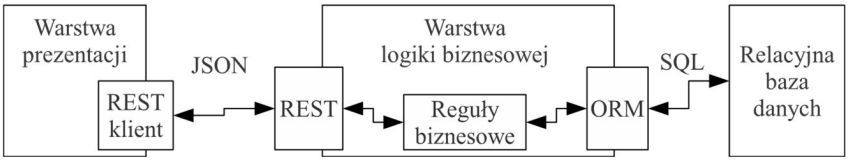
[19] "TensorFlow." [Online]. Available: https://www.tensorflow.org/. [Accessed: 11-Apr-2017].

Rozdział 4

Usługa ORM typu REST – model i implementacja

1. Wstęp

Współczesne systemy informatyczne charakteryzują się architekturą warstwową[1], gdzie warstwą najwyższą jest interfejs użytkownika, który komunikuje się z warstwą aplikacji implementującą scenariusze przypadków użycia. Dane przetwarzane przez system składowane są w warstwie przechowywania danych, do której dostęp ma część aplikacyjna systemu. Każda z warstw zwykle tworzona jest z użyciem innej technologii informatycznej i stanowi niezależny moduł. Moduły komunikują się korzystając ze specjalizowanych języków (np. SQL, JSON, XML) i protokołów komunikacyjnych (rys. 1). Klasyczne podejście do budowy systemu zakłada, że struktury danych przetwarzanych przez każdą warstwę są ustalane na etapie analizy i projektowania. Na etapie implementacji oprogramowania struktury te opisywane są w językach programowania dedykowanych dla poszczególnych modułów.

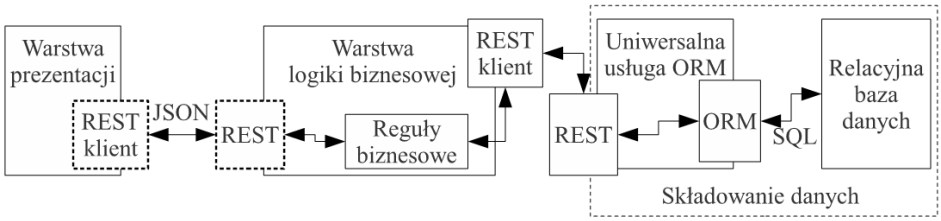


Rys. 1. Architektura trójwarstwowego systemu informatycznego

Proces tworzenia struktur służących do przechowywania danych jest obecnie zautomatyzowany. Używane są generatory kodu, które na podstawie zawartości bazy danych generują komponenty encyjne używane w warstwie aplikacji i klasy dostępu do danych, zawierające usługi realizujące podstawowe operacje typu CRUD. Generatory te wykorzystują technologie odwzorowania relacyjno obiektowego[6].

W związku z rozwojem technologii tworzenia warstwy prezentacji z użyciem języków skryptowych, proces tworzenia struktur danych wspomniany w poprzednim akapicie uległ modyfikacji. Przegląd procesu tworzenia warstwy prezentacji z użyciem języka JavaScript (framework AngularJS [5], jQuery [12]) pozwala postawić tezę, że w komponentach tworzonych w tej technologii rzadko korzysta się z opisu danych za pomocą klas. Zamiast tego, stosowane są tablice asocjacyjne o niejawnej strukturze. Oczywiście istnieje możliwość zdefiniowania zbioru klasycznych obiektów

transferowych w postaci klas, lecz wiąże się to z nakładem pracy i zwykle proces ten jest pomijany.



Rys. 2. Rola sieciowej usługi ORM

Skoro praktyka inżynierska pokazuje, że warstwa prezentacji działa poprawnie bez ściśle zdefiniowanych struktur danych, autorzy tej pracy podjęli próbę uogólnienia zadania budowy modułu dostępu do danych. Cele pracy sformułowane zostały następująco:

- opracowanie abstrakcyjnego modelu wybranych elementów relacyjnej bazy danych,
- zaprojektowanie i implementacja usługi sieciowej typu REST [7], która:
  - utworzy abstrakcyjny model danych dla wskazanej relacyjnej bazy danych,
  - wygeneruje komponenty encyjne i klasy kontrolerów Spring z podstawowymi usługami typu CRUD,
  - skompiluje wygenerowany kod i udostępni wygenerowane usługi w postaci usługi sieciowej REST z formatem danych JSON.

Jako technologię implementacji wybrano język Java [4], framework Spring [2] z biblioteką ORM Hibernate [3]. Miejsce opracowanego komponentu w architekturze aplikacji rozproszonej pokazuje rys. 2. Razem z relacyjną bazą danych pełni on rolę warstwy składowania danych. Ze względu na to, że współczesne frameworki służące do tworzenia warstwy prezentacji są w stanie zawierać kod implementujący logikę aplikacji, możliwe jest wdrożenie warstwy logiki biznesowej na urządzeniu klienckim. W takim przypadku komponenty REST klient i REST zaznaczone na rys. 2 przerywanymi pogrubionymi liniami są zbędne, ponieważ możliwa jest komunikacja przy pomocy mechanizmów lokalnych.

Dokonano przeglądu bibliotek służących do generowania komponentów encyjnych. W zależności od technologii informatycznej, dostępne są m.in. następujące rozwiązania:

- Java - Hibernate Tools [8],
- PHP - Doctrine 2 ORM [9],
- .Net - Entity Framework Power Tools.

Nie zdecydowano się na użycie biblioteki Hibernate Tools, ponieważ jej dokumentacja opisuje wyłącznie użycie z poziomu kreatorów osadzonych w środowiskach programistycznych. Biblioteka ta tworzy meta model danych i generuje komponenty encyjne. Ponieważ celem pracy było również wygenerowanie klas dostępu do danych, niezbędne było skorzystanie ze wspomnianego meta modelu. Brak dokumentacji uniemożliwił takie podejście, a analiza kodu źródłowego spowodowałaby przekroczenie

ram czasowych pracy. Z tego względu autorzy zdecydowali się na opracowanie własnego abstrakcyjnego modelu danych i generatora kodu.

2. Abstrakcyjny model danych

Celem tej sekcji jest wprowadzenie abstrakcyjnego modelu matematycznego, który opisuje wybrane obiekty relacyjnej bazy danych. Modelowanymi obiektami są tabele, kolumny, typy kolumn, ograniczenia integralnościowe kolumn, referencje między tabelami. Modelowane są również komponenty programowe języka Java, które odpowiadają wymienionym obiektom bazy danych, oraz funkcje, przy pomocy których dokonuje się konwersji modelu relacyjnego na model obiektowy.
Przyjmijmy następujące oznaczenia:

$C = \{c_1, c_2, ..., c_n\}$ zbiór kolumn,
$T = \{t_1, t_2, ..., t_m\}$ zbiór typów,
$B = \{b_1, b_2, ..., b_p\}$ zbiór nazw tabel,
$A = \{a_1, a_2, ..., a_l\}$ zbiór ograniczeń integralnościowych ,
$V = \{v_1, v_2, ..., v_k\}$ zbiór wartości kolumn.

Elementy $a_1 \in A$ i $a_2 \in A$ mają specjalne znaczenie, zdefiniowane równaniami (2) i (3).
Dana jest relacja łącząca w bazie danych kolumnę z typem:

$$R_{CT} \subseteq C \times T = \{(c_i, t_i) : c_i \in C \wedge t_i \in T\}$$

Ograniczeniami integralnościowymi kolumny w bazie danych nazywamy relację postaci

$$R_{CONSTR} \subseteq R_{CT} \times 2^A = \{((c_i, t_i), A_i) : (c_i, t_i,) \in R_{CT} \wedge A_i \in 2^A\}$$

Funkcja

$$F_{CONSTR} : (c,t) \to 2^A := \{((c_i, t_i), A_i) : (c_i, t_i,) \subseteq R_{CT} \in A_i \times 2^A\}$$

zwraca ograniczenia integralnościowe przypisane do pary (c,t).
Definicjami tabel w bazie danych nazywamy relacje postaci:

$$R_{TAB} \subseteq B \times R_{CONSTR} = \{(b, r_{CONSTR}) : b \in B \wedge r_{CONSTR} \in R_{CONSTR}\} \quad (1)$$

$$\overline{R}_{TAB} \subseteq B \times 2^{R_{CONSTR}} = \{(b, r_{CONSTR}) : b \in B \wedge r_{CONSTR} \in 2^{R_{CONSTR}}\}$$

Zbiorem wartości kolumny w tabeli relacyjnej bazy danych nazywamy relację:

$$R_v \subseteq R_{TAB} \times V := \{(((b, r_{CONSTR})), v_i) : (b, r_{CONSTR}) \in R_{TAB} \wedge v_i \in V\}.$$

Przyjmijmy następujące oznaczenie: jeżeli dana jest n-tka X postaci $(x_1, x_2, \ldots, x_n)$, to element nr i n-tki X oznaczamy $X[i]$, np. $X[2] = x_2$.
Nadajmy specjalne znaczenie elementowi $a_1 \in A$:

$$\forall (x, y) \in R_{TAB} : a_1 \in y[2] \Rightarrow \exists v_i, v_j \in V : \\ ((x, y), v_i) \in R_v \wedge ((x, y), v_j) \in R_v \Rightarrow v_i \neq v_j \quad (2)$$

Element ten modeluje klucz główny. Wyjaśnienie:

- $y[2]$ - zbiór ograniczeń integralnościowych kolumny,
- $a_1 \in y[2]$ - w zbiorze ograniczeń integr. jest klucz główny,
- $((x, y), v_i) \in R_v$ - kolumna y z tabeli x przyjmuje wartość v_i,
- $v_i \neq v_j$ - jeżeli mamy 2 wartości przypisane do kolumny, to są one różne.

Referencją w bazie danych nazywamy relację postaci:

$$R_{REF} \subseteq R_{TAB} \times R_{TAB} := \{(x, y) : x, y \in R_{TAB} \wedge \\ x[2][1][2] = y[2][1][2] \wedge \\ a_1 \in x[2][2] \wedge a_2 \in y[2][2]\} \quad (3)$$

Wyjaśnienie:

- $x[2][1][2]$ - typ kolumny t w krotce $(b, ((c, t), \{A\}))$
- typy kolumn złączonych referencją muszą być identyczne,
- $x[2][2]$ - ograniczenia integralnościowe $\{A\}$ w krotce $(b, ((c, t), A))$
- $a_1 \in x[2][2]$ - w ograniczeniach integralościowych kolumny z elementu x pary (x, y) jest klucz główny,
- $a_2 \in y[2][2]$ - w ograniczeniach integralościowych kolumny z elementu y pary (x, y) jest klucz obcy.

Przykład 1.
Załóżmy, że mamy następujące zbiory:

$C =$ $\{id, imie, nazwisko, ulica, numer\_domu, numer\_mieszkania, id\_osoby\}$
$T =$ $\{varchar, int16, int32\}$
$B =$ $\{dane\ osobowe, adresy\}$
$A =$ $\{primary\ key, foreign\ key, not\ null, unique\}$

Niech relacja łącząca kolumnę tabeli z typem będzie w omawianym przykładzie określona następująco:

$$R_{CT} = \{(id, int32), (imie, varchar), (nazwisko, varchar), (ulica, varchar), (numer\_domu, int16), (numer\_mieszkania, int16), (id\_osoby, int32)\}$$

Niech relacja tworząca ograniczenia integralnościowe kolumn R_{CT} z przykładu 1 będzie następująca:

$$\begin{aligned} R_{CONSTR} = {} & \{r_{CONSTR1}, r_{CONSTR1}, \ldots, r_{CONSTR7}\} = \\ & \{((id, int32), \{primary\ key\}), ((imi, varchar), \{not\ null\}), \\ & ((nazwisko, varchar), \{not\ null\}), ((ulica, varchar), \{unique, not\ null\}), \\ & ((numer\_domu, int16), \{not\ null\}), (numer\_mieszkania, int16), \\ & ((id\_osoby, int32), \{foreign\ key, not\ null\})\} \end{aligned} \quad (4)$$

Tabele w bazie danych określa następująca relacja:

$$\begin{aligned} R_{TAB} = {} & \{(dane\ osobowe, r_{CONSTR1}), (dane\ osobowe, r_{CONSTR2}), \\ & (dane\ osobowe, r_{CONSTR3}), \\ & (adresy, r_{CONSTR1}), (adresy, r_{CONSTR4}), \ldots, (adresy, r_{CONSTR7})\} \\ \overline{R}_{TAB} = {} & \{(daneosobowe, \{r_{CONSTR1}, r_{CONSTR2}, r_{CONSTR3}\}), \\ & (adresy, \{r_{CONSTR1}, r_{CONSTR4}, \ldots, r_{CONSTR7}\})\} \end{aligned}$$

Referencja jest zbiorem jednoelementowym, zdefiniowanym następująco:

$$\begin{aligned} R_{REF} = {} & \{((dane\ osobowe, ((id, int32), \{primary\ key\}), \\ & (adresy, ((id\_osoby, int32), \{foreign\ key, not\ null\})))\} \end{aligned} \quad (5)$$

Zgodnie z definicją relacji R_{REF}, mamy:

$x = (dane\ osobowe, ((id, int32), \{primary\ key\})$
$y = (adresy, ((id\_osoby, int32), \{foreign\ key, not\ null\}))$
$x[2][1][2] = int32$
$y[2][1][2] = int32 \quad x[2][2] = \{primary\ key\}$
$y[2][2] = \{foreign\ key, not\ null\}$

Relacja (5) łączy kolumnę *id_osoby* z tabeli adresy z kolumną *id* tabeli dane osobowe. Koniec przykładu 1.

Wprowadźmy następujące zbiory:

$C' = \{c'_1, c'_2, ..., c'_{n'}\}$ - zbiór pól klas, $n' \geq n$,

$T' = \{t'1, t'2, ..., t'_m\}$ - zbiór typów języka Java,

$B' = \{b'_1, b'_2, ..., b'_p\}$ - zbiór nazw klas komponentów encyjnych języka Java,

$A' = \{a'_1, a'_2, ..., a'_{l'}\}$ - adnotacje tworzące ograniczenia integralnościowe w encji, $l' \geq l$.

W zbiorze T' oznaczmy element t'_m jako kolekcja (np. typ Set). Zdefiniujmy następujące funkcje, których zadaniem jest odwzorowanie tabel bazy danych na klasy języka Java i ograniczeń integralnościowych kolumn tabel na adnotacje tworzące ograniczenia integralnościowe w komponentach encyjnych:

$$F_B : B \to B' = \{(b_i, b'_i) : b_i \in B \wedge b'_i \in B'\}, \forall b_i, b_j \in B : b_i, \neq b_j \Rightarrow F_B(b_i) \neq F_B(b_j),$$
$$F_A : A \to 2^{A'} = \{(a_i, A'_i) : a_i \in A \wedge A'_i \in 2^{A'}\}.$$

Nadajmy specjalne znaczenie elementom a'_1, a'_2 i a'_3:

$$F_A(a_2) = \{a'_1, a'_2\}, \nexists a_i \in A : F_A(a_i) = a'_3.$$

Wyjaśnienie:

- zbiór $\{a'_1, a'_2\}$ reprezentuje adnotacje @ManyToOne i @JoinColumn, służące do implementacji asocjacji o krotności "wiele - jeden",
- element a'_3 to adnotacja @OneToMany, która nie ma odwzorowania w bazie danych, jednak istnieje w modelu obiektowym.

Zdefiniujmy następujące funkcje, których zadaniem jest odwzorowanie kolumn tabel na pola klas i typów bazy danych na typy języka Java:

$$F_C : C \to C' := \{(c_i, c'_i) : c_i \in C \wedge c'_i \in C'\},$$

funkcja F_c jest różnowartościowa, tj. $\exists c_i, c_j \in C : c_i, \neq c_j \Rightarrow F_C(c_i) \neq F_C(c_j)$

$$F_T : T \to T' := \{(t_i, t'_i) : t_i \in T \wedge t'_i \in T'\},$$

funkcja F_T nie jest różnowartościowa, tj. $\exists t_i, t_j \in T : ti, \neq tj \Rightarrow F_T(t_i) = F_T(t_j)$. Założenie takie jest potrzebne, ponieważ kilka typów bazy danych może być odwzorowanych w jeden typ języka Java. Oznaczmy jako C^* podzbiór tych kolumn bazy danych, które są kluczami obcymi:

$$C^* = \{c_i \in C : \exists r_{CONSTR} \in R_{CONSTR} \wedge c_i = r_{CONSTR}[1][1] \wedge a_2 \in r_{CONSTR}[2]\}$$

Wyjaśnienie:

- $c_i = r_{CONSTR}[1][1]$ - kolumna c_i jest w relacji z pewnym zbiorem ograniczeń integralnościowych,
- $a_2 \in r_{CONSTR}[2]$ - ograniczenie integralnościowe a_2 (czyli klucz obcy) jest w zbiorze ograniczeń kolumny c_i.

Funkcja F_C^* ma za zadanie utworzenie pól służących do implementacji asocjacji typu "jeden - wiele", dla kolumn kluczy obcych:

$$F_C^* : C^* \to C' := \{(c_i, c'_i) : c_i \in C^* \wedge c'_i \in C' \wedge c'_i \neq F_C(c_i) \wedge \exists r_{REF} \in R_{REF} : c_i = r_{REF}[2][1][1]\}$$

Wyjaśnienie:

- $c'_i \neq F_C(c_i)$ - pole nie może być takie samo, jak to, na którym zaimplementowano asocjację "wiele – jeden",
- $\exists r_{REF} \in R_{REF} : c_i = r_{REF}[2][2][1]$ - kolumna c_i jest referencją.

Przykład 2.
Uzupełnijmy przykład 1 o następujące zbiory:

$$C' = \{id, imie, nazwisko, ulica, numerDomu, numerMieszkania, osoba, adresy\}$$
$$T' = \{String, Long\}$$
$$B' = \{DaneOsobowe, Adres\}$$
$$A' = \{@ManyToOne, @JoinColumn, OneToMany, @Id, @NotNull, @Unique\}.$$

Funkcja F_B odwzorowująca tabele ze zbioru B z przykładu 1 na klasy ma postać:

$$F_B = \{(dane\_osobowe, DaneOsobowe), (adresy, Adres)\}.$$

Funkcja F_T odwzorowująca typy ze zbioru T z przykładu 1 na typy ze zbioru T' ma postać:

$$F_T = \{(varchar, String), (int16, Long), (int32, Long)\}.$$

Zbiór C^* kolumn, które są kluczami obcymi wygląda następująco:

$$C^* = \{id\_osoby\},$$

ponieważ n-tka $e = ((id\_osoby, int32), \{foreign\ key, not\ null\})$ należy do R_{CONSTR} i $e[1][1] = "id\_osoby"$ i zbiór $e[2] = \{foreign\ key, not\ null\}$ zawiera element "foreign key".
Funkcja F_C wygląda następująco:

$$\begin{aligned} F_C = \ & \{(id, id), (imie, imie), (nazwisko, nazwisko), (ulica, ulica), \\ & (numer\_domu, numerDomu), (numer\_mieszkania, numerMieszkania), \\ & (id\_osoby, osoba)\} \end{aligned}$$

Funkcja F_C^* wygląda następująco:

$$F_C^* = \{(id\_osoby, adresy)\}$$

Określmy funkcję odwzorowującą ograniczenia integralnościowe SQL ze zbioru A w adnotacje ze zbioru A':

$$\begin{aligned} F_A = \{ \ & (primarykey, \{@id\}), (not\ null, \{@NotNull\}), \\ & (unique, \{@Unique\}), (foreign\ key, \{@ManyToOne, @JoinColumn\})\} \end{aligned}$$

Koniec przykładu 2.

W języku Java relacja łącząca pole klasy z typem określona jest następująco:

$$\begin{aligned} R'_{CT} \subseteq C' \times T' = \{(c'_i, t'_i) : \ & c'_i \in C' \wedge t'_i \in T' \wedge \exists t_i \in T : t'_i = F_T(t_i) \wedge \\ & \exists c_i \in C : c'_i = F_C^{-1}(c_i) \wedge \\ & (F_C^{-1}(c'_i), t_i) \in R_{CT} \wedge \\ & \nexists r_{REF} \in R_{REF} : F_C^{-1}(c'_i) = r_{REF}[2][1][1]\} \end{aligned}$$

Wyjaśnienie:

- $\exists c_i \in C : c'_i = F_C^{-1}(c_i)$ - pole c'_i jest związane z kolumną c_i, która nie jest referencją,
- $t'_i = F_T(t_i)$ – typ Javy odpowiada typowi SQL,
- $(F_C^{-1}(c'_i), t_i) \in R_{CT}$ – para (kolumna, typ SQL) jest zdefiniowana, kolumna odpowiada polu,
- $\nexists r_{REF} \in R_{REF} : F_C^{-1}(c'_i) = r_{REF}[2][1][1]$ – kolumna c'_i nie jest kluczem obcym.

Relacja R'_{CT} nie zawiera tych pól, które należą do zbioru wartości funkcji F_C^*, czyli powiązane są z kolumnami tworzącymi referencje. Typem tego pola powinna być klasa odpowiadająca tabeli, która jest po drugiej stronie referencji. Relacja łącząca pole klasy z typem w przypadku, kiedy na kolumnie związanej z polem utworzona jest referencja, wygląda następująco:

$$R''_{CT} \subseteq C' \times B' = \{(c'_i, b'_i,) : \quad c'_i \in C' \wedge b'_i \in B' \wedge \exists b_i \in B : b'_i = F_B(b_i) \wedge \exists r_{REF} \in R_{REF} : F_C^{-1}(c'_i) = r_{REF}[2][2][1] \wedge F_B^{-1}(b'_i) = r_{REF}[1][1]\}$$

Wyjaśnienie:

- $b'_i = F_B(b_i)$ - klasa Javy odpowiadająca tabeli SQL,
- $\exists r_{REF} \in R_{REF} : F_C^{-1}(c'_i) = r_{REF}[2][2][1]$ – kolumna c'_i jest kluczem obcym,
- $F_B^{-1}(b'_i) = r_{REF}[1][1]$ – typ związany z polem nazywa się tak, jaka jest nazwa klasy komponentu encyjnego, który jest po drugiej stronie referencji.

W tych komponentach encyjnych, które występują po stronie referencji o krotności "jeden", mogą powstać pola zawierające kolekcję komponentów:

$$R'''_{CT} \subseteq C' \times T' = \{(c'_i, t'_i) : c'_i \in F_C^* \wedge t'_i \in T' \wedge t_i = t'_m\}$$

W oparciu o relacje R_{CT} i R'_{CT}, zdefiniujmy funkcję

$$F_{CT} : R_{CT} \to R'_{CT} = \{((c_i, t_i), (c'_i, t'_i,)) \quad : (c_i, t_i) \in R_{CT} \wedge (c_{i'}, t_{i'}) \in R'_{CT} \wedge c'_i = F_C(c_i) \wedge t'_i = F_T(t_i)\}$$

Posiadając parę kolumna z typem z bazy danych (c_i, t_i), można określić parę (pole, typ) (c'_i, t'_i) z języka Java używając funkcji

$$F_{CT}((c_i, t_i)) := (c'_i, t'_i) \in R'_{CT} : c'_i = F_C(c_i) \wedge t'_i = F_T(t_i)$$

Podobne funkcje można zdefiniować w przypadku tych pól klas, które implementują referencje:

$$F'_{CT} : R_{CT} \to R''_{CT} = \{((c_i, t_i), (c'_i, b'_i)) : (c_i, t_i) \in R_{CT} \wedge (c'_i, b'i) \in R''_{CT}\}$$
$$F''_{CT} : R_{CT} \to R'''_{CT} = \{((c_i, t_i), (c'_i, t'm,)) : (c_i, t_i) \in R_{CT} \wedge (c'_i, t'_m) \in R'''_{CT}\}$$

Posiadając parę kolumna z typem z bazy danych (c_i, t_i), gdzie kolumna c_i jest kluczem obcym, można określić parę pole, klasa (c'_i, b'_i) z języka Java używając funkcji:

$$F'_{CT}((c_i, t_i)) := \{(c'_i, b'_i) \in R''_{CT}\}$$

Posiadając parę kolumna z typem z bazy danych (c_i, t_i), gdzie kolumna c_i jest kluczem obcym, można określić parę pole, kolekcja (c'_i, t'_m) z języka Java używając funkcji:

$$F''_{CT}((c_i, t_i)) := \{(c'_i, t'_m) \in R'''_{CT}\}.$$

Przykład 3.
Uzupełnijmy przykład 2 o relacje łączące pola klas z typami.

$$R_{CT}' \subseteq C' \times T' = \{(id, Long), (imie, String), \ldots, (numerMieszkania, String)\}. \quad (6)$$

Podczas tworzenia relacji opuszczone zostały pola `osoba`, `adresy`, ponieważ:

- $F_C^{-1}("osoba") = "id\_osoby"$ i "id_osoby" jest kluczem obcym,
- $F_C^{-1}("adresy")$ jest nieokreślona (nie jest spełniony warunek $\exists c_i \in C : "adresy" = F_C^{-1}(c_i))$).

Pole osoba jest typu DaneOsobowe:

$$R''_{CT} = \{(osoba, DaneOsobowe)\}$$

ponieważ:

- n-tka $e = ((daneosobowe, ((id, int32), \{primarykey\}),$ $(adresy, ((id\_osoby, int32), \{foreign\ key, not\ null\})))$ należy do R_{REF},
- $F_C^{-1}("osoba") = "id\_osoby"$,
- $e[2][2][1] = "id\_osoby" = F_C^{-1}("osoba")$,
- $e[1][1] = "daneosobowe" = F_B^{-1}("DaneOsobowe") = "dane\_osobowe"$.

Pole adresy jest kolekcją:

$$R'''_{CT} = \{(adresy, Set)\},$$

ponieważ zbiorem wartości funkcji F_C^* jest adresy. Funkcje odwzorowujące pary (kolumna, typ) na pary (pole, typ) są następujące:

$$F_{CT} := \{((id, int32), (id, Long)), ((imie, varchar), (imie, String)), \ldots\},$$
$$F'_{CT} := \{((id\_osoby, int32), (osoba, DaneOsobowe))\},$$
$$F''_{CT} := \{((id\_osoby, int32), (adresy, Set))\}.$$

Koniec przykładu 3.

Adnotacje pól klas w języku Java, odpowiadające ograniczeniom integralnościowym opisuje następująca relacja:

$$\begin{aligned} R_{CONSTR}' \subseteq \quad & (R_{CT}' \cup R_{CT}'' \cup R_{CT}''') \times 2^{A'} := \{(x', y') : \\ & \exists x \in R_{CT} : x' = F_{CT}(x) \Rightarrow y' = \{y_i' : \forall y_i \in F_{CONSTR}(x) : y_i' = F_A(y_i)\} \vee \\ & \exists x \in R_{CT} : x' = F'_{CT}(x) \Rightarrow y' = \{y_i' : \forall y_i \in F_{CONSTR}(x) : y_i' = F_A(y_i)\} \vee \\ & \exists x \in R_{CT} : x' = F''_{CT}(x) \Rightarrow y' = \{a_3'\}\}. \end{aligned}$$

Wyjaśnienie:

- element x' jest parą (pole, typ), element y' jest zbiorem adnotacji,
- jeżeli x' powstał z kolumny, która nie jest kluczem obcym (wtedy $x' = F_{CT}(x)$), to ma adnotacje (zbiór y') utworzone przez funkcję F_A na podstawie ograniczeń integralnościowych pary (kolumna, typ SQL). Otrzymamy te ograniczenia używając funkcji F_{CONSTR}.
- jeżeli x' powstał z kolumny, która jest kluczem obcym (wtedy $x' = F'_{CT}(x)$), to adnotacje tworzymy jak wyżej,
- jeżeli x' implementuje asocjację "jeden - wiele" (wtedy $x' = F''_{CT}(x)$), to ma jedną adnotację $\{a_3'\}$.

W oparciu o relacje R_{CONSTR} i R'_{CONSTR} określmy funkcję odwzorowującą ograniczenia integralnościowe bazy danych na adnotacje języka Java

$$\begin{aligned} F'_{CONSTR} : R_{CONSTR} \rightarrow \quad & R'_{CONSTR} := \{((x, y), (x', y')) : \forall (x, y) \in R_{CONSTR} : \\ & \exists x_i \in R_{CT} : x' = F_{CT}(x_i) \vee \exists x_j \in R_{CT} : x' = F_{CT}(x_j) \Rightarrow \\ & y' = \{y_i' : \forall y_i \in F_{CONSTR}(x) : y_i' = F_A(y_i)\}\} \end{aligned}$$

Funkcja ta nie wiąże kolumny z ograniczeniem klucz obcy z polem posiadającym adnotację @OneToMany. W celu utworzenia takiej zależności, oznaczmy jako R^*_{CONSTR} podzbiór tych ograniczeń integralnościowych bazy danych, które są kluczami obcymi:

$$R^*_{CONSTR} = \{r^*_{CONSTR} \in R_{CONSTR} : a_2 \in r^*_{CONSTR}[2]\} \subset R_{CONSTR}.$$

Określmy funkcję odwzorowującą ograniczenia integralnościowe bazy danych typu klucz obcy na adnotacje "jeden – wiele" języka Java:

$$F''_{CONSTR} : R^*_{CONSTR} \rightarrow R'_{CONSTR} := \{((x,y),(x',y')) : \quad \forall (x,y) \in R^*_{CONSTR} : \\ x' = F''_{CT}(x) \wedge y' = \{a'_3\}\}.$$

Wyjaśnienie:

- elementami R'_{CONSTR} są ((kolumna, typ), zbiór ograniczeń integr.),
- para (kolumna, typ) zamieniana jest na parę (pole, kolekcja): $x' = F''_{CT}(x)$
- zbiór adnotacji pary x' to {@OneToMany}: $y' = \{a'_3\}$.

Definicjami klas w języku programowania obiektowego nazywamy relacje postaci:

$$R_{CLASS} \subseteq B' \times R'_{CONSTR} = \{(b', r'_{CONSTR}) : b' \in B' \wedge r'_{CONSTR} \in R'_{CONSTR}\} \tag{7}$$

Funkcja F_{CLASS} odwzorowuje definicję tabeli w definicję klasy:

$$F_{CLASS} : R_{TAB} \rightarrow R_{CLASS} := \quad \{((b, r_{CONSTR}),(b', r'_{CONSTR})) : \\ \forall (b, r_{CONSTR}) \in R_{TAB} : \\ b' = F_B(b) \wedge r'_{CONSTR} = F'_{CONSTR}(r_{CONSTR})\}.$$

Wyjaśnienie:

- elementami R_{TAB} są pary (nazwa tabeli, (kolumna, typ, zbiór ograniczeń interg.)),
- $b' = F_B(b)$ – nazwa klasy powiązana z tabelą,
- $r'CONSTR = F'CONSTR(r_{CONSTR})$ – ograniczenia integr. odwzorowane z SQL na język Java.

Zbiór par (nazwa klasy, pola z ograniczeniami integralnościowymi), które tworzą definicję klas, należy uzupełnić o dodatkowe elementy, które powstają w wyniku implementacji asocjacji o krotnościach "jeden - wiele":

$$F'_{CLASS} : R_{TAB} \rightarrow \quad R_{CLASS} := \{((b, r^*_{CONSTR}),(b', r'_{CONSTR})) : \\ \forall (b, r^*_{CONSTR}) \in R_{TAB} : \\ r^*_{CONSTR} \in R^*_{CONSTR} \wedge r'_{CONSTR} = F''_{CONSTR}(r^*_{CONSTR}) \wedge \\ \exists r_{REF} \in R_{REF} : b = r_{REF}[2][1] \wedge r^*_{CONSTR} = r_{REF}[2][2] \wedge \\ b^* = r_{REF}[1][1] \wedge b' = F_B(b^*).$$

Wyjaśnienie:

- rozpatrujemy wyłącznie ograniczenia integr. typu klucz obcy: $r^*_{CONSTR} \in R^*_{CONSTR}$
- w wyniku odwzorowania ograniczenia integralnościowego z SQL na język Java powstanie pole typu kolekcja z adnotacją @OneToMany: $r'_{CONSTR} = F''_{CONSTR}(r^*_{CONSTR})$

- para (b, r^*_{CONSTR}) jest kluczem obcym pewnej referencji $(\exists r_{REF} \in R_{REF} : b = r_{REF}[2][1] \wedge r^*_{CONSTR} = r_{REF}[2][2])$ – element $r_{REF}[2]$ referencji modeluje część definiującą klucz obcy,
- tabela, która jest po przeciwnej stronie klucza obcego to $b^* = r_{REF}[1][1]$,
- klasa, która otrzyma pole typu kolekcja z adnotacją @OnoToMany to $b' = F_B(b^*)$.

W celu przekształcenia modelu R_{TAB} relacyjnej bazy danych opisanego równaniem (7), na model obiektowy R_{CLASS} opisany równaniem 7, należy użyć obydwu funkcji F_{CLASS} i F'_{CLASS}:

$$\forall (b, r_{CONSTR}) \in R_{TAB} : R_{CLASS} = F_{CLASS}(b, r_{CONSTR}) \cup F'_{CLASS}(b, r_{CONSTR})$$

Przykład 4.

Relacja, która łączy pary (pole, typ) z równania 6 z adnotacjami ze zbioru A' z przykładu 2 jest następująca:

$$\begin{aligned} &R'_{CONSTR} = \{r'_{CONSTR1}, r'_{CONSTR2}, \ldots, r'_{CONSTR8}\} = \\ &\{((id, Long), \{@Id\}), ((imie, String), \{@NotNull\}), \ldots, \\ &((osoba, DaneOsobowe), \{@ManyToOne, @JoinColumn\}), \\ &((adresy, Set), \{@OneToMany\})\} \end{aligned} \tag{8}$$

Funkcje, które odwzorowują kolumnę, typ i zbiór ograniczeń integralnościowych na odpowiednie elementy języka Java, wyglądają następująco:

$$\begin{aligned} F'_{CONSTR} := \{ \quad &((id, int32), \{primary\ key\}), ((id, Long), \{@Id\}), \\ &((imie, varchar), \{not\ null\}), ((imie, String), \{@NotNull\}), \ldots \\ &((id\_osoby, int32), \{foreign\ key, not\ null\}), \\ &((osoba, DaneOsobowe), \{@ManyToOne, @JoinColumn, @NotNull\})\} \end{aligned}$$

Zbiór ograniczeń integralnościowych, które są kluczami obcymi, ma postać:

$$R^*_{CONSTR} = \{((id\_osoby, int32), \{foreign\ key, not\ null\})\}.$$

Funkcja, która tworzy adnotacje @OneToMany:

$$\begin{aligned} F''_{CONSTR} := \{ \quad &((id\_osoby, int32), \{foreign\ key, not\ null\}), \\ &((adresy, Set), \{@OntToMany\})\} \end{aligned}$$

Definicje klas języka Java zostaną utworzone przez funkcje:

$$F_{CLASS} = \{ \quad ((dane\ osobowe, r_{CONSTR1}), (DaneOsobowe, r'_{CONSTR1})),$$
$$((dane\ osobowe, r_{CONSTR2}), (DaneOsobowe, r'_{CONSTR2})),$$
$$((dane\ osobowe, r_{CONSTR3}), (DaneOsobowe, r'_{CONSTR3})),$$
$$((adresy, r_{CONSTR1}), (Adresy, r'_{CONSTR1})), \ldots$$
$$((adresy, r_{CONSTR7}), (Adresy, r'_{CONSTR7}))\}$$

$$F'_{CLASS} = \{((adresy, r_{CONSTR7}), (daneOsobowe, r'_{CONSTR8}))\},$$

gdzie:

- $r_{CONSTR1}, \ldots, r_{CONSTR7}$ zdefiniowane są równaniem 4,
- $r'_{CONSTR1}, \ldots, r'_{CONSTR7}$ równaniem 8.

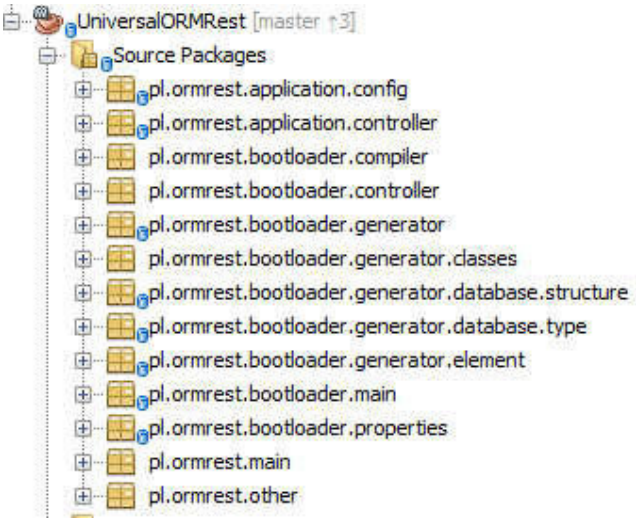
Koniec przykładu 4.

---

3. Implementacja modelu

Model opisany w poprzedniej sekcji został zaimplementowany jako usługa sieciowa typu REST w języku Java w środowisku programistycznym Netbeans 8.1, z wykorzystaniem technologii Spring 1.5.3 i Hibernate 5.0. Obsługiwane bazy danych to MySQL w wersji 10.1 i PostgreSQL w wersji 9.6. Usługa składa się z dwóch głównych modułów (rys. 3):

- bootloader - tworzy model, komponenty encyjne, usługi CRUD,
- aplikacja kontrolera udostępniająca usługi CRUD



Rys. 3. Struktura komponentu sieciowej usługi ORM

Abstrakcyjny model danych pokazany w sekcji 2 utworzony jest w taki sposób, że występują w nim zbiory i relacje określone na przestrzeni dwuwymiarowej. W związku z tym, do implementacji w języku Java zastosowano standardowe typy `Set` i `HashMap`, które posiadają wydajne funkcje wyszukiwania i iteracji po elementach kolekcji. Kod generowany z modelu sformalizowany został przy użyciu gramatyki BNF. Przykład opisu komponentu encyjnego pokazuje rys. 4.
Scenariusz użycia komponentu jest następujący:

1. Klient usługi wywołuje metodę inicjalizującą bootloadera.
   1.1. Tworzony jest abstrakcyjny model danych.
   1.2. Generowane są komponenty encyjne i klasy kontrolerów Spring z metodami typu CRUD.
   1.3. Wygenerowany kod jest kompilowany z użyciem interfejsu `JavaCompiler` pakietu `javax.tools`.
   1.4. Następuje restart modułu kontrolera.
2. Klient wywołuje metody typu CRUD kontrolera.

Tabela 1. Porównanie wydajności usługi ORM ze Spring Data Rest

	usługa ORM		Spring Data Rest	
Operacja	średni czas odpowiedzi [ms]	odch. stand. [ms]	średni czas odpowiedzi [ms]	odch. stand. [ms]
zapis	60,33	22,10	67,71	6,82
odczyt	73,62	9,95	62,13	14,04
aktualizacja	17,29	5,12	26,42	4,39

```
<model-class> ::= <class-package> <import-libraries> <class-code>
    <class-package> ::= "package" <package-name> ";"
    <import-libraries> ::= <import-library> <import-library>*
        <import-library> ::= "import" <package-name> "." <class-name> ";"
    <class-code> ::= <class-annotations> <class-definition> <class-body>
        <class-annotations> ::= <class-annotation> <class-annotation>*
            <class-annotation> ::= "@" <annotation-name> <annotation-parameters> | ""
                <annotation-parameters> ::= "(" <annotation-parameter>+ ")"
                    <annotation-parameter> ::= <parameter-name> "=" <parameter-value> "," | ""
        <class-definition> ::= "public" "class" <class-name> "implements" <interface-name>
        <class-body> ::= "{" <class-field-and-getter-setter> <class-field-and-getter-setter>* <class-constructors> "}"
            <class-field-and-getter-setter> ::= <class-field> <class-getter> <class-setter>
                <class-field> ::= <field-annotations> "private" <field-type> <field-name> ";"
                    <field-annotations> ::= <field-annotation> <field-annotation>*
                        <field-annotation> ::= "@" <annotation-name> <annotation-parameters> | ""
                <class-getter> ::= <getter-method-definition> <getter-method-body>
                    <getter-method-definition> ::= "public" <field-type> <getter-method-name> "()"
                    <getter-method-body> ::= "{" "return" "this." <field-name> ";" "}"
                <class-setter> ::= <setter-method-definition> "{" <setter-method-body> "}"
                    <setter-method-definition> ::= "public" "void" <setter-method-name> <setter-method-input>
                        <setter-method-input> ::= "(" <field-type> <field-name> ")"
                    <setter-method-body> ::= "this." <field-name> "=" <field-name> ";"
                <class-constructors-> ::= <full-constructor> <empty-constructor>
                    <full-constructor> ::= "public" <class-name> <full-constructors-method-input> <full-constructor-method-body>
                        <full-constructors-method-input> ::= "(" <input-parameter> <input-parameter>* ")"
                            <input-parameter> ::= <field-type> <field-name> "," | ""
                        <full-constructor-method-body> ::= "{" <attribution-operation> <attribution-operation>* "}"
                            <attribution-operation> ::= "this." <field-name> "=" <field-name> ";"
                        <empty-constructor> ::= "public" <class-name> "()" "{}"
```

Rys. 4. Gramatyka BNF komponentu encyjnego

Wdrożona usługa ORM została poddana testom i porównana z funkcjonalnie analogiczną usługa utworzoną z użyciem biblioteki Spring Data Rest. W tym celu opracowane zostały

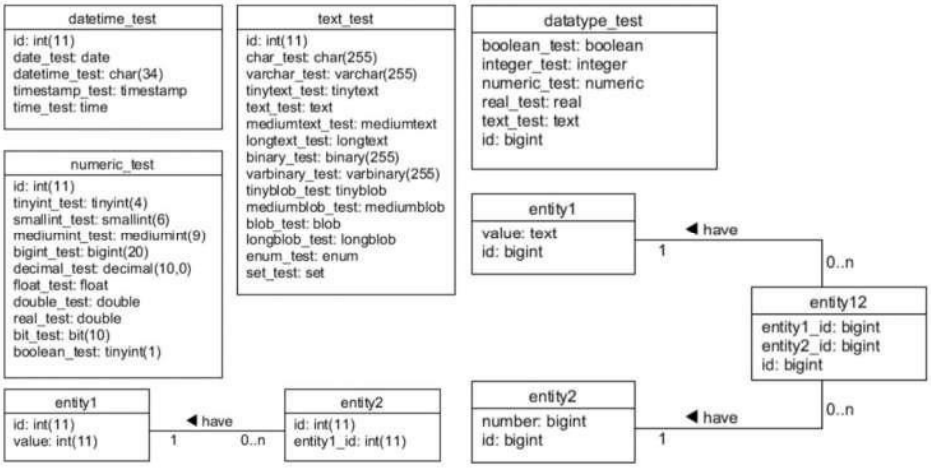
testowe bazy danych, zawierające table, którym odpowiadają diagramy klas pokazane na rys. 5. Środowisko testowe charakteryzowało się podanymi niżej parametrami:

- komputer z procesorem Intel Core i5-3210M CPU 2.50 GHz,
- pamięć RAM 8,00 GB (dostępne 7,90 GB),
- 64-bitowy system operacyjny Windows 10 Pro,
- bazy danych MySQL 10.1, PostgreSQL 9.6.

Wyniki pomiarów średnich czasów realizacji operacji zapisu, odczytu i aktualizacji danych pokazano w tabeli 1. Operacje realizowano na strukturze danych `datatype_test` z rys. 5b. Wnioski z porównania opracowanego komponentu z biblioteką Spring Data Rest są następujące:

- opracowany komponent usługi ORM i Spring Data Rest przyspieszają proces produkcji oprogramowania,
- operacje zapisu i aktualizacji danych przeprowadzane są szybciej przez usługę ORM,
- operacja odczytu danych jest wolniejsza w przypadku opracowanego komponentu.

Omawiając porównanie biblioteki Spring Data Rest z opracowanym komponentem, należy wspomnieć o różnicach w formacie zwracanych danych. Wyjaśnienie pokazane jest na listingach 1 i 2. Przedstawiony został obiekt pobrany z tabeli `datetime_test` z rys. 5a. Spring Data Rest stosuje pośrednie odwołania do danych, za pomocą adresów URL. Ma to znaczenie szczególnie w przypadku pobierania rekordów powiązanych w bazie danych przy pomocy referencji. Uniwersalna usługa ORM zwraca zawsze kompletne dane, bez stosowania odwołań pośrednich.



(a) Dla bazy danych MySQL (b)Dla bazy danych PostgreSQL

Rys. 5. Struktury danych użyte przy testach uniwersalnej usługi ORM

4. Podsumowanie

W rozdziale przedstawiono propozycję sieciowej usługi dostępu do relacyjnej bazy danych. Usługa ta tworzona jest w sposób dynamiczny, na podstawie zawartości wskazanej bazy danych. Autorzy osiągnęli następujące rezultaty:

- opracowany został formalny model matematyczny, służący do opisu wybranych elementów bazy danych, odpowiadających im konstrukcji języka Java oraz zawierający definicje odwzorowań elementów bazy danych na język Java,
- wspomniany model został zaimplementowany z użyciem współczesnych technologii informatycznych i przetestowany z użyciem programów SOAP UI [10] i JMeter [11],
- przeprowadzono badania porównujące wydajność opracowanego komponentu z funkcjonalnie analogicznym modułem zbudowanym w oparciu o bibliotekę Spring Data Rest.

Zastosowanie opracowanego komponentu w procesie implementacji systemów informatycznych pozwoli na pominięcie etapu tworzenia warstwy dostępu do danych. Komponent można również zastosować jako usługę ORM w tych technologiach, w których biblioteki ORM znajdują się na wczesnym etapie rozwoju i są niskiej jakości, a technologia udostępnia usługę klienta REST używającego formatu danych JSON.

```
{
"id": 1,
"date_test": "2017-01-02",
"datetime_test": "2017-05-15 21:14:46",
"timestamp_test": "2017-05-24 23:47:26.0",
"time_test": "12:30:30"
}
```

Listing 1: Przykład danych zwracanych przez usługę ORM

```
{
 "_embedded" : {
     "datetimeTest" : [ {
     "dateTest" : "2017-01-02",
     "datetimeTest" : "2017-05-15 21:14:46",
     "timestampTest" : "2017-05-24T21:47:26.000+0000",
     "timeTest" : "12:30:30",
     "_links" : {
      "self" : {"href" : "http://localhost:8082/datetimeTest/1"},
      "datetimeTest" : {
         "href" : "http://localhost:8082/datetimeTest/1"
      }
     }
}]
},
"_links" : {
 "self" : {"href" : "http://localhost:8082/datetimeTest"},
 "profile":{"href":"http://localhost:8082/profile/datetimeTest"}
},
"page" : {
 "size" : 20, "totalElements":1, "totalPages":1, "number":0
 }
}
```

Listing 2: Przykład danych zwracanych przez Spring Data Rest

Bibliografia

[1] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.
[2] L. Dewailly, Building a RESTful Web Service with Spring, Packt Publishing Ltd, 2015.
[3] C. Bauer, G. King, G. Gregory, Java Persistence with Hibernate, Manning Publications, 2015.
[4] B. Eckel, Thinking in Java: (4th Edition), Prentice Hall, 2006.
[5] M. Frisbie, AngularJS Web Application Development Cookbook, Prentice Hall, 2014.
[6] A.D. Lindsay, M.G. Polan, T. Tong, Mapping relational tables to object oriented classes, US Patent 6,754,670 2004
[7] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures (Ph.D.), University of California, Irvine, 2000.
[8] M. Andersen, O. Chikvina, S. Mukhina, Red Hat JBoss Developer Studio 7.0. Hibernate Tools Reference Guide, RedHat, 2011.
[9] K. Dunglas, Persistence in PHP with Doctrine ORM, Packt Publishing, 2013.
[10] P. Nandan, Mastering SoapUI, Packt Publishing, 2016.

[11] B. Erinle, *Performance Testing with Jmeter - Second Edition*, Packt Publishing, 2015.
[12] J. Duckett, *JavaScript and JQuery: Interactive Front-End Web Development*, Wiley; 1 edition, 2014.

Rozdział 5

Concluder – narzędzie analizy problemu zgodności par

1. Wstęp

Metoda porównania parami (ang. pairwise comparison, PC) to jedna z najstarszych metod oceny obiektów, preferencji, postaw, systemów głosowania, wyboru społecznego, wyborów publicznych, itd. Po raz pierwszy została zaproponowana ponad 200 lat temu przez Condorcet'a [1] and Borda'a [2] w odniesieniu do problemu głosowania. Następnie w latach dwudziestych XX wieku użyto jej w psychologii eksperymentalnej Thorndike [3] and Thurstone [4]. Jednak najbardziej znanym przykładem jej stosowania jest metoda AHP (ang. *Analytic Hierarchy Process*) zaproponowana przez Saaty'ego w latach osiemdziesiątych XXw. [5]. Powszechność stosowania metody PC wynika z faktu, iż w życiu codziennym używa się jej do oceny ilościowej własności obiektów (np. pomiar długości, wagi, itd.), rozstrzygania czy dwa przedmioty są identyczne, wyboru właściwych decyzji, itp. Jednym z kluczowych problemów w metodzie PC jest problem zgodności porównywanych par oraz określenie tzw. stopnia niezgodności samej macierzy wykonywanych porównań. Istnieje co najmniej dwa ważne podejścia używane w rozwiązywaniu tego problemu, zaś w rozdziale przedstawione zostanie narzędzie (Concluder) wspomagające jedno z nich.

2. Porównywanie parami i zagadnienie niezgodności

W metodzie porównywania parami dla zbioru n obiektów (a w szczególności ich atrybutów) możliwe jest zrealizowanie problemu decyzyjnego w następującej postaci. Dla dowolnej pary (i,j), $i,j = 1, 2,\ldots, n$ należy udzielić odpowiedzi, ile razy bardziej obiekt i jest preferowany (jest ważniejszy) niż obiekt j a wynik tego porównania jest prezentowany jako element a_{ij} macierzy porównań $\mathbf{A} = [a_{ij}]_{i,j = 1, 2,\ldots, n} \in \boldsymbol{R}^{n \times n}$. Należy zauważyć, że: $a_{ij} > 0$, $a_{ii} = 1$ oraz $a_{ij} = 1/a_{ji}$. Zapisane oceny mogą dotyczyć nie tylko preferencji osobistych (nawet prywatnych), ale także ocen ekspertów, przy czym skala tych ocen może być dowolna, byleby nie posiadała wartości ujemnych. Zwykle w takim przypadku stosuje się podejście, które pozwala na zmapowanie prezentowanych przez eksperta (lub innym podmiot) ocen (nawet w postaci opisowej) do wartości liczbowych. Tabela 1 obrazuje tego typu, przykładowe podejście.

Tabela 1. Przykładowa skala porównań

Wartość liczbowa	Definicja	Wyjaśnienie
1	Równe znaczenie	Równy wkład
2	Słabe znaczenie jednej oceny względem innej	Słaba przewaga jednego kryterium nad drugim
3	Zasadnicze lub silne znaczenie	Silna przewaga jednego kryterium nad drugim
4	Kluczowe znaczenie	Silna dominacja
5	Absolutne znaczenie	Najbardziej preferowany wybór
1.2, 2.3, …, etc.	Wartości pośrednie	Zastosowanie, gdy potrzebny jest kompromis

Dla każdego z ocenianych kryteriów można przypisać wartości liczbowe zgodnie z preferencjami oceniającego, choć tak naprawdę nie są one ocenami o charakterze absolutnym, co wynika choćby z faktu, że pewne miary po prostu nie istnieją (np. ocena stopnia bezpieczeństwa zdrowotnego), ale można np. dokonywać ich porównań (np. względem zanieczyszczenia środowiska). Dokonane w ten sposób oceny ważności wielu kryteriów w naturalny sposób tworzą swoistego rodzaju macierz. Jednak dla tego typu macierzy ważnym zagadnieniem jest zbadanie, na ile oceny eksperta są zgodne pomiędzy sobą.

O macierzy **A** porównań parami można powiedzieć, że jest zgodna, jeżeli dla każdego jej elementu jest spełniona własność

$$a_{ij} a_{jk} = a_{ik} , \tag{1}$$

dla wszystkich indeksów (i,j,k), $i,j,k = 1, 2,\ldots, n$. W związku z tym najważniejszym problemem w ocenie macierzy **A** jest zbadanie, czy jest ona zgodna oraz ewentualna ocena stopnia jej niezgodności. W [5] zostało zaproponowane podejście (aktualnie najpowszechniej akceptowane), w którym ocena stopnia niezgodności bazuje na obliczeniu największej wartości własnej macierzy λ_{max}, która spełnia warunek $\lambda_{max} \geq n$ zaś dla macierzy zgodnej mamy: $\lambda_{max} = n$. Współczynnik niezgodności CI jest definiowany jako:

$$CI_n = \frac{\lambda_{\max} - n}{n-1} . \tag{2}$$

Saaty w [5] bazując na serii losowych eksperymentów generacji macierzy **A** oraz badania ich współczynnika niezgodności, zaproponował zdefiniowanie współczynnika CR (ang. *inconsistency ratio*):

$$CR_n = \frac{CI_n}{RI_n} , \tag{3}$$

gdzie RI_n oznacza średnią wartość losowo otrzymanych wartości indeksów niezgodności CI_n. Zastosowanie współczynnika (3) zakłada, że jeżeli w macierzy $\mathbf{A}_{[n\times n]}$

poziom niezgodności *CI* nie przekracza 10% wartości *CR*, to można uznać taką wartość za akceptowalną.

Współczynnik CR_n ma kluczowe zastosowanie w metodzie AHP, która została opracowana i upowszechniona przez Saaty'ego, ale ta wszechobecna definicja niezgodności ma pewne wady, w szczególności na pierwszy plan wysuwa się tu problem odszukania najbardziej niezgodnego elementu, co wynika m.in. za zastosowania wartości własnej λ_{max} a ta jest cechą globalną macierzy. Dlatego Koczkodaj w [6] zaproponował zdecydowanie odmienne podejście do tego problemu.

Dla pojedynczej triady (x, y, z) wartości znajdujących się w macierzy **A** wartość współczynnika niezgodności *ii* jest definiowana jako:

$$ii = 1 - \min\left(y/(x*z), (x*z)/y\right). \tag{4}$$

Jeżeli w macierzy **A** wszystkie triady spełniają ten warunek, to macierz jest zgodna zaś współczynnik *ii* można zinterpretować jako względną odległość do najbliższej zgodnej macierzy odwrotnej. Dla przykładowej macierzy $\mathbf{B}_{[3\times3]}$:

$$B = \begin{bmatrix} 1 & 1.5 & 2.5 \\ 1/2 & 1 & 2 \\ 5/2 & 2/3 & 1 \end{bmatrix} \tag{5}$$

$b_{12} = 1.5$, $b_{13} = 2.5$ i $b_{23} = 2$. Jak można zauważyć $b_{12} * b_{23} \neq b_{13}$ ($1.5 * 2 \neq 2.5$) – macierz nie jest zgodna – zaś bazując na zależności (4) otrzymujemy:

$$ii = 1 - \min\left(2.5/(1.5*2), 1.5*2/2.5\right) = 1 - \min(0.83333..., 1.875) \cong 0.16666... .$$

Otrzymany wynik pokazuje stopień niezgodności triady (1.5, 2.5, 2) w macierzy **B**.

Przewagą podejścia zaproponowanego w [6] nad koncepcją Saaty'ego jest to, że wartość *ii* jest znormalizowana, $ii \in \langle 0,1 \rangle$, co w rezultacie nie prowadzi do paradoksów oraz problemów z interpretacją otrzymywanych wyników [7]. Ale najważniejszy jest fakt możliwości wskazania, która z triad macierzy nie spełnia warunku zgodności i korekta tej sytuacji. Jeżeli tego typu triad jest w macierzy więcej, to można operację korekty wartości triad powtórzyć wielokrotnie.

Poszukiwanie niezgodnych triad w macierzy ocen jest operacją dość czasochłonną, zwłaszcza gdy liczba n obiektów jest dość duża. Łatwo zauważyć, że wraz ze wzrostem wielkości macierzy rośnie ona jak $O(n^2)$. Dlatego można posłużyć się narzędziami automatyzacji tego procesu.

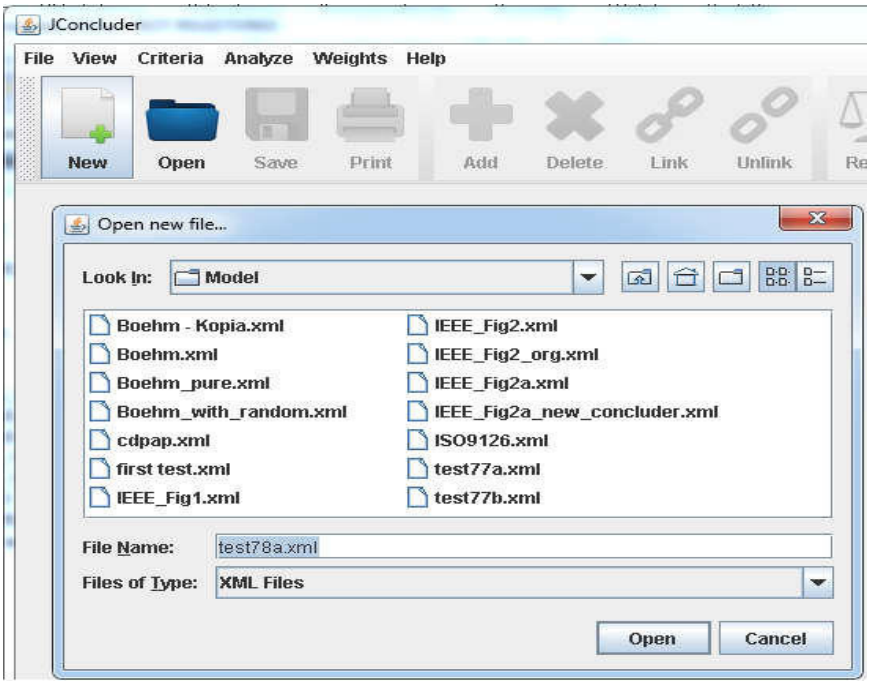
3. Narzędzie analizy problemu niezgodności par – Concluder

Aplikacja pozwalająca na automatyzację procesu poszukiwania niezgodności ocen w macierzy ocen nosi nazwę Concluder i jest dostępna jako narzędzie typu Open Access na stronie [8]. Narzędzie jest napisane jako aplikacja uruchamiana w środowisku Java

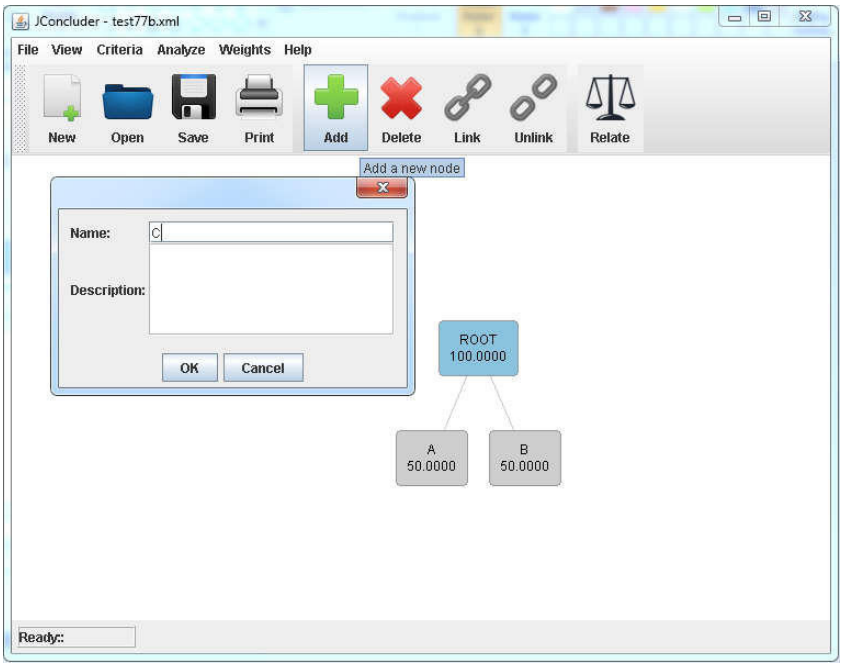
zatem jest niezależna od platformy systemu operacyjnego zaś aktualnie najnowsza wersja oprogramowania posiada oznaczenie Concluder3.33019.jar.

Po uruchomieniu aplikacji (Rys. 1) użytkownik powinien rozpocząć pracę klikając w górnym menu na przycisk **New** i podać nazwę nowego pliku danych aplikacji. Istnieje także możliwość otwierania wcześniej zapisanych plików – należy kliknąć **Open**. W programie Concluder fakt istnienia poszczególnych kryteriów ocen (cech) oraz istniejących pomiędzy nimi zależności jest reprezentowany w postaci drzewa (grafu), które może posiadać do ośmiu poziomów (korzeń znajduje się w wierzchołku Root). Zaznaczając odpowiedni wierzchołek klikając ikonę **Add** można dodawać kolejne liście drzewa podając nazwę (Rys. 2).

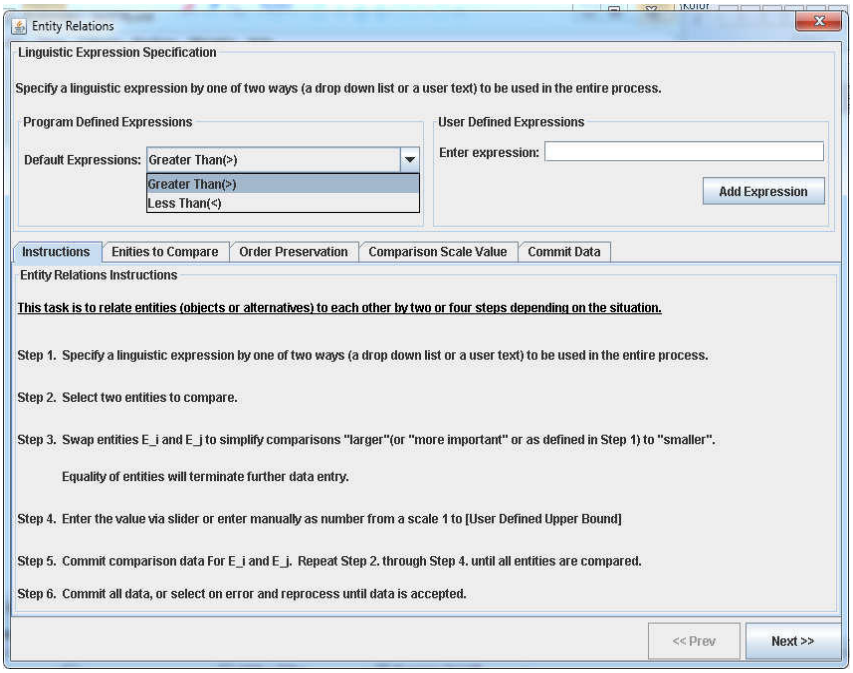
Kolejnym krokiem po stworzeniu grafu, jest wykonanie operacji **Relate** (Rys. 3), czyli ustawienia relacji, jakie istnieją pomiędzy poszczególnymi wierzchołkami grafu. Dla tej opcji istnieją w programie instrukcje (Rys. 3) należy podać, jakiego typu jest to relacja (Rys. 4), ustawić, w jakim stopnie poszczególne wartości macierzy są od siebie zależne (Rys. 5). Na koniec istnieje możliwość sprawdzenia (Rys. 6) wszystkich istniejących w macierzy relacji oraz poziomu wartości porównań (Rys. 7).



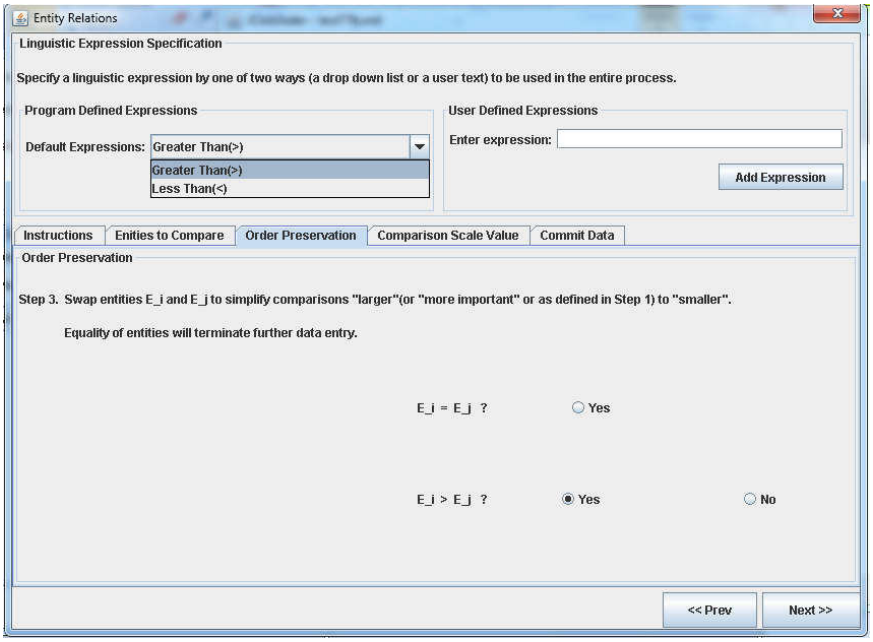
Rys. 1. Ekran początkowy program Concluder



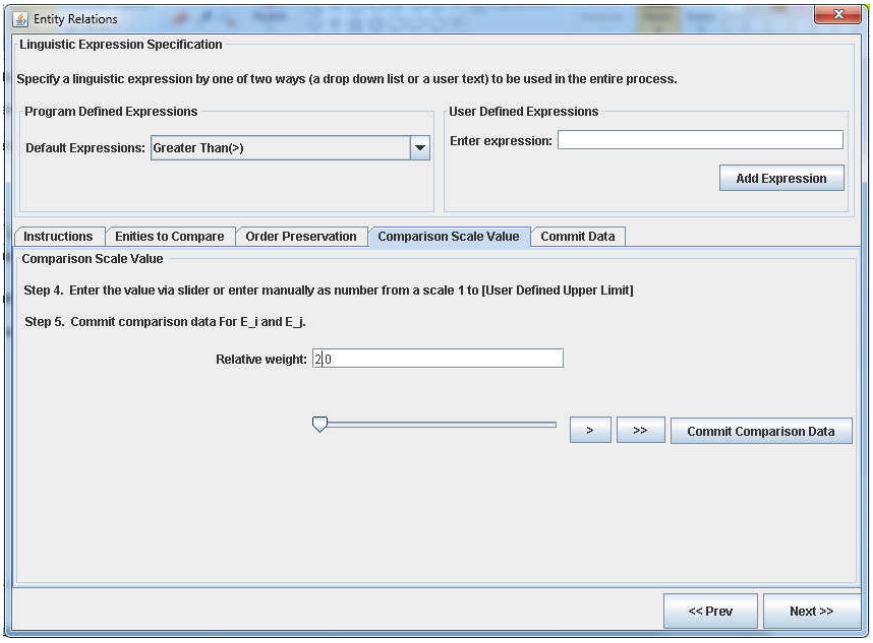
Rys. 2. Dodawanie kolejnych elementów grafu (macierzy)



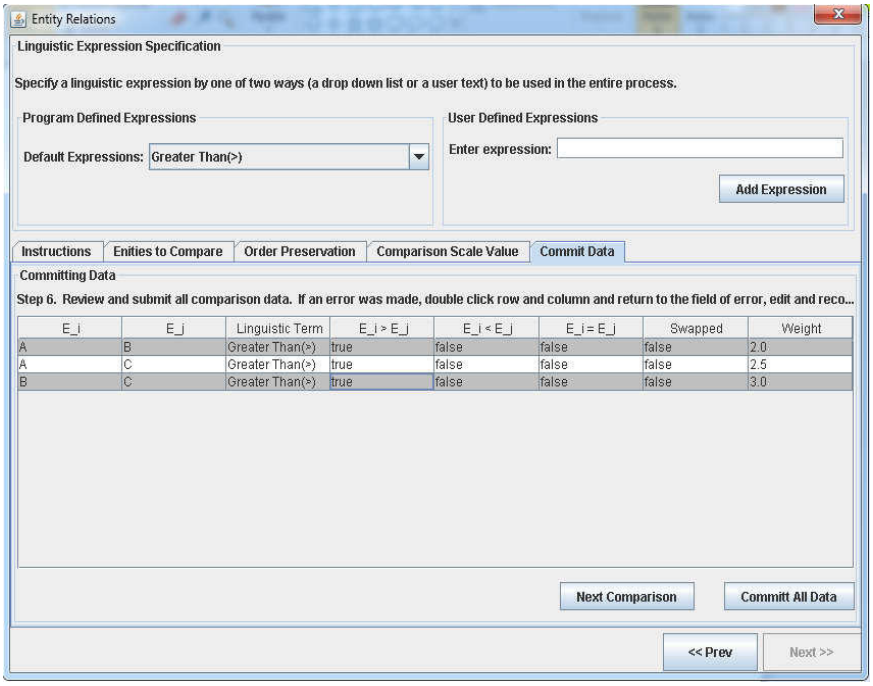
Rys. 3. Początkowe okno operacji **Relate** z instrukcjami postępowania



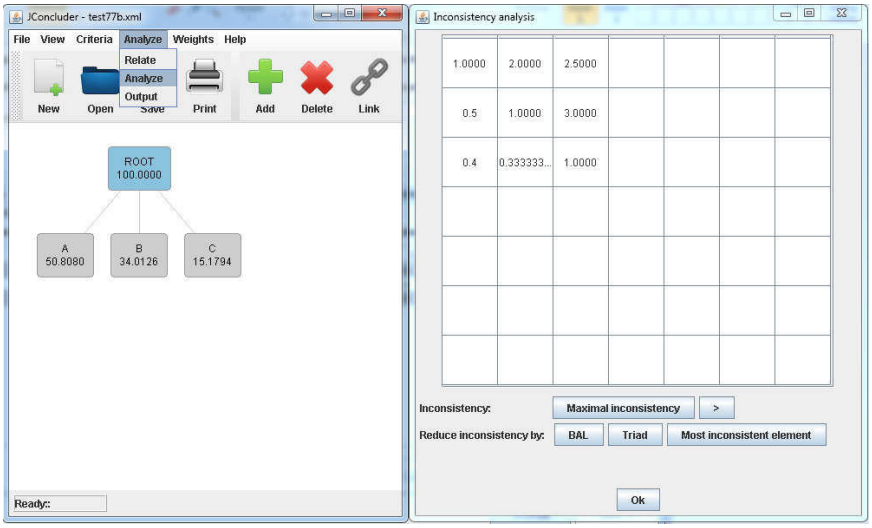
Rys. 4. Operacja wpisywania relacji pomiędzy poszczególnymi elementami macierzy. Możliwe opcje to: dla $E_i > E_j$: Większy niż (Grater Than), Mniejszy niż (Less Than) lub równy $E_i = E_j$



Rys. 5. Wpisywanie liczbowej wartości relacji



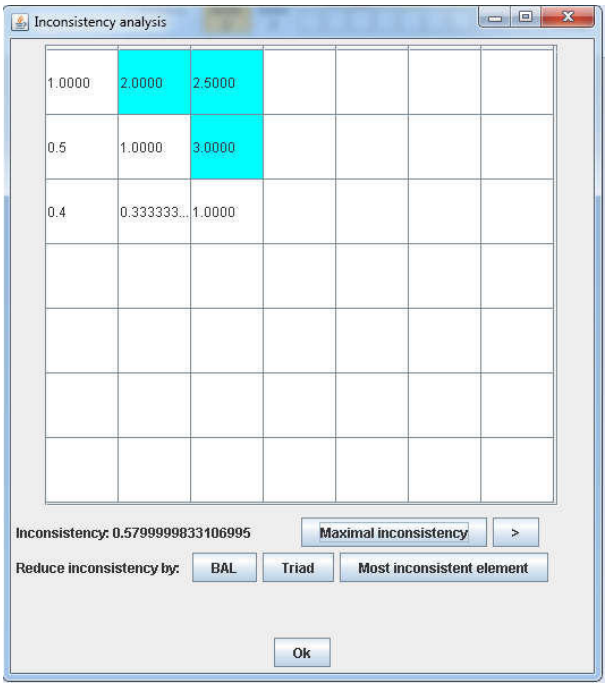
Rys. 6. Przegląd wartości wpisanych relacji macierzy porównań elementów



Rys. 7. Wynik działania polecenia **Analysis**. Macierz przedstawia prowadzone wcześniej wartości porównań

Po wprowadzeniu macierzy porównywanych elementów, z Menu **Analyze** należy wybrać polecenie **Analyze** (Rys. 7) i efektem tej operacji będzie wyświetlenie macierzy porównań elementów, po kliknięciu na polecenie **Maximal inconsistency** pojawi się

wskazanie (podświetlone elementy, Rys. 8) dotyczące stopnia niezgodności (*Inconsistency*) w macierzy, ze wskazaniem triady, która generuje określoną wartość niezgodności. Osoba wykonująca analizę może w każdym kroku podjąć decyzję, czy istniejąca w macierzy niezgodność jest do zaakceptowania. W szczególnym przypadku można doprowadzić do sytuacji gdy $ii = 0$.



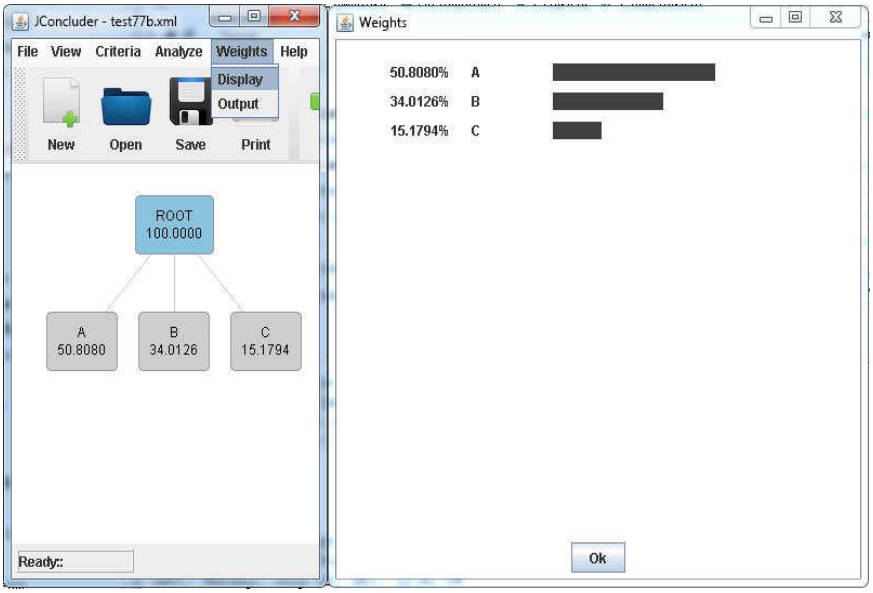
Rys. 8. Analiza poziomu niezgodności. Podświetlona została triada o największej wartości niezgodności

Końcową fazą użycia programu może być wygenerowanie wartości wag, jakie powinny być brane pod uwagę w odniesieniu do poszczególnych argumentów. Służy temu polecenie **Weights/Display** (Rys. 9). Warto zauważyć, że na samym początku, kiedy macierz porównań jest dopiero konstruowana, program Concluder przyjmuje, że znaczenie wszystkich argumentów (węzłów grafu) jest dokładnie takie samo (Rys. 2, gdzie dla dwóch istniejących węzłów A i B przypisano wagi wynoszące 50%). Jednak operacja poszukiwania niezgodności ocen oraz redukowania tego zjawiska finalnie prowadzi do zmiany wartości wag końcowych poszczególnych ocen tak, aby była zachowana zgodność dla całej macierzy (Rys. 9).

W prezentowanej wersji oprogramowania, po nawiązaniu kontaktu z jego twórcą i pomysłodawcą, została znacząco poprawiona metodyka wyznaczania wartości *ii*. W poprzednich wersjach automatyzacja obliczeń mogła prowadzić do nieoczekiwanych rezultatów związanych m.in. z niewłaściwym zaokrąglaniem uzyskiwanych wyników. Było to szczególnie widoczne, gdy używano niecałkowitych wartości liczbowych ocen porównań (zgodnie z Tabelą 1).

Warto także podkreślić, że w pracy [9] wykazano matematycznie, podając również kontrprzykłady, że *CI* nie spełnia podstawowych wymogów matematycznych, zaś

w artykule [10] jednoznacznie wykazano, że zbieżność macierzy zgodnej jest do wektora średnich geometrycznych a nie do wektora wartości własnych (również podając odpowiednie kontrprzykłady).



Rys. 9. Finalny efekt działania aplikacji wraz z wyświetlonymi wynikami wartości wag końcowych poszczególnych kryteriów branych w ocenie końcowej analizowanego problemu.

4. Podsumowanie

W rozdziale przedstawiono możliwości pracy w programie Concluder, które jest prostym, intuicyjnym, ale bardzo skutecznym narzędziem analizy problemu zgodności macierzy ocen w metodzie porównywania parami. Rozważania odniesiono do koncepcji współczynnika niezgodności zaproponowanego przez W. W. Koczkodaja, który zdecydowanie lepiej odnosi się do tego zagadnienia niźli rozwiązania znane z metody AHP. W szczególności program Concluder pozawala na łatwe odszukanie tych triad, które są niezgodne, dzięki czemu możliwe są działania prowadzące np. do zmiany ocen lub też metod automatycznego wyrównywania ocen. Program nadal będzie rozwijany poprzez dodawanie kolejnych funkcjonalności.

Zaprezentowane w poprzednim rozdziale podejście zapewne nie jest bardzo przełomowe, bowiem prezentuje stosunkowo dość proste narzędzie automatyzacji pewnego procesu, który jest m.in. częścią AHP. Należy jednak zauważyć, że kluczowa jest tu kwestia samej niezgodności. Warto podkreślić, że współczynnik *ii* może posiadać interpretację, która z pozoru może wydawać się nieco zaskakująca, ale w rzeczywistości jest miarą stopnia niespójności wydawanych ocen. W pracy [11] pokazano, że niewłaściwe podejście do problemu oceny niezgodności, w szczególności użycie np.

metryki euklidesowej, jest podejściem błędnym, prowadzącym do paradoksów i powinno opierać się o wskaźnik, który jest unormowany.

Bibliografia

[1] M. Condorcet, Essai sur l'application de l'analyse à la probabilité des décisions rendues á la pluralité des voix, Paris, 1785.
[2] J.C. do Borda, Mémoire sur les électiones au scrutin, Histoire de l'Académie Royale des Sciences, Paris, 1781.
[3] E. L. Thorndike, A constant error in psychological ratings, Journal of Applied Psychology **4** (1920), 25–29.
[4] L. L. Thurstone, The method of paired comparisons for social values, Journal of Abnormal and Social Psychology 21 (1927), 384–400.
[5] T.L. Saaty, The analytic hierarchy process, McGraw-Hill, New York, 1980 and 1990.
[6] W.W. Koczkodaj, A new definition of consistency of pairwise comparisons, Math. Comput. Model. 18(7**)** (1993), 79–84.
[7] W.W. Koczkodaj, Pairwise comparisons rating scale paradox, in: Transactions on Computational Collective Intelligence XXII, 9655 (2016), 1–9.
[8] Concluder, https://sourceforge.net/projects/concluder/, (2016).
[9] W.W. Koczkodaj, R. Szwarc: On Axiomatization of Inconsistency Indicators for Pairwise Comparisons, Fundamenta Informaticae 132(4**)** (2014), 485–500
[10] W. W. Koczkodaj, J. Szybowski: The limit of inconsistency reduction in pairwise comparisons, International Journal of Applied Mathematics and Computer Science 26(3**)** (2016), 721–729.
[11] W.W.Koczkodaj, J.-P.Magnot, J.Mazurek, J.F.Peters, H.Rakhshani, M.Soltys, D.Strzałka, J.Szybowski, A.Tozzi: On normalization of inconsistency indicators in pairwise comparisons, International Journal of Approximate Reasoning 86 (2017), 73–79.

Część 2
Projektowanie, analiza i zastosowanie systemów czasu rzeczywistego

Rozdział 6

Tryb konfiguracji pewnego rozproszonego systemu sterowania

1. Wprowadzenie

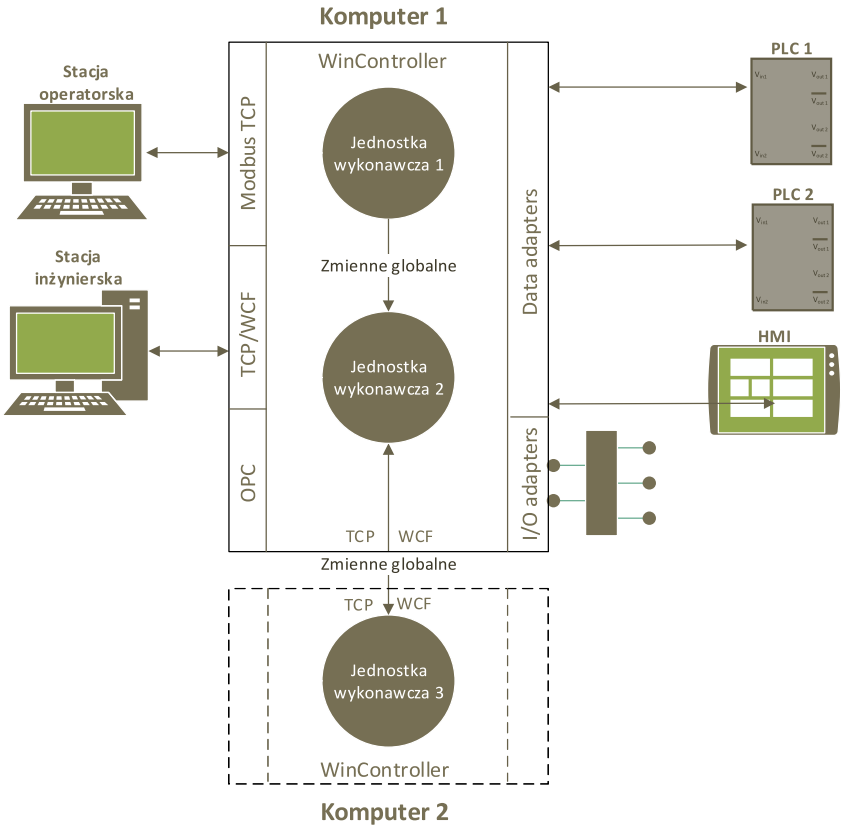
Prototypowa wersja środowiska inżynierskiego CPDev (*Control Program Developer*) służącego do programowania sterowników PLC/PAC w tekstowych językach ST, IL normy IEC 61131-3 [1] powstała w Politechnice Rzeszowskiej 10 lat temu we współpracy z LUMELem, Zielona Góra, przy wsparciu z KBN/MNiSW [2]. Z założenia CPDev miał być niezależny od platformy sprzętowej, tak że obecnie można go implementować w procesorach AVR, ARM, modułach typu Raspberry Pi, komputerach PC (Windows, Linux, QNX) oraz w układach FPGA [3]. Wynika to stąd, że skompilowany projekt jest wykonywany przez maszynę wirtualną dostosowaną do konkretnej platformy [4]. W ciągu ostatnich lat CPDev został uzupełniony o nowe edytory graficznych języków FBD i LD [5], a niedawno także o edytor sekwencyjnych schematów SFC [7]. Rozbudowano również znacząco kompilator ST umożliwiając tworzenie zaawansowanych programów z wielowymiarowymi tablicami i hierarchicznymi strukturami, podobnie jak w znanym pakiecie CODESYS [8].

Dotychczasowe zastosowania środowiska CPDev dotyczyły sterowników PLC/PAC oraz niewielkich systemów rozproszonych (miniDCS). Jednak zwiększenie możliwości kompilatora wskazało, że warto byłoby podjąć próbę zastosowania CPDeva również w komputerze nadrzędnym rozproszonego systemu DCS średniej wielkości. Założono więc na wstępie, że komputer będzie zawierał system Windows (w szczególności Windows Embedded), a wykonywany program będzie implementacją usługi *Windows service* [9]. Należało zatem utworzyć oprogramowanie, które jako implementacja usługi Windows będzie wykonywało skompilowane projekty CPDev. Nadano mu nazwę WinController.

Celem niniejszej pracy jest przedstawienie prototypu WinControllera w aspekcie trybu konfiguracji wykonywanych przez niego programów użytkowych i powiązań między nimi. Powinno to dać pogląd odnośnie perspektywy praktycznego zastosowania. Konfigurację przedstawiono na przykładzie dwu projektów wykonywanych bądź przez jeden komputer (WinController), bądź przez dwa oddzielne komputery w sieci. Projekty wymieniają dane przez połączenie zmiennych globalnych..

2. Architektura ogólna

Architekturę rozpatrywanego systemu pokazano na rys. 1. Centralną część stanowią komputery nadrzędne połączone z jednej strony ze sterownikami i innymi urządzeniami obiektowymi, a z drugiej ze stacjami operatorskimi i stacją inżynierską. Oprogramowanie komputera nadrzędnego nosi nazwę WinController i składa się z jednostek wykonujących projekty CPDev (tzw. *execution units*). Jednostki te wymieniają między sobą dane poprzez zmienne globalne zadeklarowane w projektach. Dotyczy to również wymiany danych między komputerami.



Rys. 1. Architektura systemu rozproszonego konfigurowanego w CPDev

Sterowniki i urządzenia obiektowe dołącza się za pośrednictwem interfejsów *Data adapters*. Obejmuje to obsługę standardowych protokołów komunikacyjnych Modbus RTU, czy Profibus DP, a także implementację kanałów niestandardowych, w tym logicznych, przekazujących dane wszystkich typów zdefiniowanych w normie IEC

61131-3 [1], również struktur i tablic. Interfejsy *Data Adapters* oraz ich konfiguratory tworzone są indywidualnie dla określonych rozwiązań sprzętowych (pliki *.dll*).

Natomiast do I/O adapters należą interfejsy kart wejścia/wyjścia dostępne od paru lat w implementacjach PC środowiska CPDev (jako *softPLC*). Są one jednak ograniczone do zmiennych analogowych (REAL) i binarnych (BOOL), obsługując obecnie m.in. karty National Instruments USB 6008 i RT-DAC/USB z Inteco Kraków. W WinControllerze dodano jeszcze *Keyboard Led adapter* aktywujący trzy LEDy klawiatury (p. 5), co przydaje się do prostego testowania.

Komunikację między komputerami oraz stacją inżynierską z pakietem programistycznym CPDev do tworzenia oprogramowania dla WinControllera zapewnia mechanizm TCP/WCF. Stację operatorską SCADA, np. z pakietem InTouch Wonderware [10], dołącza się za pomocą Modbusa TCP. Dostępny jest również protokół OPC, ale na razie w ograniczonym zakresie.

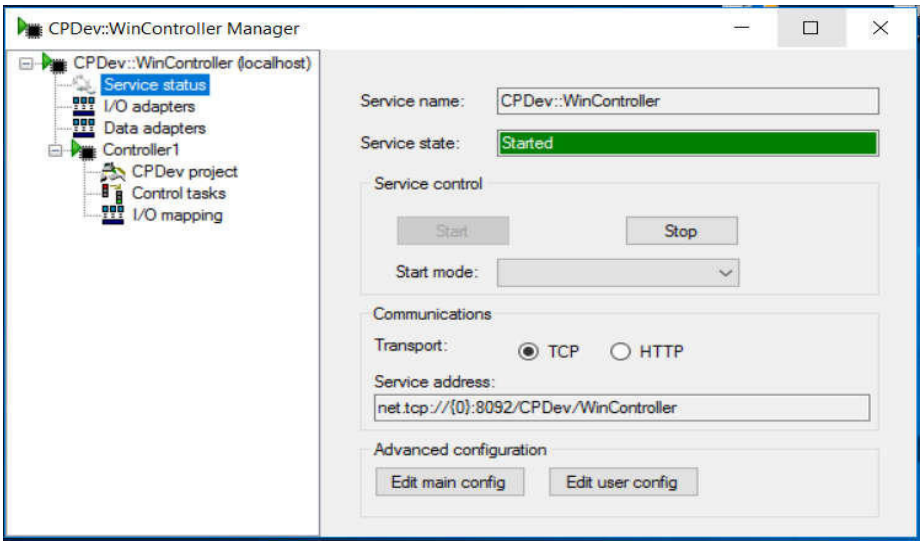
3. Charakterystyka WinControllera

WinController jest oprogramowaniem wykonawczym (*runtime*) dla projektów przygotowanych w środowisku CPDev, przeznaczonym do ciągłego funkcjonowania w systemie operacyjnym Windows. Składa się z ośmiu modułów:

- WinController service
- WinController manager
- Data source for CPSim and CPVis
- External variables
- Modbus slave (TCP)
- OPC server bridge
- I/O adapters
- Data adapters.

Zasadniczym modułem jest *WinController service* implementujący usługę *Windows service* [9]. Pracuje ona nieprzerwanie w tle (*background*) uruchamiając się automatycznie po włączeniu komputera. Usługa nie ma interfejsu użytkownika, a do konfiguracji służy *WinController manager* zwykle uruchomiony na stacji inżynierskiej i komunikujący się z usługą za pośrednictwem TCP/WCF (rys. 1). Moduł *WinController service*, zwany dalej w skrócie WinControllerem, może jednocześnie wykonywać wiele projektów CPDev za pomocą jednostek wykonawczych (*execution units*). Na jednostkę składa się maszyna wirtualna CPDev [4], jej otoczenie oraz skompilowany projekt. Liczba jednostek jest konfigurowana. Zakłada się, że projekty wykonywane przez jednostki dotyczą przede wszystkim zaawansowanego przetwarzania, bo bezpośrednie sterowanie obiektem prowadzą sterowniki.

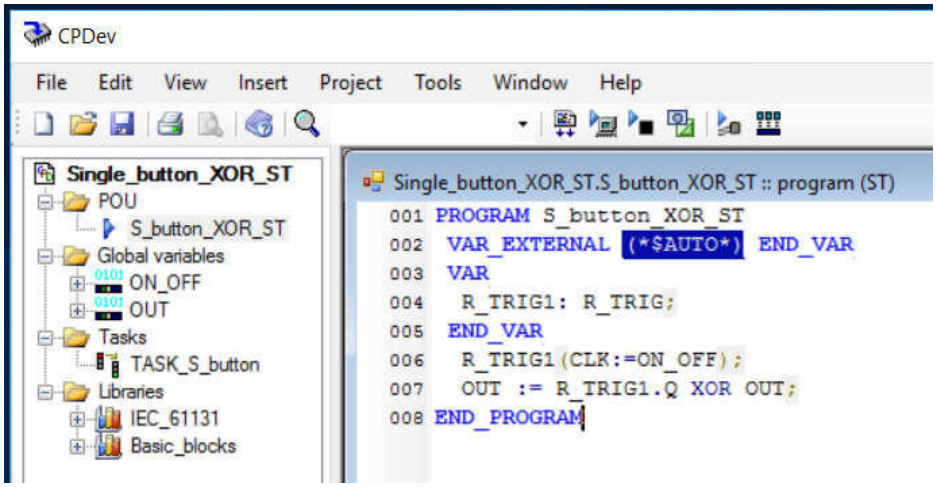
Po pierwszym uruchomieniu WinControllera okno menedżera ma postać jak na rys. 2. Drzewo po lewej stronie zawiera *Service status, I/O adapters* i *Data adapters* oraz domyślną jednostkę wykonawczą *Controller1* z pustymi elementami *CPDev project*, *Control tasks* i *I/O mapping*. Należy dodać, że przyciski Start, Stop są potrzebne tylko w specjalnych sytuacjach (wymagane uprawnienia administratora), bo WinController powinien pracować nieprzerwanie.



Rys. 2. *WinController manager* przy pierwszym uruchomieniu

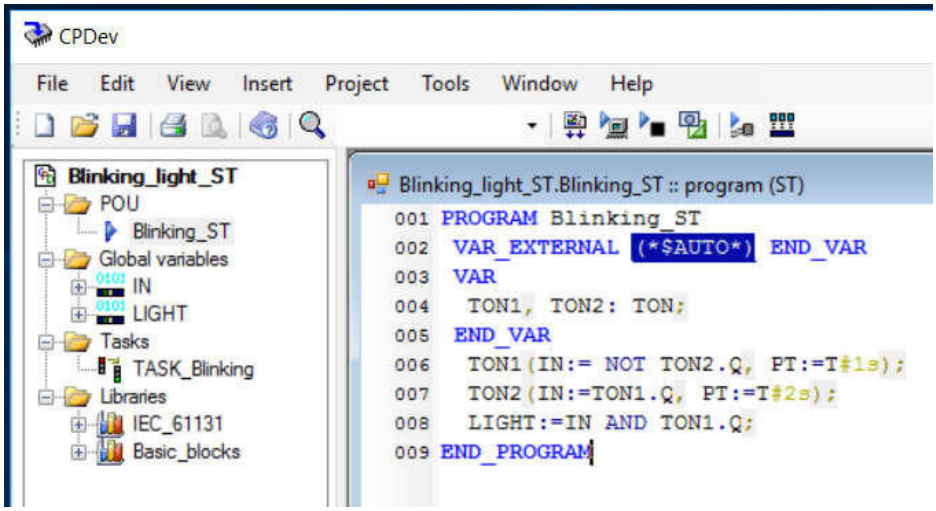
4. Dwa przykładowe projekty

Jako przykład przedstawimy skonfigurowanie w WinControllerze jednostek wykonawczych dla dwóch projektów CPDev. W następnym punkcie jednostki te zostaną połączone za pomocą zmiennych tworząc jedno spójne oprogramowanie komputera.



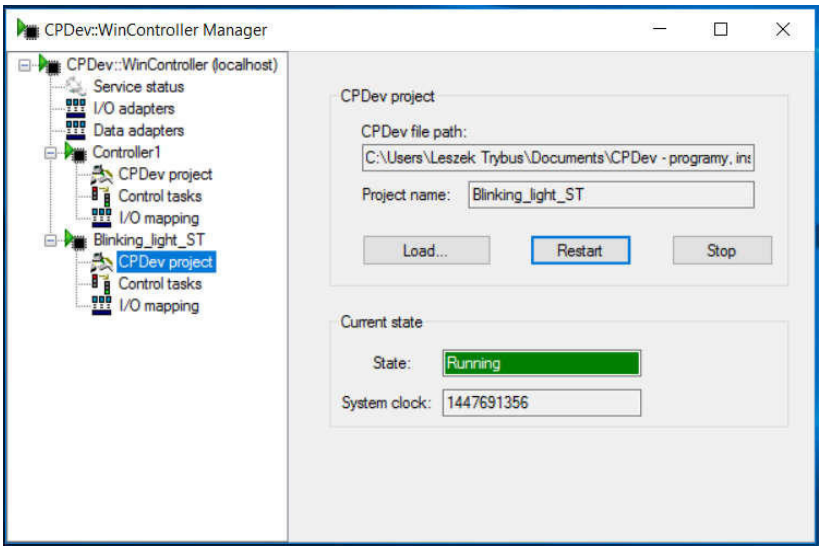
Rys. 3. Projekt *Single_button_XOR_ST*: ustawianie i zerowanie wyjścia OUT jednym przyciskiem

Program w języku ST dla pierwszego projektu o nazwie *Single_button_XOR_ST* pokazano na rys. 3 [2,4]. Za pośrednictwem detektora narastającego zbocza R_TRIG z biblioteki IEC_61131 oraz bramki XOR objętej sprzężeniem zwrotnym, kolejne „naciśnięcia" ON_OFF powodują ustawienie lub zerowanie wyjścia OUT (TRUE, FALSE).



Rys. 4. Projekt *Blinking_light_ST*: pulsowanie zmiennej LIGHT pod warunkiem ustawienia wejścia IN

W drugim projekcie *Blinking_light_ST* pokazanym na rys. 4, wyjście LIGHT powoli pulsuje, gdy wejście IN jest ustawione. Wyjście TON1Q pierwszego z dwu czasomierzy TON połączonych w układ oscylatora pulsuje bez przerwy (2+1 sek.). Przed zainstalowaniem w WinControllerze obydwa projekty powinny być skompilowane, tzn. dostępne w formie plików *.dcp*.



Rys. 5. WinController z dwoma jednostkami wykonawczymi: *Controller1* i *Blinking_light_ST*

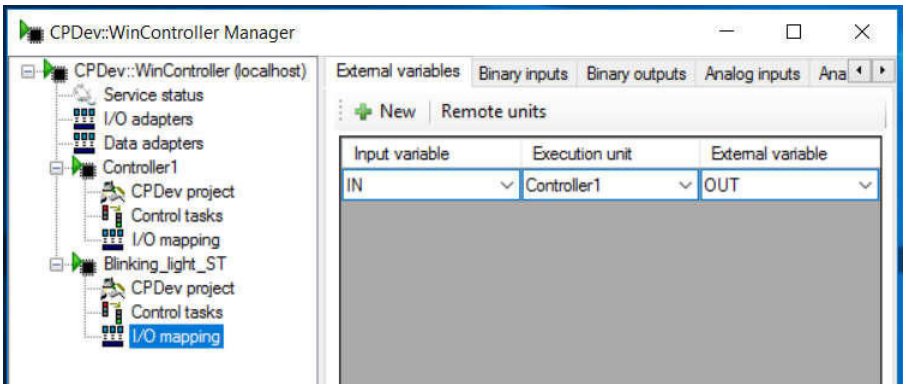
Załóżmy, że pierwszy z tych projektów ma być wykonywany przez domyślną jednostkę *Controller1*, początkowo pustą. Po wybraniu w oknie managera *Controller1* → *CPDev Project* → *Load*, należy wskazać skompilowany plik projektu *Single_button_XOR_ST*. Projekt zostanie wtedy załadowany i WinController od razu podejmie jego wykonywanie (*Running*).

Druga jednostka wykonawcza może zachować nazwę projektu, tj. *Blinking_light_ST*, co bywa wygodniejsze niż pozostawienie nazwy domyślnej (*Controller2, 3* itd.). W tym celu z menu kontekstowego elementu *CPDev:: WinController (localhost)* należy wybrać opcję *Add execution unit for a project*, i wskazać plik projektu *Blinking_light_ST*. Utworzona zostanie wówczas druga jednostka wykonawcza z *CPDev project* w stanie *Running*, jak to pokazano na rys. 5.

5. Połączenie jednostek wykonawczych

Załóżmy teraz, że zmienna IN projektu *Blinking_light_ST* ma być połączona ze zmienną wyjściową OUT projektu *Single_button_XOR_ST*. Zgodnie z normą IEC 61131-3 obydwie te zmienne powinny być zadeklarowane jako zmienne globalne [1].

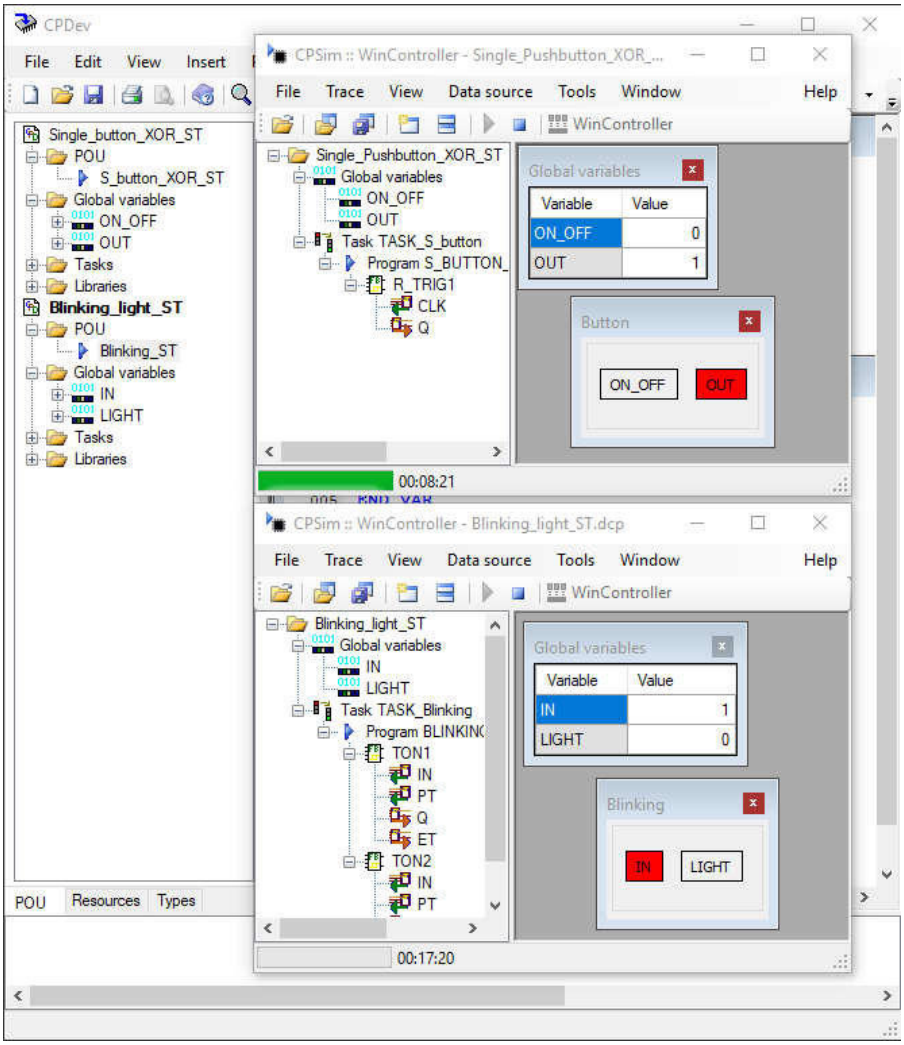
Utworzenie połączenia OUT=>IN (odpowiednik IN:=OUT) rozpoczyna się od wybrania opcji *I/O mapping* w jednostce „odbierającej" *Blinking_light_ST*, co oznacza, że pewne zmienne wykonywanego przez nią projektu będą mogły być połączone ze zmiennymi zewnętrznymi pochodzącymi z różnych źródeł, takich jak *Execution units* (*local, remote*), *I/O adapters* i *Data adapters*. Wykaz możliwych źródeł pojawia się wtedy w menadżerze, poczynając od *External variables*, *Binary inputs*, aż do *Data adapters*.



Rys. 6. Połączenie wejścia IN z wyjściem OUT dwu jednostek wykonawczych

Ponieważ w omawianym przypadku chodzi o połączenie IN ze zmienną OUT, wybieramy zakładkę *External variables* i po naciśnięciu *New* spośród zmiennych pojawiających się na liście przyporządkowanej do kolumny *Input variable* zaznaczamy IN, jak na rys. 6 (kolumna ta dotyczy jednostki z wybranym *I/O mapping*, tutaj *Blinking_light_ST*). Następnie w kolumnie *Execution unit* trzeba wskazać jednostkę ze zmienną, do której IN ma być dołączone, czyli *Controller1*. Wreszcie, w kolumnie *External variable* spośród zmiennych globalnych projektu wykonywanego przez *Controller1*, czyli *Single_button_XOR_ST*, wybieramy OUT. Teraz jednostkę *Blinking_light_ST* (z IN) należy ponownie uruchomić (*Restart*), aby WinController wczytał nową konfigurację *I/O mapping*. Kończy to tworzenie i uruchamianie połączenia OUT=>IN.

Sprawdzenia, czy obydwie jednostki faktycznie zostały połączone można dokonać za pomocą aplikacji CPSim [2,6,7]. W przypadku, gdy *CPDev::WinController* został zarejestrowany w CPSim jako aktualne źródło (*Data source*), można prowadzić śledzenie *on-line* wskazanej jednostki, testować oraz prowadzić uruchamianie (*commissioning*).

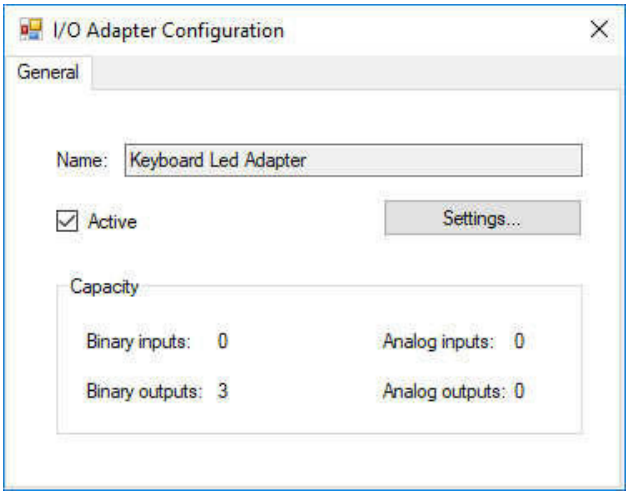


Rys. 7. Okno CPDev oraz dwa okna CPSim dla projektów wykonywanych przez WinController

Załóżmy, że w drzewie głównego okna kompilatora CPDev znajdują się dwa rozpatrywane projekty, tj. *Single_button_XOR_ST* i *Blinking_light_ST* (lewa strona rys. 7). Należy teraz zaznaczyć jeden z nich, uruchomić CPSim, a następnie poprzez *Start tracing* aktywować śledzenie. Pojawia się wtedy pytanie o wskazanie śledzonej jednostki, a po jej wybraniu CPSim nawiązuje połączenie z WinControllerem. Po utworzeniu panelu wizualizacyjnego można śledzić i ustawiać zmienne tej jednostki. Analogicznie należy postąpić dla drugiego projektu. Na rys. 7 pokazano główne okno CPDev (w tle), a nad nim dwa okna aplikacji CPSim dla projektów wykonywanych przez WinController: górne dla *Single_button_XOR_ST*, dolne dla *Blinking_light_ST*. LEDy IN i OUT zachowują się identycznie w obu oknach, czyli zmiana wartości IN w górnym pociąga za

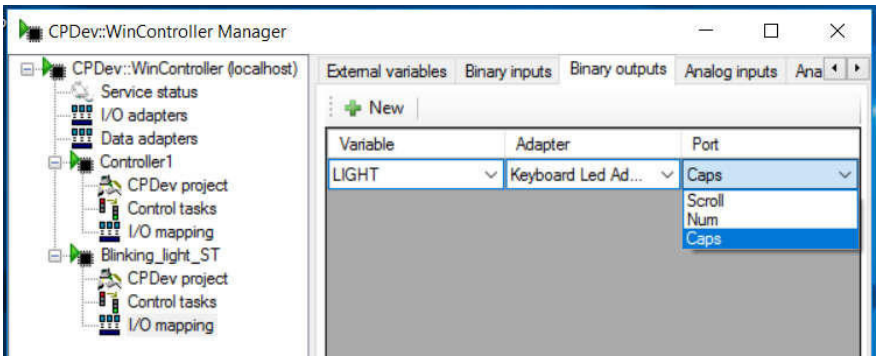
sobą automatyczną zmianę OUT w dolnym. Świadczy to o połączeniu zmiennych globalnych projektów.

Jak wspomniano w p.2, wśród *I/O adapters* dostępnych w WinControllerze znajduje się *Keyboard Led Adapter* obsługujący trzy LEDy klawiatury PC, tj. ScrollLock, NumLock i CapsLock. Są one traktowane jako wyjścia binarne, więc mogą być ustawione przez zmienne boolowskie służąc np. do wstępnego testowania. Rysunek 8 przedstawia główne okno konfiguracyjne tego adaptera. Podobnie konfiguruje się adaptery I/O dla wspomnianych kart NI USB 6008 i RT_DAC/USB, przy czym oczywiście liczba wejść i wyjść binarnych i analogowych jest właściwa dla tych kart.



Rys. 8. Konfiguracja adaptera *Keyboard Led Adapter*

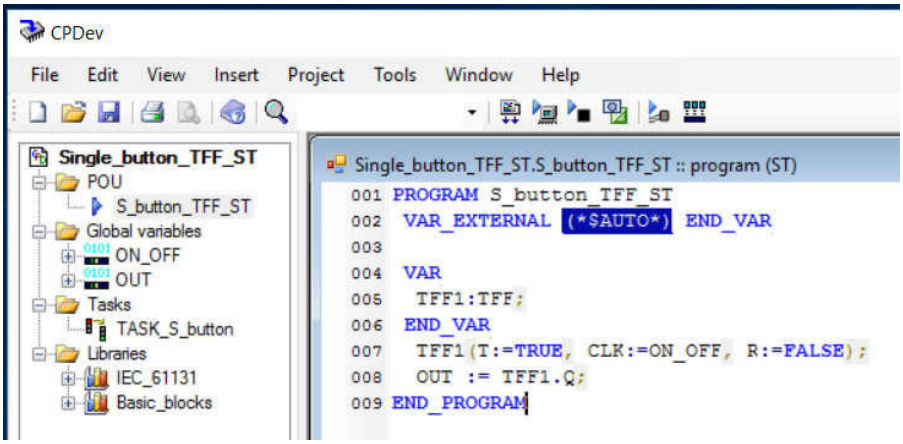
Przypuśćmy, że LED CapsLock ma być ustawiany przez zmienną LIGHT z *Blinking_light_ST*. Konfigurowanie rozpoczyna się tak jak poprzednio, tzn. od zaznaczenia *I/O mapping* w tej jednostce. Teraz jednak w oknie menedżera należy wybrać zakładkę *Binary outputs*, a potem w kolumnie *Variable* ustawić LIGHT. Następnie w kolumnie *Adapter* wybieramy *Keyboard Led Adapter*, a w kolumnie *Port* LED Caps (rys. 9). Po restarcie projektu *Blinking_light_ST* dioda CapsLock rozpoczyna powolne pulsowanie (pod warunkiem, że klawiatura je wspiera; zaleca się potem deaktywować adapter, aby nie zakłócał normalnego funkcjonowania klawiszy). Dla kart NI USB 6008 i RT_DAC/USB dostępne są dodatkowo zakładki wejść binarnych i analogowych, przy czym *Port* wybiera numer końcówki (pin).



Rys. 9. Konfiguracja ustawienia LEDa CapsLock przez zmienną LIGHT

6. Rozproszone WinControllery

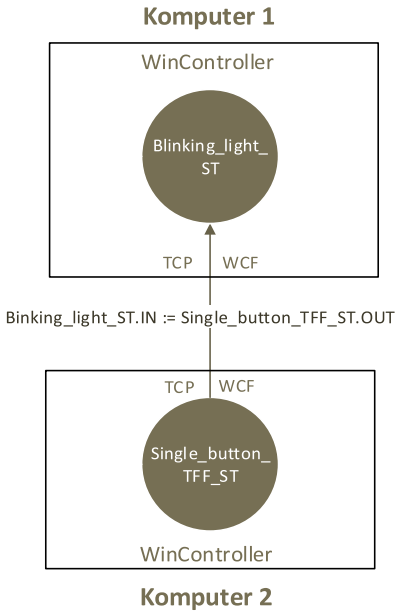
Skonfigurujemy jeszcze najprostszy system rozproszony złożony z dwu komputerów, z których pierwszy, lokalny, będzie nadal wykonywał projekt *Blinking_light_ST*, natomiast drugi, zdalny, wykonywał nieznacznie zmieniony projekt *Single_button_TFF_ST* pokazany na rys. 10. Zamiast detektora R_TRIG i bramki XOR (rys. 3) zawiera on przerzutnik TFF z biblioteki *Basic_blocks* [7], ale zmienne ON_OFF i OUT pozostały niezmienione.



Rysunek 10. Projekt *Single_button_TFF_ST*: ustawianie i zerowanie wyjścia OUT przez przerzutnik TFF

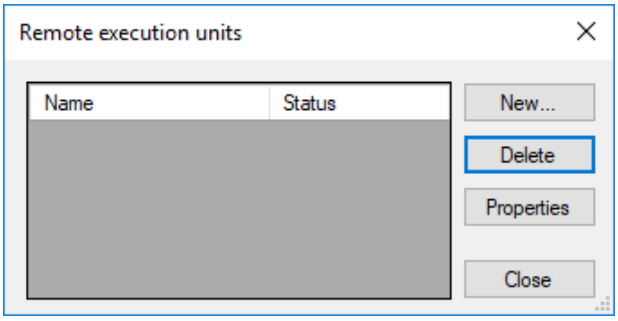
Strukturę tworzonego połączenia przedstawiono na Rys. 11. *Komputer 1* (lokalny) wykonuje projekt *Blinking_light_ST*, zaś *Komputer 2* (zdalny) projekt *Single_button_TFF_ST*. Zmienna IN lokalnej jednostki *Blinking_light_ST* ma być połączona ze zmienną OUT zdalnej jednostki *Single_button_TFF_ST*, co symbolizuje strzałka na Rys. 11. Oba komputery muszą znajdować się w tej samej sieci. Jak wspomniano wcześniej, komunikacja przebiega protokołem TCP/IP z wykorzystaniem

mechanizmu WCF. Numer portu używanego do komunikacji ustala się przy instalacji WinControllera.



Rys. 11. Połączenie projektów *Blinking_light_ST*, *Single_button_TFF_ST* w systemie rozproszonym

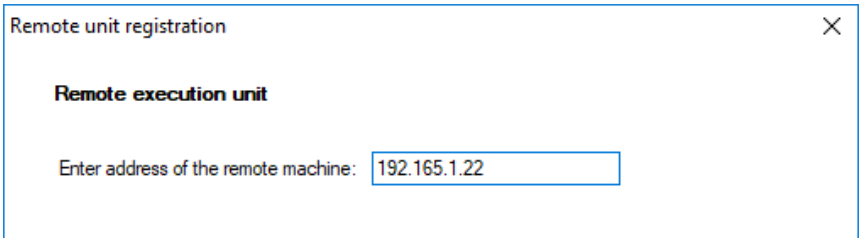
Postępując podobnie jak w przypadku połączenia lokalnego, w *Blinking_light_ST* należy zaznaczyć *I/O mapping*, w oknie menedżera wybrać zakładkę *External variables*. Teraz jednak należy przed połączeniem zmiennych zarejestrować w WinControllerze lokalnym zdalną jednostkę wykonującą projekt *Single_button_TFF_ST*. W tym celu należy kliknąć *Remote units* (por. rys. 6). Pojawia się wtedy główne okno konfiguratora zdalnych jednostek *Remote execution units* pokazane rys. 12. Jest ono puste, gdyż na razie nie skonfigurowano żadnych zdalnych jednostek.



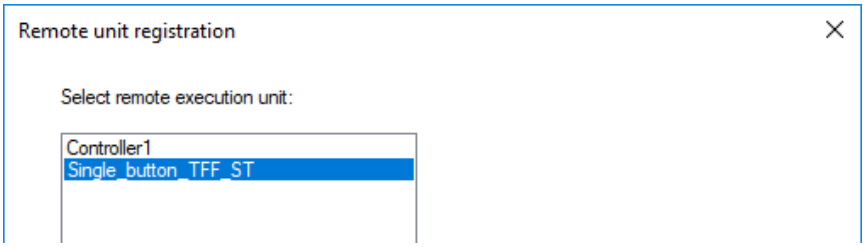
Rys. 12. Konfigurator *Remote execution units*

Naciśnięcie *New* w tym oknie otwiera pierwsze z trzech wewnętrznych okien konfiguratora, w którym należy wpisać adres IP komputera zdalnego (rys. 13a). W drugim oknie (rys. 13b) wybiera się jednostkę wykonawczą komputera zdalnego, czyli tutaj *Single_button_TFF_ST*, co można uczynić, gdy jest on aktywny (widoczny w sieci). Trzecie okno (rys. 13c) służy do zatwierdzenia (lub zmiany) domyślnej nazwy zdalnej jednostki, na którą składa się adres IP (bez kropek), dwukropek oraz nazwa tej jednostki w zdalnym komputerze, tj. *192165122:Single_button_TFF_ST*.

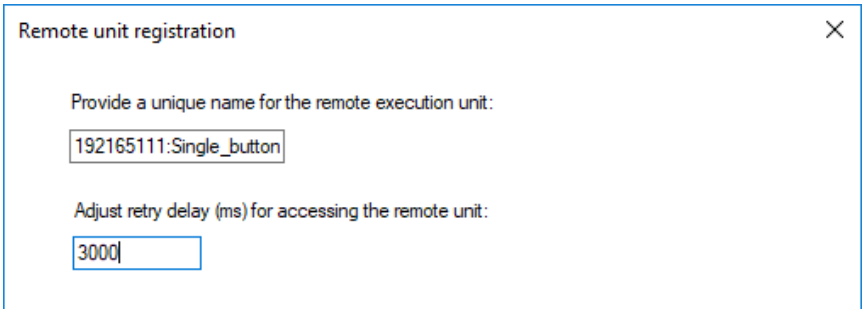
a)



b)



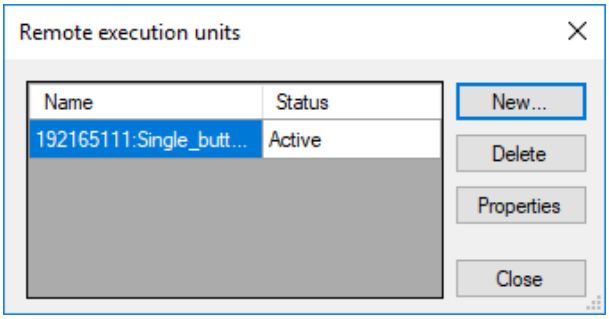
c)



Rys. 13. Okna definiowania zdalnej jednostki wykonawczej: a) adres IP, b) nazwa jednostki w komputerze zdalnym, c) nazwa zdalnej jednostki w komputerze lokalnym

Po zatwierdzeniu nazwy na liście konfiguratora *Remote execution units* pojawia się nowa zdalna jednostka wykonawcza, do której zmiennych komputer lokalny ma dostęp (rys. 14). W ten sam sposób można zarejestrować inne zdalne jednostki, do których zmiennych komputer lokalny może się odwoływać. Za pomocą kolumny *Status* można określić, które zdalne jednostki aktualnie pracują (*Active*). Zdalna jednostka jest niedostępna (*Inactive*), gdy zdalny komputer jest wyłączony, zdalna usługa WinControllera nie działa lub nie odpowiada, albo jednostka została z niego usunięta.

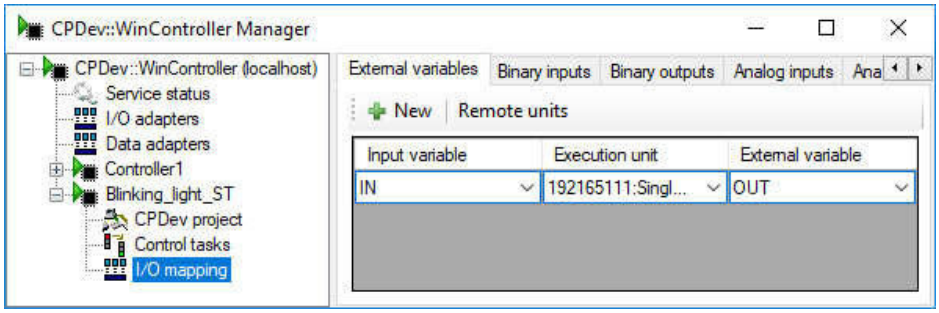
W takiej sytuacji powiązanie zmiennych nie będzie realizowane. WinController będzie jednak sprawdzał status zdalnej jednostki co ustalony czas (3000ms na rys. 13c) i gdy tylko stanie się dostępna automatycznie przywróci powiązanie.



Rys. 14. Konfigurator *Remote execution units* ze skonfigurowaną zdalną jednostką wykonawczą

Teraz należy wrócić do menedżera i dla *I/O mapping* jednostki *Blinking_light_ST* w kolumnie *Input variable* wybrać IN, w *Execution unit* zdalną jednostkę *192165122:Single_button_TFF_ST*, a w *External variable* zmienną OUT, jak to pokazano na rys. 15. Ostatnią czynnością jest restart jednostki *Blinking_light_ST*, której *I/O mapping* uległo zmianie. Warto dodać, że w kolumnie *Execution unit*, oprócz jednostki zdalnej, pozostaje jeszcze do wyboru lokalna jednostka *Controller1* (chyba, że wcześniej została usunięta).

Przy tak skonfigurowanym połączeniu zmienna IN projektu *Blinking_light_ST* ustawiana jest automatycznie na podstawie aktualnej wartości zmiennej OUT w zdalnie wykonywanym projekcie *Single_button_TFF_ST*.

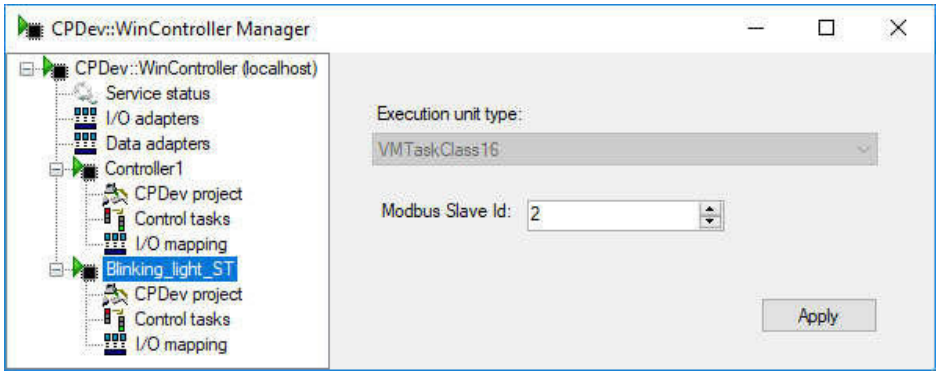


Rys. 15. Połączenie wejścia IN z wyjściem OUT zdalnej jednostki wykonawczej

Warto zauważyć, że w strukturze połączenia zdalnego rozważanej w przykładzie i pokazanej na rys. 11, WinController uruchomiony na *Komputerze 1* (określany tutaj jako lokalny) pełni rolę *klienta*, który pobiera wartości zmiennych z *serwera* (WinControllera na zdalnym *Komputerze 2*). Można ten schemat określić także jako *master-slave*, gdzie lokalny WinController jest *masterem*.

Należy dodać, że dla danego projektu w zakładce *External variables* (rys. 15) konfiguruje się wyłącznie zmienne wejściowe, czyli takie, których wartość jest ustawiana lokalnie. W koncepcji systemu rozproszonego opartego o WinController nie przewidziano możliwości ustawiania wartości w zdalnym projekcie. Argumentem przemawiającym za takim ograniczeniem jest, oprócz względów bezpieczeństwa, spójność i uproszczenie zarządzania strukturą systemu rozproszonego. W sytuacji gdy w projekcie występują zarówno zmienne wejściowe jak i wyjściowe odwołujące się do zdalnego projektu, ich konfigurację należy przeprowadzić osobno dla każdego projektu, przy czym wyjścia należy uwzględnić przy konfiguracji wejść w drugim projekcie. Przy takiej konfiguracji systemu rozproszonego komunikacja jest dwukierunkowa, a jeden WinController działa zarówno jako *master* jak i *slave*.

Warto jeszcze wspomnieć o komunikacji WinControllera ze stacją operatorską, do czego służy protokół Modbus TCP i ewentualnie OPC (por. rys. 1). Modbus TCP jest zaimplementowany jako *slave* udostępniający funkcje odczytu 1, 3 i zapisu 5, 6, 15, 16. Wystarczy to dla pakietów SCADA, np. takich jak InTouch. Modbus TCP WinControllera jest stosunkowo elastyczny, bo każda jednostka wykonawcza posiada własny identyfikator (numer, rys. 16), co pozwala współbieżnie obsługiwać obrazy InToucha. Interfejs OPC jest tymczasem tylko jeden. Należy przypomnieć, że obsługę komunikacji Modbus RTU (RS-485) przewidziano dla *Data adapters*.



Rys. 16. Identyfikator jednostki wykonawczej w protokole Modbus TCP

7. Podsumowanie

Na przykładzie dwu prostych projektów przygotowanych w środowisku CPDev przedstawiono tryb konfiguracji oprogramowania WinController dla komputera PC, który ma wykonywać te projekty. WinController jest oprogramowaniem *runtime* implementowanym jako usługa *Windows service*, z przeznaczeniem dla komputera nadrzędnego w rozproszonym systemie sterowania DCS średniej wielkości, w szczególności do sterowania nadrzędnego, monitorowania i alarmowania. Projekty są wykonywane przez oddzielne jednostki WinControllera, wymieniające dane przez połączenie zmiennych globalnych zadeklarowanych w projektach. Dotyczy to również jednostek umieszczonych w różnych komputerach systemu. Karty wejścia/wyjścia

obsługiwane są za pomocą właściwych adapterów I/O, zaś sterowniki lokalne i inne urządzenia obiektowe dołącza się poprzez moduł *Data adapters*, z interfejsami przygotowanymi indywidualnie dla konkretnych rozwiązań sprzętowych i protokołów komunikacyjnych. Do komunikacji ze stacją operatorską służy Modbus TCP.

Bibliografia

[1] IEC 61131–3 – Programmable controllers – Part 3: Programming languages, 2013.
[2] D.Rzońca, J.Sadolewski, A.Stec, Z.Świder, B.Trybus, L.Trybus: Programming controllers in Structured Text language of IEC 61131-3 standard, Journal of Applied Computer Science,16,1,49-67,2008.
[3] Z.Hajduk, B.Trybus, J.Sadolewski, Architecture of FPGA Embedded Multiprocessor Programmable Controller, Industrial Electronics, IEEE Transactions on ,62, 5, 2952-2961,2015.
[4] B. Trybus: Development and implementation of IEC 61131-3 virtual machine, Theoretical and Applied Informatics, 23, 1, 21-35, 2011.
[5] M. Jamro, D. Rzońca: Automatic Connections in IEC 61131-3 Function Block Diagrams. Federated Conference on Computer Science and Information Systems (FedCSIS), Kraków, 463-469, 2013.
[6] D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus: LD Graphic Editor Implemented in CPDev Engineering Environment. In: Advances in Intelligent Systems and Computing, Automation 2017, 178-185, 2017.
[7] D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus: CPDev engineering environment for control programming. XIX Krajowa Konferencja Automatyki, Kraków, 18-21.06, 2017 (w druku).
[8] CODESYS, 3S-Smart Software Solutions GmbH, http://www.codesys.com.
[9] J.R. Olivas Mendoza Developing Windows Services Succinctly, Syncfusion Inc., 2015.
[10] Wonderware InTouch, Schneider Electric Software, http://software.invensys.com/products/wonderware/hmi-and-supervisory-control/intouch/.

Rozdział 7

Wykorzystanie języka SFC pakietu CPDev w procesach sterowania sekwencyjnego

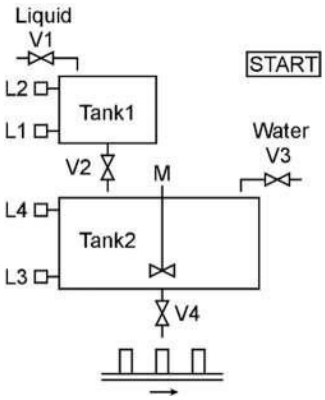
1. Wprowadzenie

Oprogramowanie CPDev (*Control Program Developer*) [1] jest środowiskiem inżynierskim do programowania sterowników PLC zgodnie z normą PN-EN 61131-3 [2-4]. Opracowano je z myślą użycia w niewielkich i średnich systemach bazujących na mikrokontrolerach AVR i ARM, a także zbudowanych w oparciu o platformy sprzętowe x86 i FPGA [5]. Oprogramowanie umożliwia korzystanie z wszystkich pięciu języków (ST, LD, FBD, IL, SFC), przy czym wiodącym jest ten pierwszy, do którego konwertowane są diagramy utworzone w edytorach graficznych. Podobne rozwiązanie można spotkać także w innych pakietach oprogramowania sterowników PLC [6], choć wtedy konwersja wykonywana jest zwykle do języka IL, a nie ST.

Język SFC (*Sequential Function Chart*), wywodzący się z sieci Petriego i języka GRAFCET, jest szczególnie użyteczny w przypadku programowania procesów sterowania sekwencyjnego. Umożliwia on podział programu na mniejsze części (etapy/kroki), których wykonywanie uzależnia się od uprzedniego spełnienia warunków ich rozpoczęcia. Dzięki temu poprawia się przejrzystość kodu oraz jego efektywność, gdyż wykonywane są tylko niezbędne fragmenty kodu, a nie całość. Należy także zaznaczyć, że poszczególne kroki i warunki przejścia między nimi są programowane w pozostałych czterech językach normy, choć w przypadku CPDev jest to język ST.

2. Przykładowy proces sterowania sekwencyjnego

Górny zbiornik napełniany jest płynem rozcieńczanym następnie wodą w dolnym zbiorniku (rys. 1). Po krótkim mieszaniu otrzymany roztwór rozlewa się do pojemników przesuwających się na taśmie.



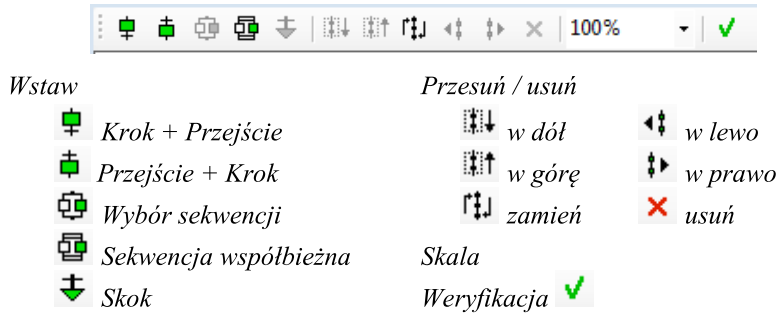
Rys. 1. Obiekt sterowania

Proces rozpoczyna się od naciśnięcia przycisku START. Po napełnieniu górnego zbiornika (Tank 1) przez zawór V1 przynajmniej do poziomu L1, otwierany jest zawór opróżniający V2 oraz zawór z wodą V3 w celu napełnienia dolnego zbiornika (Tank 2). Napełnianie obu zbiorników prowadzone jest równocześnie, przy czym jednorazowe napełnienie dolnego zbiornika wymaga kilkukrotnego napełnienia zbiornika górnego. Gdy poziom roztworu w dolnym zbiorniku osiągnie L4, zawory V2 i V3 są zamykane a mieszanie włączane na 10 sekund. Jeśli zbiornik górny nie jest pełny, kontynuuje się jego napełnianie do poziomu L2. Po wymieszaniu roztwór z dolnego zbiornika rozlewa się do pojemników w cyklach: 3-sekundowe napełnianie i 2-sekundowa przerwa. Proces kończy się po opróżnieniu dolnego zbiornika. Naciśnięcie przycisku START rozpoczyna sekwencję od początku.

3. Tworzenie diagramu SFC

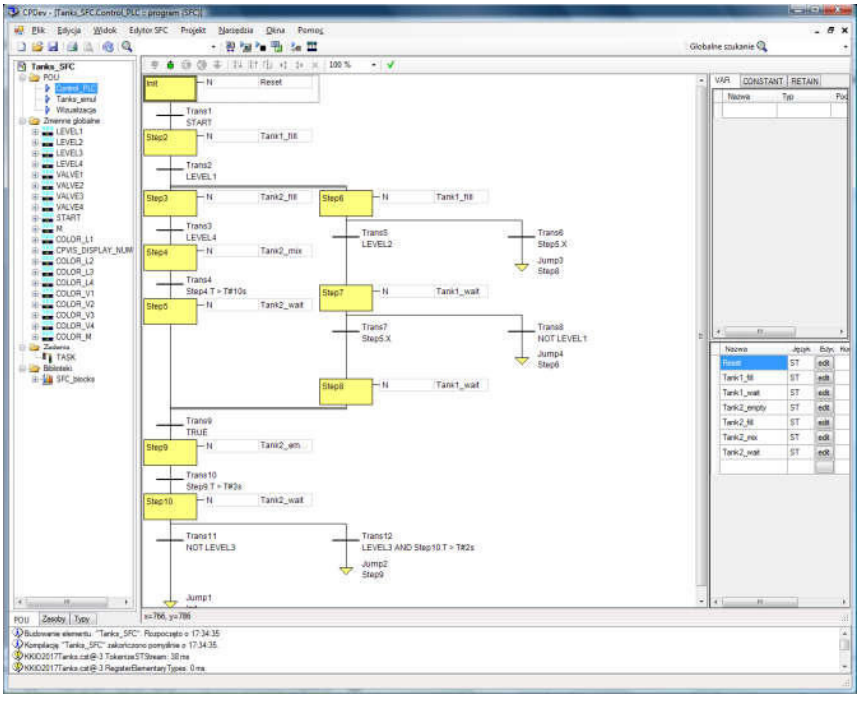
Po utworzeniu nowego projektu i dodaniu programu lub bloku funkcjonalnego SFC w polu edycji pojawia się sekwencja początkowa składająca się z kroku początkowego (*Init*) z pustą akcją (*Action*), przejścia/tranzycji (*Trans1*) z warunkiem TRUE oraz skokiem (*Jump1*) do kroku początkowego. Kolejne elementy dodawane są poprzez ich wybór z menu programu lub paska narzędziowego (rys. 2). Zgodnie z regułami SFC każdy krok (*Step*) musi być rozdzielony tranzycją (*Transition*). W celu usprawnienia procesu, do diagramu wstawia się każdorazowo sekwencję składającą się jednocześnie z kroku i tranzycji, przy czym kolejność zależy od miejsca ich dodania. Gałąź sekwencji współbieżnej (*Simultaneous sequence*) dodaje się do kroku, a wybór sekwencji (*Sequence selection*) do tranzycji. W tym przypadku każdorazowo wstawiane jest rozejście i zejście.

Kroki, tranzycje i skoki konfigurowane są w oknach właściwości. W przypadku kroku użytkownik może zmienić jego nazwę, dodać komentarz, skojarzyć akcje i określić jej kwalifikator. W oknie tranzycji oprócz nazwy i komentarza definiuje się wyrażenie określające warunek przejścia do następnego kroku.



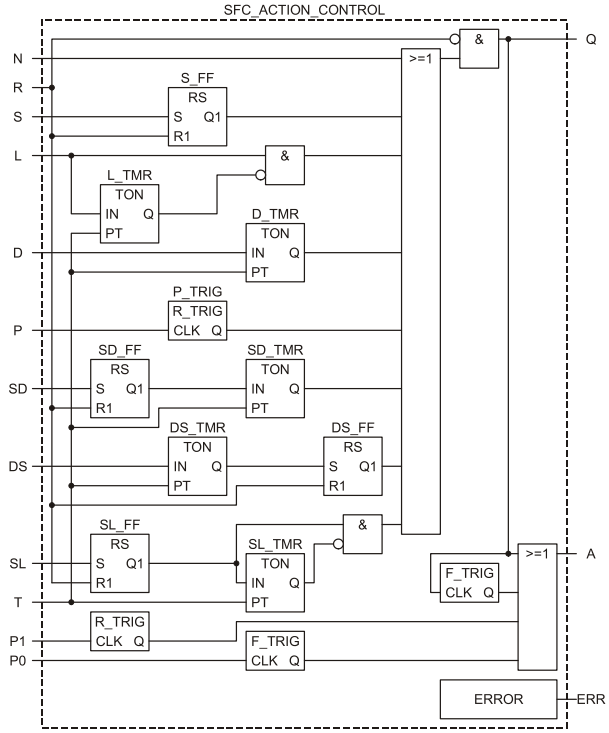
Rys. 2. Pasek narzędziowy edytora SFC

Program implementujący rozwiązanie postawionego zadania (rys. 3) zawiera prawie wszystkie typowe konstrukcje języka SFC. Oprócz zwykłych kroków i przejść są sekwencje współbieżne reprezentowane przez gałęzie umieszczone pomiędzy podwójnymi liniami poziomymi oraz kilkukrotnie zastosowany wybór sekwencji umożliwiający przeskok do wybranego kroku w diagramie. Z krokami skojarzone są akcje będące fragmentami programów w języku ST. W omawianym przykładzie wszystkie one opatrzone są kwalifikatorem N, tzn. wykonywane jedynie w trakcie aktywności kroku. Przejścia pomiędzy krokami są wykonywane po spełnieniu odpowiedniego warunku podawanego w postaci wyrażenia języka ST.



Rys. 3. Diagram SFC w środowisku CPDev

Jak widać, wykorzystanie języka SFC znacznie upraszcza programowanie procesów sekwencyjnych, zwłaszcza w stosunku do pozostałych języków standardu IEC 61131-3. Niestety dzieje się to kosztem większego rozmiaru programu oraz pamięci. Wynika to z konieczności implementacji bloku `SFC_ACTION_CONTROL` (rys. 4) odpowiedzialnego za aktywność akcji [7]. Norma zaleca, aby blok ten obsługiwał aż 11 kwalifikatorów (N, R, S, L, D, P, SD, DS, SL, P1, P0), z których najczęściej do aktywacji danej akcji używany jest tylko jeden. Ponieważ liczba instancji deklarowanych w programie SFC wynika z łącznej liczby akcji wszystkich kroków diagramu, wynika z tego, że znaczna część kodu nie jest wykorzystywana. Problem ten można byłoby rozwiązać przez opracowanie wersji biblioteki z blokiem obsługującym zredukowaną liczbę kwalifikatorów, jednak biorąc pod uwagę dostępność coraz większej ilości pamięci programu i danych w nowo produkowanych sterownikach, wydaje się to zbyteczne.



Rys. 4. Struktura wewnętrzna bloku `SFC_ACTION_CONTROL` [7]

Zgodnie z normą zabrania się jednoczesnego wywoływania tej samej akcji w kilku krokach z kwalifikatorami uzależnionymi od odmierzania czasu. Wynika to z tego, że blok ma jedno, wspólne wejście T dla elementów L_TMR, D_TMR, SD_TMR, DS_TMR i SL_TMR, więc nie może ono w tym samym momencie przyjmować wielu wartości. Ponieważ nie każdy użytkownik musi być tego świadomym, a jednocześnie dla ułatwienia wykrywania niedozwolonego użycia, blok `SFC_ACTION_CONTROL` został rozszerzony w stosunku do normy o wyjście `ERR`.

W środowisku CPDev program sterowania opracowany w postaci diagramu SFC jest przed kompilacją poddawany konwersji do języka ST. Wygenerowany kod składa się z czterech zasadniczych części (rys. 5):

- deklaracji zmiennych,
- aktywacji kroku/kroków dla danego cyklu obliczeniowego,
- wykonania programu aktywnych akcji,
- sprawdzenia warunków przejścia i ustawienia znaczników aktywacji następnych kroków.

```
001 PROGRAM Control_PLC
002 VAR_EXTERNAL (*AUTO*) END_VAR
003 VAR
004   (* ----- System variables ----- *)
005   Init : SFC_STEP;
006   Step2 : SFC_STEP;
007   Step3 : SFC_STEP;
008   Step4 : SFC_STEP;
009   Step5 : SFC_STEP;
010   Step6 : SFC_STEP;
011   Step7 : SFC_STEP;
012   Step8 : SFC_STEP;
013   Step9 : SFC_STEP;
014   Step10 : SFC_STEP;
015   _Init : BOOL := TRUE;
016   _Step2 : BOOL;
017   _Step3 : BOOL;
018   _Step4 : BOOL;
019   _Step5 : BOOL;
020   _Step6 : BOOL;
021   _Step7 : BOOL;
022   _Step8 : BOOL;
023   _Step9 : BOOL;
024   _Step10 : BOOL;
025   _ACTION_T : TIME;
026   Reset : SFC_ACTION_CONTROL;
027   Tank1_fill : SFC_ACTION_CONTROL;
028   Tank1_wait : SFC_ACTION_CONTROL;
029   Tank2_empty : SFC_ACTION_CONTROL;
030   Tank2_fill : SFC_ACTION_CONTROL;
031   Tank2_mix : SFC_ACTION_CONTROL;
032   Tank2_wait : SFC_ACTION_CONTROL;
033   (* ------ User variables ------ *)
034 END_VAR
```

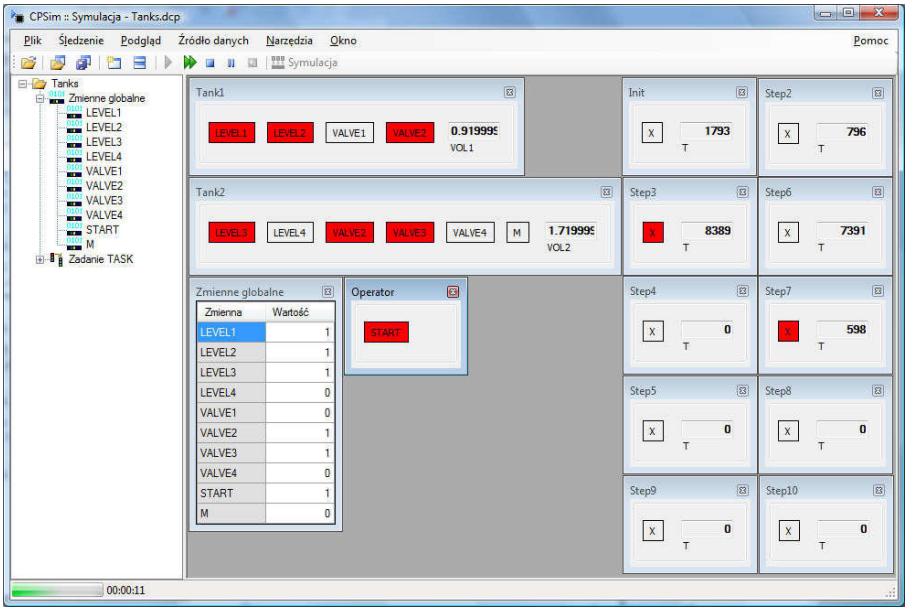
```
036 Init(IN:=_Init);
037 Step2(IN:=_Step2);
038 Step3(IN:=_Step3);
039 Step4(IN:=_Step4);
040 Step5(IN:=_Step5);
041 Step6(IN:=_Step6);
042 Step7(IN:=_Step7);
043 Step8(IN:=_Step8);
044 Step9(IN:=_Step9);
045 Step10(IN:=_Step10);
046
047 Reset(N:=Init.X);
048 IF Reset.A THEN
049 (* /---- Reset (begin) ----\ *)
050   VALVE1 := FALSE;
051   VALVE2 := FALSE;
052   VALVE3 := FALSE;
053   VALVE4 := FALSE;
054 (* \----  Reset (end)  ----/ *)
055 END_IF
056 Tank1_fill(N:=Step2.X OR Step6.X);
057 IF Tank1_fill.A THEN
058 (* /---- Tank1_fill (begin) ----\ *)
059   VALVE1 := TRUE;
060 (* \----  Tank1_fill (end)  ----/ *)
061 END_IF

103 IF Init.X THEN
104     IF START THEN
105         _Init := FALSE;
106         _Step2 := TRUE;
107     END_IF
108 END_IF
109 IF Step2.X THEN
110     IF LEVEL1 THEN
111         _Step2 := FALSE;
112         _Step3 := TRUE; _Step6 := TRUE;
113     END_IF
114 END_IF
```

Rys. 5. Fragmenty kodu w języku ST

Zmienne o identyfikatorach tożsamych z nazwami kroków lub akcji są instancjami bloków funkcjonalnych biblioteki *SFC_blocks*. Pierwsze (`SFC_STEP`) udostępniają informację o stanie i czasie trwania odpowiedniego kroku, drugie (`SFC_ACTION_CONTROL`) kontrolują wykonywanie poszczególnych akcji. Pomocnicze zmiennie typu BOOL pełnią funkcję znaczników uaktywniających odpowiedni krok lub kroki w następnym cyklu obliczeniowym. Bloki biblioteki SFC importowane są automatycznie przy dodawaniu do projektu nowego programu.

4. Symulacja w CPSim

Symulator CPSim służy do testowania poprawności działania programów CPDev. Zmienne są wyświetlane w postaci listy lub elementów na panelach (rys. 6). Monitorowane mogą być zarówno zmienne globalne projektu jak i lokalne jednostek POU (programów, bloków funkcjonalnych oraz funkcji). W przypadku języka SFC szczególnie użyteczna jest możliwość grupowania wyświetlanych na panelach wartości związanych z danym krokiem lub akcją. Zmienne typu `BOOL` przedstawiane są jako prostokąty zmieniające swój kolor, co poprawia czytelność i zwraca uwagę w przypadku ich zmiany. Wartości zmiennych pozostałych typów są prezentowane w postaci wartości liczbowych. W omawianym przykładzie osobne panele zostały utworzone dla każdego z kroków schematu SFC. Można w ten sposób monitorować ich stan (*step_name.X*) oraz czas aktywności (*step_name.T*).



Rys. 6. Przykładowe okno symulacji CPSim

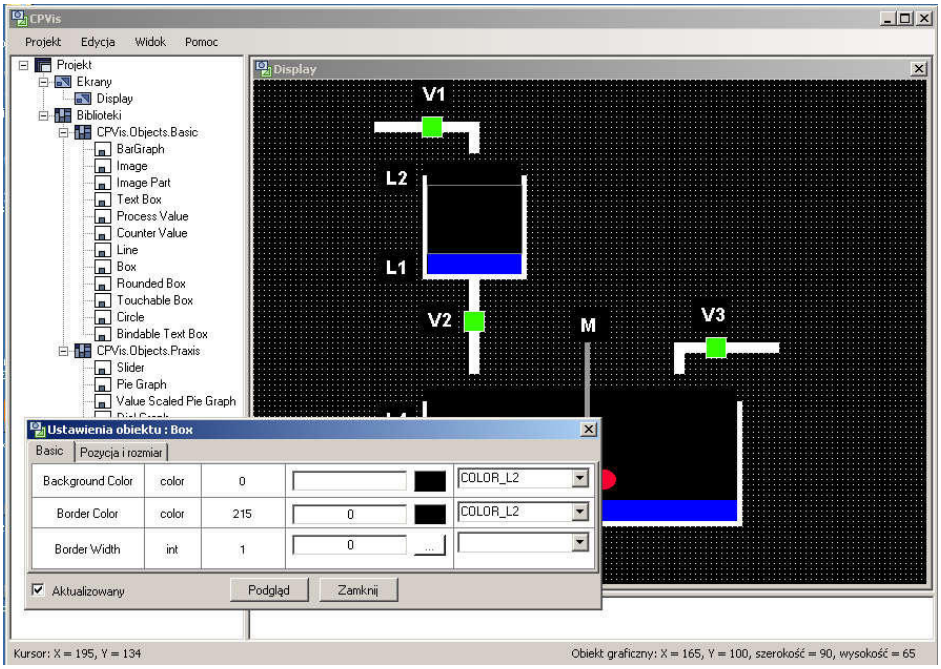
Symulator CPSim umożliwia także monitorowanie zmiennych w trybie komunikacji ze sterownikiem. W takim przypadku wyświetlane wartości dotyczą rzeczywistego procesu, którym steruje PLC, a nie jego symulacji.

5. Graficzny interfejs operatorski w CPVis

Program sterowania przedstawiony w sekcji 3 uzupełniono o prostą wizualizację, przygotowaną w programie CPVis. CPVis [8, 9] jest składnikiem pakietu inżynierskiego CPDev umożliwiającym tworzenie graficznych interfejsów operatorskich HMI (*Human-*

Machine Interface). Ekrany wizualizacyjne tworzone są w graficznym edytorze CPVis z użyciem kontrolek będących elementami graficznymi, prostymi (np. linia, prostokąt) lub złożonymi (np. bargraf, wykres).

Projektant interfejsu, tworząc ekrany wizualizacyjne, może z poszczególnymi kontrolkami powiązać zmienne globalne z programu sterowania, uzależniając bieżący wygląd kontrolki (np. kolor wypełnienia, punkt na wykresie) od aktualnej wartości zmiennej. Wizualizacja, odbywająca się na PC lub dedykowanym panelu HMI, wykorzystuje bieżące wartości odczytywane z maszyny wirtualnej CPDev, działającej na sterowniku. Edytor CPVis podczas tworzenia opisywanej przykładowej wizualizacji pokazano na rys. 7. Umieszczone na pierwszym planie okno dialogowe pozwala na zdefiniowanie koloru wypełnienia prostokąta reprezentującego ciecz w górnej części górnego zbiornika (poziom L2). Obecnie kolor ten powiązano ze zmienną `COLOR_L2`.



Rys.7. Edytor CPVis

Niejednokrotnie parametry wymagane przez wizualizację muszą być obliczane w sterowniku na podstawie wartości zmiennych programu sterowania. Zazwyczaj wymaga to przygotowania osobnego programu wizualizacyjnego (lub kilku takich programów) w jednym z języków normy PN-EN 61131-3. Podkreślić należy, że język użyty do opracowania programu wizualizacyjnego nie musi być tym samym, który wykorzystano przy implementacji programu sterowania. W opisywanym przykładzie program sterowania zaimplementowano wykorzystując język SFC, a program wizualizacyjny w języku ST.

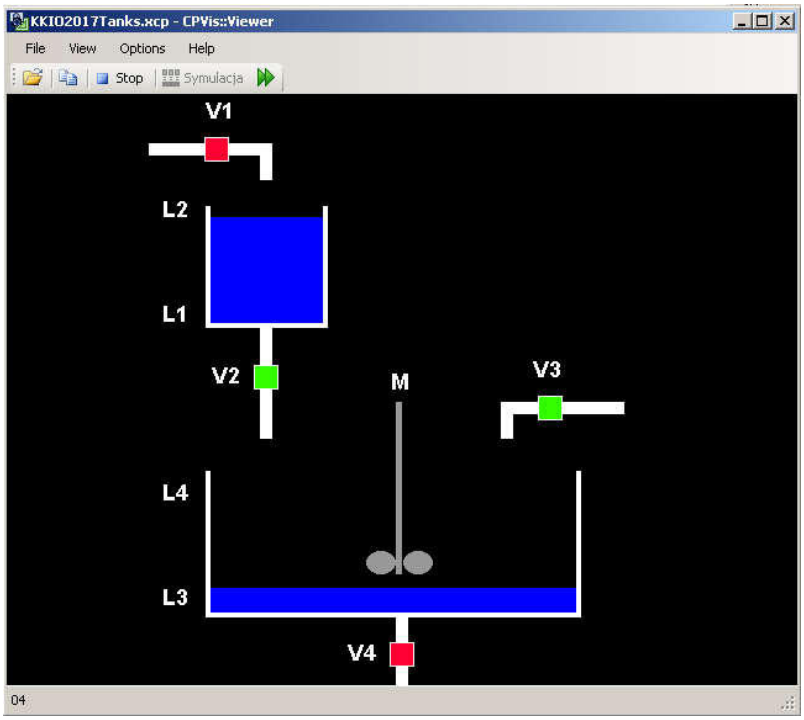
Wizualizacja opisywanego przykładu bazuje na zmiennych globalnych `LEVEL1` – `LEVEL4`, reprezentujących czujniki poziomu w zbiornikach, `VALVE1 – VALVE4` oznaczających stan zaworów i `M` – mieszadło. Na ich podstawie obliczane są wartości pomocniczych zmiennych wizualizacyjnych. Przykładowo obliczenie koloru wypełnienia prostokąta reprezentującego ciecz w górnej części zbiornika na podstawie wartości z czujnika poziomu L2 zapisano jako:

```
if LEVEL2 then COLOR_L2 := 5; else COLOR_L2 := 0; end_if
```

Należy zauważyć, że w opisywanym przykładzie każdy ze zbiorników wyposażony jest w dwa binarne czujniki poziomu dające jedynie informację czy dany poziom jest osiągnięty, czy też nie. Pozwala to na określenie jedynie trzech poziomów cieczy w zbiornikach, tj. zbiornik pusty (oba czujniki w stanie niskim), zbiornik częściowo napełniony (dolny czujnik w stanie wysokim, górny w niskim) oraz zbiornik pełny (oba czujniki w stanie wysokim). W takim systemie wystarczająca jest wizualizacja poziomu cieczy oparta na dwóch prostokątach, których kolor wypełnienia zmieniany jest na czarny (dana część zbiornika pusta) bądź niebieski (dana część zbiornika pełna). Bardziej precyzyjna wizualizacja poziomu cieczy, oparta na bargrafie, wymagałaby czujnika analogowego, podającego stopień wypełnienia zbiornika.

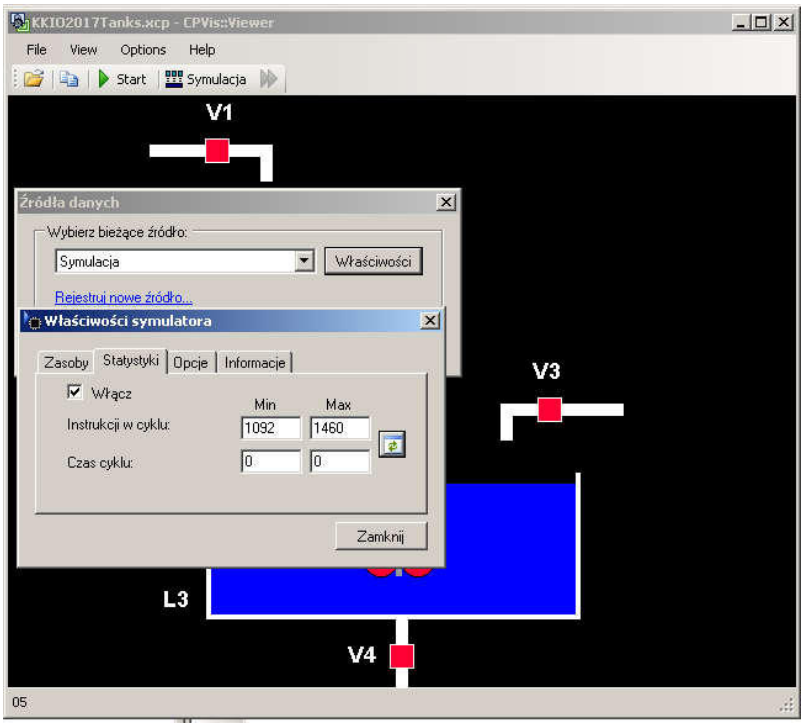
Oprócz zmiennych poziomu, zaworów i mieszadła do obsługi panelu operatorskiego zdefiniowano zmienną globalną `START_BUTTON`. Reprezentuje ona stan przycisku uruchamiającego cykl napełniania. Przycisk ten może być rzeczywistym włącznikiem sprzętowym podłączonym do wejścia binarnego modułu I/O, bądź też przyciskiem wirtualnym, powiązanym ze zdefiniowanym obszarem panelu dotykowego HMI, bądź wybranym klawiszem na klawiaturze.

Wizualizowany jest poziom cieczy w zbiornikach, stan zaworów i praca mieszadła. Na rys. 8 pokazano działanie opracowanego ekranu wizualizacyjnego w programie CPVis::Viewer.



Rys. 8. Okno CPVis::Viewer

W poprzedniej sekcji podano, że wartości zmiennych obserwowane w programie CPSim mogą pochodzić z programu sterowania działającego na symulatorze, bądź też mogą być odczytywane na bieżąco z rzeczywistego sterownika PLC. Podobnie podczas wizualizacji źródłem danych może być symulator, bądź też sterownik, sprzętowy lub programowy (WinController opisany w kolejnej sekcji). Komunikacja realizowana jest przez dodatkową bibliotekę, moduł źródła danych. Implementowany protokół komunikacyjny i zastosowane łącze danych zależy od wykorzystywanego źródła, przykładowo dla sprzętowego sterownika PLC może to być protokół Modbus (RTU lub TCP), a dla WinControllera WCF (*Windows Communication Foundation*). Modułowe rozwiązanie jest elastyczne, w razie potrzeby inżynier wdrażający pakiet CPDev na kolejnej platformie sprzętowej może łatwo zaimplementować obsługę nowego protokołu komunikacyjnego we własnej bibliotece, którą można dołączyć do pakietu CPDev i wykorzystywać w poszczególnych programach składowych. Poza bieżącymi wartościami zmiennych źródło danych może udostępniać dodatkowe informacje, np. statystyczne. Przykład prostych statystyk dotyczących liczby instrukcji w cyklu i osiągniętego czasu obliczeń pojedynczego obiegu pętli sterowania podczas cyklu, zbieranych podczas wizualizacji na symulatorze przedstawiono na rys. 9.



Rys. 9. Statystyki zbierane podczas wizualizacji na symulatorze

6. Środowisko uruchomieniowe

Choć pierwotnie CPDev był projektowany do stosowania w niewielkich systemach sterowania, z biegiem czasu pojawiły się potrzeby uruchamiania większych programów. W szczególności dotyczy to projektów tworzonych w oparciu o język SFC, gdzie każdy krok może zawierać dużą ilość kodu oraz konieczne jest utworzenie i zarządzanie wieloma instancjami bloków obsługi akcji i kroków `SFC_ACTION_CONTROL`, `SFC_STEP` o czym była mowa w p.3. Obsługa graficznego interfejsu operatorskiego HMI dodatkowo zwiększa zapotrzebowanie na pamięć. Wychodząc naprzeciw tym oczekiwaniom maszyna wirtualna CPDev [10], będąca zasadniczym elementem środowiska uruchomieniowego (*runtime*) została rozszerzona, tak aby możliwe było przekroczenie dotychczasowego limitu 64KB (adres 16-bitowy). Obecna teoretyczna granica wynosi aż 4GB pamięci danych i kodu. Uruchamianie takich programów wymaga oczywiście odpowiedniej platformy sprzętowej, przy czym w grę wchodzą procesory przynajmniej 32-bitowe (ARM, x86, itp.).

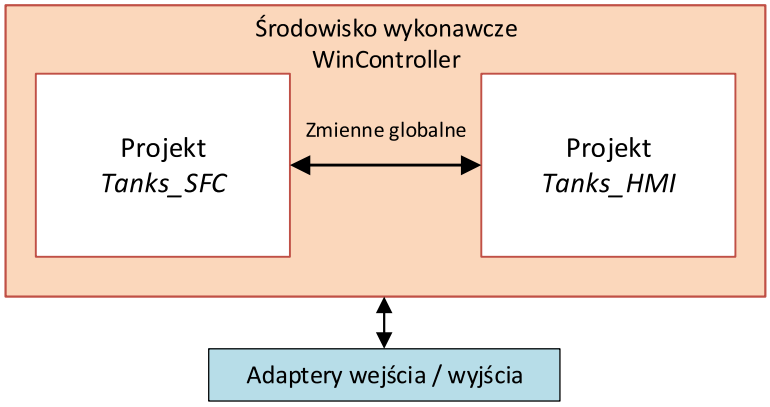
Rozszerzona wersja maszyny wirtualnej CPDev znalazła zastosowanie w nowym środowisku uruchomieniowym dla maszyn o architekturze PC (komputerów sterujących) pracujących pod kontrolą systemu operacyjnego Windows. Środowisko to nazwano WinController i można go postrzegać jako tzw. sterownik programowy (*soft PLC*). Do

istotnych cech stanowiących o architekturze WinControllera należą skalowalność i rozszerzalność. *Skalowalność* oznacza w tym przypadku:

- dostępność dwóch typów maszyn wirtualnych: 16-bitowej oraz 32-bitowej, stosowanych w zależności od rozmiaru programu;
- możliwość wykonywania wielu projektów CPDev jednocześnie.

Każdy z projektów CPDev jest przypisany do tzw. *jednostki wykonawczej* (*execution unit*). Istnieje możliwość utworzenia wielu takich jednostek (wirtualnych sterowników), działających współbieżnie z wykorzystaniem mechanizmów wielozadaniowości systemu operacyjnego. Każda jednostka może pracować z innym cyklem oraz wymieniać dane z innymi jednostkami. Dzięki temu można zintegrować w jednej maszynie sterowanie wieloma różnymi obiektami. Wymiana danych jest dostępna w sieci lokalnej, w której węzły z działającym WinControllerem mogą tworzyć rozproszony system sterowania.

Rozszerzalność dotyczy możliwości doposażenia środowiska WinController o dodatkowe, specjalizowane moduły wejściowo-wyjściowe (tzw. adaptery I/O) i komunikacyjne (adaptery danych). Mogą one obsługiwać karty I/O np. podłączane do komputera za pomocą łącza USB lub dodatkowe protokoły komunikacyjne (Modbus RTU, Profibus DP). Warto dodać, że obsługa protokołów Modbus TCP, OPC oraz WCF (*Windows Communication Foundation*) jest dostępna standardowo.

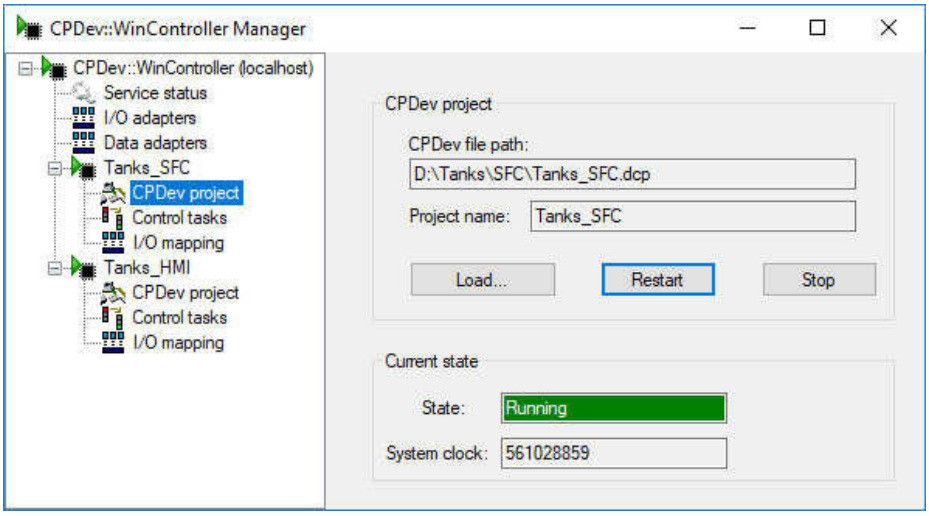


Rys. 10. Jednostki wykonawcze projektów *Tanks_SFC* i *Tanks_HMI*

W omawianym przykładzie sterowania sekwencyjnego napełnianiem zbiorników należy uwzględnić wykonywanie dwóch projektów: *Tanks_SFC* i *Tanks_HMI*. Z tego powodu WinController skonfigurowano tworząc dwie jednostki wykonawcze (rys. 10). Każda z nich będzie działała z właściwym dla siebie cyklem. Jednostka *Tanks_SFC* wykonuje program sterowania sekwencyjnego w języku SFC z p.3. Czas cyklu sterownika wynosi 100ms. Druga jednostka *Tanks_HMI* odpowiada za obsługę panelu operatorskiego opisanego w p. 5. Jak podano, został on utworzony w języku ST, a czas cyklu wynosi w tym przypadku 200ms.

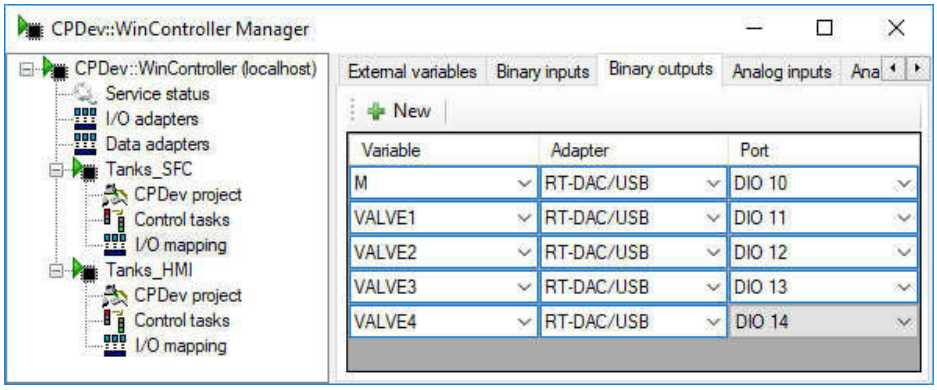
Rysunek 11 przedstawia panel konfiguracyjny jednostki *Tanks_SFC*. Załadowano do niej plik projektu CPDev o tej samej nazwie. W przedstawionej sytuacji jednostka jest już uruchomiona (*Running*). Czas systemowy jest odmierzany dla każdej z jednostek

oddzielnie. Oznacza to, że jednostki mogą być traktowane jako niezależne sterowniki PLC.



Rys. 11. Panel właściwości jednostki wykonawczej projektu *Tanks_SFC*

Sygnały wejściowe i wyjściowe obsługiwane są w WinControllerze za pomocą adapterów I/O (rys. 10). Z wejściami i wyjściami binarnymi kart można skojarzyć zmienne globalne typu `BOOL`, zaś z analogowymi – zmienne typu `REAL`. Rysunek 12 przedstawia panel konfiguracyjny, w którym binarne zmienne wyjściowe (*Binary outputs*) o nazwach `M`, `VALVE1…4` przywiązano do odpowiednich portów karty I/O. Wyjścia te sterują zaworami i mieszalnikiem. W ten sam sposób łączy się z I/O zmienne wyjściowe i analogowe. W prototypie laboratoryjnym zastosowano adaptery dla kart RT-DAC USB (Inteco) oraz NI-DAQ USB 6008 (National Instruments).

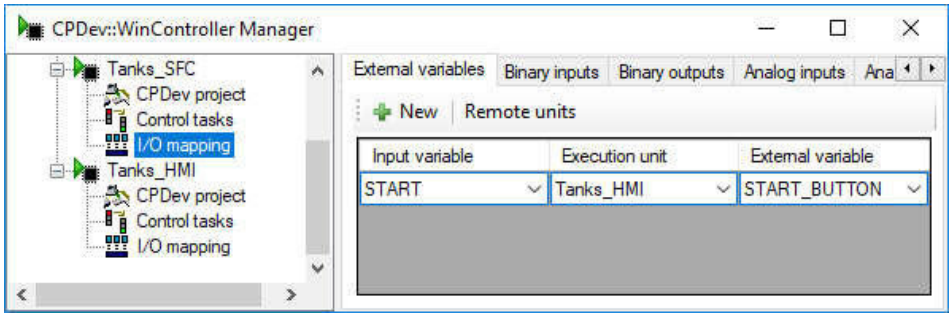


Rys. 12. Przypisanie zmiennych globalnych programu SFC do portów karty I/O

Jednostki wykonawcze *Tanks_SFC* i *Tanks_HMI* wymieniają między sobą dane (rys. 10). Standardowym mechanizmem określonym do tego celu w normie 61131-3 są

zmienne globalne. W przypadku WinControllera oznacza to, że wybrane zmienne jednej jednostki (wejściowe) są ustalane na podstawie wartości zmiennych drugiej jednostki (wyjściowych). Na rys. 13 przedstawiono panel konfiguracyjny takiej wymiany zmiennych (*External variables*) dla projektu SFC. Widać, że przywiązano zmienną `START` projektu *Tanks_SFC* (lokalną) do zmiennej zdalnej `START_BUTTON` projektu *Tanks_HMI*. W ten sposób przycisk zdefiniowany w interfejsie HMI służy do włączania cyklu napełniania w programie SFC. Analogicznie, choć w odwrotnym kierunku przywiązano zmienne binarne określające poziomy i stany zaworów oraz mieszalnika (`LEVEL`, `VALVE`, `M`, nie pokazane na rys. 13). W ten sposób jednostka *Tanks_HMI* może na podstawie wartości tych zmiennych przygotować wygląd ekranu panelu operatorskiego odzwierciedlający stan systemu.

Warto dodać, że skalowalność WinControllera, o której wspomniano wcześniej, może oznaczać w przypadku omawianego przykładu sterowania sekwencyjnego z panelem HMI, że jeden komputer będzie sterował dwoma (lub więcej) instalacjami napełniania zbiorników z rys. 1, obsługiwał panel operatorski oraz ewentualnie sterował innymi obiektami. W takim przypadku każda z jednostek wykonawczych WinControllera będzie wykonywała odpowiedni projekt CPDev w języku SFC, przygotowany dla właściwego obiektu sterowania. Do obsługi paneli HMI można zastosować tyle jednostek ile jest instalacji lub przygotować jeden zbiorczy projekt obsługujący wszystkie na raz, z przełączanymi ekranami. Możliwe jest również przeniesienie obsługi panelu operatorskiego w systemie rozproszonym na inny komputer sieci lokalnej (stację operatorską) poprzez powiązanie zmiennych między zdalnymi WinControllerami.



Rys. 13. Konfigurowanie wymiany zmiennych między projektami SFC i HMI

7. Podsumowanie

Język SFC dostępny w pakiecie CPDev pozwala usprawnić tworzenie oprogramowania sterującego dla procesów sekwencyjnych. Istotnie poprawia czytelność oraz umożliwia niezależne opracowywanie obsługi poszczególnych kroków. Wadą jest narzut związany z koniecznością obsługi bloków odpowiadających za obsługę kroków, aktywność akcji i sporej ilości kwalifikatorów.

Duże programy tworzone w języku SFC z krokami programowanymi w ST wykonywane są przez nowe środowisko uruchomieniowe WinController, skalowalne do potrzeb. Dzięki niemu, oprócz programu sterującego SFC możliwe jest wykonywanie

innych zadań sterujących oraz obsługa panelu HMI. Warto dodać, że w przypadku bardziej zaawansowanych interfejsów operatorskich niż pokazany w przykładzie, obsługa również może być oparta o schemat SFC, określający stany w jakich panel może się znajdować. Program taki mógłby na przykład obejmować wiele ekranów (widoków), przejścia pomiędzy nimi i warunki takich przejść.

Bibliografia

[1] M. Jamro, D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus, L. Trybus, CPDev Engineering Environment for Modeling, Implementation, Testing, and Visualization of Control Software, [in:] R. Szewczyk, C. Zieliński, M. Kaliczyńska (eds.): Advances in Intelligent Systems and Computing vol. 267, Recent Advances in Automation, Robotics and Measuring Techniques, Springer-Verlag, Berlin Heidelberg (2014), 81–90.

[2] Polska norma PN-EN 61131-3 – Sterowniki programowalne. Część 3: Języki programowania, PKN, Warszawa, 2013.

[3] J. Kasprzyk, Programowanie sterowników przemysłowych, WNT, Warszawa, 2006.

[4] K-H. John, M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, Springer-Verlag, Berlin Heidelberg, 2010.

[5] Z. Hajduk, J. Sadolewski, B. Trybus, Architecture of FPGA embedded multiprocessor programmable controller. IEEE Transactions on Industrial Electronics, vol. 62, no. 5 (2015), 2952–2961.

[6] J. Asensio, F.Ortuño, M. Damas, H. Pomares, Industrial Automation Programming Environment with a New Translation Algorithm among IEC 61131□3 Languages Based on the TC6□XML Scheme, International Journal of Automation and Control Engineering, vol. 2, no. 2 (2013), 47–55.

[7] A. Stec, SFC Graphic Editor for CPDev Environment, [in:] R. Szewczyk, C. Zieliński, M. Kaliczyńska (eds.), Advances in Intelligent Systems and Computing vol. 550, Automation 2017 Innovations in Automation, Robotics and Measurement Techniques, Springer, 2017, 186-194.

[8] M. Jamro, B. Trybus, Configurable Operator Interface for CPDev Environment, Pomiary Automatyka Robotyka, R. 17, 2 (2013), 426–431.

[9] M. Jamro, B. Trybus, IEC 61131-3 programmable human machine interfaces for control devices, 2013 6th International Conference on Human System Interactions (HSI), Sopot, 2013, 48–55.

[10] B. Trybus: Development and implementation of IEC 61131-3 virtual machine, Theoretical and Applied Informatics, 23, 1, (2011), 21–35.

Rozdział 8

Optymalizacja skryptu NuSMV modelującego scenariusz zdarzeń zagrażających bezpieczeństwu na lotnisku

1. Wprowadzenie

Tematem tego rozdziału jest analiza bezpieczeństwa naziemnego ruchu samolotów. Zależnie od czasu rozpoczęcia i czasu trwania zdarzeń dotyczących dynamiki samolotu, jego radiowej komunikacji z wieżą kontroli itp. oraz infrastruktury lotniska (w tym przebiegu dróg kołowania i pasów startowych), możliwe jest osiągnięcie stanu hazardowego, w którym bezpieczeństwo uczestników ruchu zagrożone jest kolizją. Analizę bezpieczeństwa prowadzi się pod kątem potencjalnego wystąpienia danego hazardu dla wybranych scenariuszy zdarzeń, uwzględniając ich parametry czasowe.

Analiza Systemu Kontroli Lotniska (ang. *Airport Control System Analysis*, ACSA) jest projektem modelowania i weryfikacji bezpieczeństwa zachowania aktorów lotniska, czyli pojazdów, wieży kontroli itp. Obecnie możliwe jest modelowanie pewnych aspektów zachowania poruszającego się samolotu, w tym procedury wstrzymania ruchu lub odlotu samolotu [1]. Zachowanie aktorów modelowane jest, a jego własności, w tym osiągalność hazardu, weryfikowane są w symbolicznym weryfikatorze modelowym NuSMV [2][3], gdzie scenariusz zdarzeń opisywany jest w języku skryptowym NuSMV jako maszyna skończenie stanowa, a badane własności, w tym osiągalność hazardu kolizji, w języku logiki temporalnej CTL [4] i TCTL [5].

W tym rozdziale przedstawiono sposób modelowania scenariusza zdarzeń jako czasowe automaty skończenie stanowe, w szczególności modelowanie zmiany czasu oraz losowego wyboru momentu zajścia zdarzenia. Ze względu na duży rozmiar przestrzeni stanów tworzonego modelu przedstawiono metody optymalizacji skryptu NuSMV, w szczególności przez niedeterministyczne losowanie czasu zajścia zdarzeń już w stanie początkowym, co nazwano *predestynacją*. Skraca to czas kompilacji skryptu i weryfikacji zawartych w nim formuł nawet o 90%. Przedstawiono też sposób jakościowej i ilościowej analizy osiągalności hazardu przez weryfikację określonych wyrażeń i formuł CTL.

Układ kolejnych podrozdziałów: 2 – modelowa weryfikacja w NuSMV; 3 – studium przypadku wstrzymania kołowania samolotu; 4 – wybrane wzorce modelowania; 5 – metody i efekt optymalizacji skryptu; 6 – metoda weryfikacji modelu w analizie hazardu; 7 – podsumowanie.

2. Modelowa weryfikacja w NuSMV

Zgodność działania systemu z jego oczekiwanymi własnościami bada się w modelowej weryfikacji systemu [6][7], sprawdzając możliwość utworzenia automatu produktowego, łączącego zachowanie automatu modelującego system, z zachowaniem automatu modelującego jego zanegowane własności, czyli nieprawidłowe zachowanie. Możliwość ta dowodzi niespełnienia własności. System modeluje się jako skończenie stanowy automat (lub automaty), a każdą własność definiuje się jako formułę wybranej logiki temporalnej, w tym przypadku CTL i TCTCL.

Istnieją narzędzia automatyzujące proces weryfikacji, w tym: Kronos [8], SMV [9], Spin [10], VerICS [11] i Verus [12]. Jednym z nich jest też, wykorzystywany w ACSA, symboliczny weryfikator modelowy (ang. *Symbolic Model Verifier*) NuSMV. Konstruowane w nim maszyny skończenie stanowe definiowane są w jego skryptowym języku przez zmienne określające stan systemu, w tym czas oraz pozycję i dynamikę pojazdu. Dostępne są też stałe i zmienne zamrożone, czyli jednokrotnego zapisu. Jest to budowa modularna i hierarchiczna – system modelowany jest jako moduł `Main`, gdzie definiowane są m.in. globalne zmienne. Zmienne definiujące powtarzalny element systemu, np. samolot, można zdefiniować jako osobny moduł, mający żądaną liczbę instancji w module `Main`. Zmienne definiujące niepowtarzany element można zdefiniować bezpośrednio w module `Main`.

Dla zmiennej definiuje się typ i zakres wartości oraz, opcjonalnie, wartość początkową i funkcję przejścia. Zdefiniowanie stanu automatu (np. początkowego, końcowego, hazardowego) polega na określeniu charakteryzujących go wartości zmiennych. Niezdefiniowanie wartości zmiennej w początkowym stanie lub wartości zmiennej w następnym stanie w funkcji przejścia, spowoduje jej losowy wybór z zakresu zadeklarowanego w definicji zmiennej. Podobnie, nieprecyzyjne zdefiniowanie wartości zmiennej w początkowym stanie lub wartości zmiennej w następnym stanie w funkcji przejścia, spowoduje jej losowy wybór z tak zdefiniowanego zakresu. W ten sposób możliwe jest modelowanie niedeterminizmu zachowania systemu. Ograniczeniem jest dyskretyzacja parametrów liczbowych, gdyż mają typ całkowitoliczbowy (Integer).

Wartościowanie zmiennych modelowane jest wyrażeniami: `INIT` (definicja stanów początkowych przez podanie początkowej wartości zmiennych), `INVAR` (definicja stanów niezmiennych przez podanie niezmiennie zachowywanych wartości lub relacji zmiennych) i `TRANS` (funkcja przejścia – definicja następnego stanu przez podanie jego wartości zmiennych). Wartościowanie może uwzględniać bieżący i następny stan. W ten sposób unika się ręcznego definiowania stanów automatu, uwzględniającego wszystkie możliwe osiągalne kombinacje wartości zmiennych, jednakże czas kompilacji skryptu i analizy modelu pozostaje dość długi (zob. podrozdział 5). Wyrażenia te powinny być prawdziwe, gdyż fałszywe wyrażenie INIT skutkuje brakiem stanów początkowych, fałszywe wyrażenie INVAR skutkuje brakiem stanów osiągalnych, a fałszywe wyrażenie TRANS skutkuje brakiem następnego stanu.

NuSMV podczas kompilacji skryptu do modelu systemu, przekształca go do postaci binarnego drzewa decyzyjnego bez udziału analityka, który może tylko generować przykładowe ścieżki stanów i weryfikować własności systemu przy pomocy wybranego algorytmu heurystycznego.

Własności modelu definiuje się jako formuły logiki temporalnej (m.in. CTL i TCTL; w [1] zamiast niej podano RTCTL), a następnie automatycznie weryfikuje ich spełnienie przez model. Można utworzyć dowolną formułę, poprawnie zdefiniowaną przez składnię wybranego języka logiki temporalnej.

Ogólnie, formuła dotyczy stanu lub stanów systemu określonych przez wartościowanie jego zmiennych i może być zaopatrzona w pary operatorów: ścieżkowy – określający możliwość (E) lub konieczność (A) jej prawdziwości i punktowy – określający jej prawdziwość dla pewnego (F) lub każdego (G) stanu w przyszłości (inne operatory punktowe tu pominięto – nie zostaną wykorzystane); np. `AG(f)` oznacza, że `f` jest prawdziwe w każdym stanie na każdej ścieżce stanów, czyli (nieformalnie) na pewno zawsze, a `EF(f)` oznacza, że `f` jest prawdziwe w pewnym stanie na pewnej ścieżce stanów, czyli (nieformalnie) być może kiedyś.

Formalnie, formuła interpretowana jest dla struktury Kripkego `M=(S,R,L)`, gdzie: `S` to zbiór stanów, w których wyrażenia mogą przyjmować wartość logiczną (prawda/fałsz); `R` to relacja takiego następstwa stanów, że każdy ma przynajmniej jeden następny i dokładnie jeden poprzedni stan; a `L` to funkcja logicznie wartościująca wyrażenia w stanach. Tak więc stany uporządkowane są w drzewo, którego korzeniem jest stan, w którym podana formuła jest prawdziwa.

3. Studium przypadku

Przykładowe wzorce projektowe skryptu NuSMV oraz formuły służące analizie osiągalności hazardu, zostaną przedstawione dla przypadku scenariusza opisującego wstrzymanie kołowania samolotu na pas startowy. Analiza podobnego przypadku – wstrzymania procedury odlotu – przedstawiona została w [1]. W nawiasach podano nazwy zmiennych użytych w skrypcie NuSMV i ewentualnie ich wartościowanie.

Modelowany jest ruch samolotu na prostej drodze kołowania od miejsca postoju do wjazdu na pas startowy. Na tej drodze jest linia zatrzymania, przed którą samolot powinien się zatrzymać, jeśli odpowiednio wcześnie otrzyma takie polecenie podczas kołowania; w przeciwnym wypadku możliwy jest hazard nieuprawnionego wjechania na pas startowy, co być może spowoduje tam kolizję z innym uczestnikiem ruchu. Aktorami tego scenariusza są kołujący samolot i wieża kontroli, którzy komunikują się ze sobą drogą radiową. Każdy komunikat ma czas wysyłania i czas odbiorcy na reakcję nań (określone przedziałowo). Miejsce zatrzymania samolotu zależy od jego początkowej prędkości hamowania i ówczesnego położenia, a pośrednio od momentu wysłania komunikatu z poleceniem wstrzymania kołowania. Celem analizy bezpieczeństwa jest m.in. sprawdzenie osiągalności stanu hazardu wjechania samolotu na pas startowy (`HazardOfEnteredRunway`), czyli odpowiedzenie na pytanie, czy i kiedy ten hazard jest osiągalny.

Scenariusz zdarzeń jest (w skrócie) następujący: 1) Samolot wysyła do wieży komunikat gotowości do kołowania. To stan początkowy, kiedy rozpoczyna się odliczanie czasu przez zegar (`Time`). 2) Wieża otrzymuje ten komunikat i 3) po pewnym czasie reakcji wysyła do samolotu komunikat zgody na kołowanie wyznaczoną trasą. 4) Samolot otrzymuje ten komunikat i 5) po pewnym czasie reakcji wysyła do samolotu komunikat potwierdzenia. 6) Wieża otrzymuje ten komunikat. 7) Samolot po

dodatkowym czasie reakcji rozpoczyna kołowanie w kierunku pasa startowego (`Motion` przyjmuje wartość `Acceleration`), przyspieszając do danej maksymalnej prędkości (`VelocityMax`). 8) Wieża wysyła do samolotu komunikat wstrzymania kołowania przed linią zatrzymania. 9) Samolot otrzymuje ten komunikat i 10) po pewnym czasie reakcji rozpoczyna hamowanie (`Motion` przyjmuje wartość `Deceleration`). 11) Samolot zatrzymuje się (`Motion` przyjmuje wartość `None`).

4. Wzorce modelowania

Teraz przedstawione zostaną wybrane wzorce modelowania, gdzie w podanych fragmentach skryptu NuSMV znaki `--` rozpoczynają komentarz.

4.1. Statyczna struktura modelu

W analizowanym przypadku każdy aktor ma tylko jedną instancję, więc cały system modelowany jest jako moduł `Main`. Samolot i jego komunikacja z wieżą modelowane są jako zbiór zmiennych, zdefiniowanych pośrednio przez stałe i zmienne zamrożone.

Samolot modelowany jest przez jego: położenie (zmienna `Position` o zakresie od `0` do `RunwayThreshold` – położenie progu pasa startowego; linie 1–2), rodzaj ruchu (zmienna `Motion` zdefiniowana jako enumeracja: `None` – brak ruchu, `Acceleration` – przyspieszanie, `Constant` – stała prędkość i `Deceleration` – hamowanie; linia 3) i prędkość (zmienna `Velocity` o zakresie od `0` do `VelocityMax` – maksymalna prędkość kołowania; linie 4–5). Zmienne te są od siebie uzależnione, zgodnie z prawami fizyki.

```
1 FROZENVAR RunwayThreshold : {602};
2 VAR Position : 0..RunwayThreshold;
3 VAR Motion : {None, Acceleration, Constant, Deceleration};
4 FROZENVAR VelocityMax : {10};
5 VAR Velocity : 0..VelocityMax;
```

Komunikacja między aktorami modelowana jest przez: flagę `LastMessage` (numer ostatnio wysłanej wiadomości 0–3; linia 1) i czas wysłania ostatniej wiadomości `LastMessageTime` (zdefiniowany jako przedział wartości <min;max>, wyznaczonych analitycznie dla skrajnych realizacji scenariusza jako <`MReadyMinReceptionTime`; `MReadyMaxReceptionTime`>; linia 2).

```
1 VAR LastMessage : 0..3;
2 VAR LastMessageTime : MReadyMinReceptionTime..MStopMaxReceptionTime;
```

4.2. Definicja stanu

Stan modelu zdefiniowany jest przez wartościowanie jego zmiennych, dane w postaci logicznego wyrażenie, gdzie nieokreślenie wartości danej zmiennej oznacza przyznanie jej dowolnej wartości, z jej zakresu. W projekcie ACSA wyróżnia się m.in. stany: początkowe, końcowe i hazardowe.

Stan początkowy definiuje się wyrażeniem `INIT` z początkowymi wartościami wybranych zmiennych (linie 1–4). Analogicznie definiuje się inny stan (w tym końcowy i hazardowy) wyrażeniem `DEFINE` (linie 5–7).

```
1 INIT Motion = None & -- brak ruchu samolotu
2 Position = 0 & -- i brak przebytej drogi samolotu
3 Time = 0 & -- i brak upływu czasu
4 Velocity = 0; -- i brak prędkości samolotu

5 DEFINE Final := -- stan końcowy Final to:
6 (Position = RunwayThreshold) | (Position > 0 & Motion = None);
7  -- samolot osiągnął próg pasa startowego lub przestał się poruszać
```

4.3. Dynamiczna struktura modelu – funkcje przejścia

Funkcję przejścia definiuje się zwykle wyrażeniem `TRANS` ustalającym wartość zmiennej `z` w następnym stanie `next(z)`; zwykle stosuje się tu wyrażenie `case` o następującej postaci (tu dla dwóch warunków, linie 1–4):

```
1 TRANS next(z) in case
2 warunek : rezultat;
3 warunek : rezultat;
5 esac;
```

gdzie warunki sprawdzane są w podanej kolejności aż do napotkania spełnionego, a `rezultat` podany przy nim określa wartość `next(z)`. Następne warunki nie są sprawdzane ani podane przy nich `rezultaty` wykonywane. Należy zdefiniować każdy możliwy warunek, ewentualnie jako zawsze prawdziwe wyrażenia `TRUE`.

Linie 1–13 przedstawiają funkcję przejścia (wyrażenie `TRANS`) zmiennej `Position` z wybranymi przypadkami.

```
 1 TRANS next(Position) in case
 2 -- Przypadek 1:
 3 Final : Position;
 4  -- Jeśli osiągnięto stan końcowy, zdefiniowany jako Final,
 5  -- położenie nie zmieni się (zob. podrozdział 5.3).
 6 -- Przypadek 2:
 7 Motion = Acceleration : Position + Velocity * 2 + AccelerationRate;
 8  -- Jeśli samolot przyspiesza, to jego położenie w następnym stanie
 9  -- zwiększy się o bieżące_położenie+bieżącą_prędkość*2+przyspieszenie
10 -- Przypadek 7 (ostatni):
11 TRUE : Position;
12  -- W innym przypadku położenie w następnym stanie się nie zmieni.
13 esac;
```

Można zapewnić zmianę wartości zmiennej w danym przedziale czasu. Linie 1–24 zawierają wybrane przypadki z funkcji przejścia dla zmiennej `Motion`: pierwszy przypadek (linie 1–14) pozwala na zmianę `Motion` z `None` na `Acceleration` (linia 12), jeśli w następnym stanie czas realizacji scenariusza (`Time+1`) przekroczy dolną granicę przedziału czasu tej zmiany (linie 6–7), ale nie osiągnie górnej (linie 9–10); drugi przypadek (linie 15–24), który rozpatrywany jest, jeśli pierwszy nie zaszedł, wymusza tę

zmianę (linia 23), gdy w następnym stanie czas (`Time+1`) osiągnie górną granicę przedziału czasu tej zmiany (linie 20–21). W ten sposób kołowanie samolotu rozpocznie się w niedeterministycznie wybranym momencie z tego przedziału. Granice przedziału zdefiniowane są przez czas wysłania ostatniej wiadomości – zgody na kołowanie (`LastMessageTime`), minimalny i maksymalny czas reakcji na tę wiadomość (`MTaxiMinReactionTime` i `MTaxiMaxReactionTime`, odpowiednio) i dodatkowy minimalny i maksymalny czas reakcji (`MTaxiMinAddReactionTime` i `MTaxiMaxAddReactionTime`, odpowiednio); wszystkie parametry czasowe określone są w modelowanym i analizowanym scenariuszu zdarzeń.

```
 1 Motion = None
 2  -- Jeśli samolot się nie rusza,
 3 & next(LastMessage) = 2
 4  -- i w następnym stanie ostatnio otrzymaną wiadomością będzie
 5  -- zgoda na kołowanie,
 6 & Time + 1 >= next(LastMessageTime) + MTaxiMinReactionTime +
 7  MTaxiMinAddReactionTime
 8  -- i od tego czasu minie dolna granica danego czasu reakcji,
 9 & Time + 1 < next(LastMessageTime) + MTaxiMaxReactionTime +
10  MTaxiMaxAddReactionTime
11  -- i od tego czasu nie minie górna granica danego czasu reakcji,
12 : {Acceleration, None};
13  -- to w następnym stanie samolot może rozpocząć przyspieszanie
14  -- lub pozostać bez ruchu.

15 Motion = None
16  -- Jeśli samolot się nie rusza,
17 & next(LastMessage) = 2
18  -- i w następnym stanie ostatnio otrzymaną wiadomością będzie
19  -- zgoda na kołowanie,
20 & Time + 1 = next(LastMessageTime) + MTaxiMaxReactionTime +
21  MTaxiMaxAddReactionTime
22  -- i od tego czasu minie dokładnie górna granica danego czasu reakcji
23 : Acceleration;
24  -- to w następnym stanie rozpocznie się przyspieszanie.
```

4.4. Zegar systemu – czas realizacji scenariusza

Aby automat NuSMV był automatem czasowym, definiuje się zmienną `Time` (linia 2) o przedziale od `0` do `TimeMax` (analitycznie wyznaczony czas najdłuższej możliwej realizacji scenariusza zdarzeń; linia 1), której funkcja przejścia (linie 3–7) inkrementuje ją o 1 z każdą zmianą stanu. Przyjmuje się, że każda inkrementacja trwa 1 jednostkę czasu (tu: sekundę). Dzięki temu długość ścieżki stanów to też czas jej realizacji.

```
1 FROZENVAR TimeMAX : {88};
2 VAR Time : 0..TimeMax;

3 TRANS next(Time) in case
4 Time = TimeMax : Time
5  -- Jeśli osiągnięto maksymalną wartość TimeMax, to brak inkrementacji.
6 TRUE : Time + 1; -- W innym przypadku inkrementacja o 1 sekundę.
7 esac;
```

5. Optymalizacja modelu

Czas kompilacji skryptu NuSMV do binarnego drzewa decyzyjnego i czas weryfikacji każdej własności systemu zauważalnie zależy od: rozmiaru przestrzeni stanów, maksymalnej długości ścieżki stanów, rzędu wielkości liczb użytych w funkcji przejścia i wykonywania na nich obliczeń; jeśli jest nieakceptowalnie długi, zaleca się jego skrócenie przez optymalizację skryptu.

5.1. Metody optymalizacji

Na podstawie przeprowadzonych eksperymentów, zaleca się następujące metody optymalizacji skryptu, niezależne nawzajem od siebie, więc kolejność ich zastosowania nie ma znaczenia:

1. Zawężenie zakresu wartości zmiennych całkowitoliczbowych (Integer).

Uwzględniając skrajne realizacje scenariusza zdarzeń, analitycznie oblicza się minimalną i maksymalną wartość danej zmiennej, np. prędkości samolotu lub czasu rozpoczęcia przez niego kołowania i tak się ją definiuje przy pomocy wyrażenia `VAR`.

2. Zdefiniowanie stanów końcowych.

Stan końcowy to taki, którego osiągnięcie wyznacza koniec analizy scenariusza zdarzeń; np. zatrzymanie kołującego samolotu lub wystąpienie analizowanego hazardu. Funkcja przejścia każdej zmiennej powinna uniemożliwiać zmianę jej wartości, jeśli osiągnięto zdefiniowany stan końcowy.

3. Ograniczenie operacji arytmetycznych w funkcjach przejścia.

Operacja arytmetyczna to obliczenie wyrażenia arytmetycznego. Praktyka pokazuje, że szczególnie czasochłonne jest obliczanie wyrażenia zawierającego mnożenie i dzielenie. Wprowadza się więc zmienne przechowujące dane pośrednie, aby obliczenia arytmetyczne w funkcjach przejścia bazowały na nich; np. prędkość hamowania samolotu w następnym stanie zależy od momentu rozpoczęcia hamowania. Tę metodę należy stosować ostrożnie, gdyż niekiedy wydłuża czas weryfikacji z powodu zwiększenia przestrzeni stanów.

4. Zmniejszenie rzędu wartości zmiennych całkowitoliczbowych (Integer).

Dotyczy to zmiennych wykorzystywanych w obliczeniach arytmetycznych, zwłaszcza w mnożeniu i dzieleniu (zob. metodę 3). W tym celu zwiększa się poziom dyskretyzacji liczb, przez przeskalowanie wybranych jednostek fizycznych, jeśli nie doprowadzi to do istotnej utraty dokładności wyników analizy. Należy zauważyć, że zastosowanie tej metody, w przeciwieństwie do pozostałych, zmienia działanie modelu.

5. Losowanie wartości zmiennych w stanie początkowym (predestynacja).

Niedeterministyczne ustalenie wartości zmiennej, modelującej moment zajścia danego zdarzenia (w ramach jej zdefiniowanego przedziału wartości) już w stanie początkowym, a nie w stanie odpowiadającym rzeczywistemu ustaleniu tej wartości w modelowanym systemie, daje najlepsze rezultaty ze wszystkich przedstawionych metod, choć już metody 1–4 mogą zauważalnie skrócić czas kompilacji skryptu i weryfikacji formuł z własnościami systemu. Takie losowanie wartości zmiennych nazywamy tu *predestynacją*, przez oczywistą analogię do teologicznego znaczenia tego terminu.

5.2. Zastosowanie metod optymalizacji w badanym przypadku

W badanym przypadku kolejno i przyrostowo (tzn. z zachowaniem wcześniejszych) zastosowano metody optymalizacji 1–3 i 5, a następnie porównano rozmiar przestrzeni stanów, czas kompilacji skryptu i czas kompilacji skryptu wraz z weryfikacją jednej prawdziwej formuły (badano 4 formuły – podstawowe typy własności systemu). Metody 4 nie zastosowano z powodu nieakceptowalnego obniżenia poziomu dyskretyzacji parametrów liczbowych. Eksperyment wykonano na komputerze z dwurdzeniowym, 64- itowym procesorem AMD Athlon II X2 270 o częstotliwości 3,4G Hz, pamięcią RAM 7,8 GB i systemem operacyjnym Ubuntu 16.04 LTS, bez obciążenia wykonywaniem programów zbędnych do działania komputera.

Tabele 1 i 2 przedstawiają uśrednione wyniki badania, gdzie: kolumna A) to zastosowane metody optymalizacji, B) to liczba wszystkich stanów, C) to liczba osiągalnych stanów, D) to czas kompilacji skryptu, E) to D + czas weryfikacji formuły `AG(Velocity>=0)`, F) to D + czas weryfikacji formuły `EG(Velocity>=0)`, G) to D + czas weryfikacji formuły `AF(Velocity>=0)` i H) to D + czas weryfikacji formuły `EF(Velocity>=0)`.

Tabela 1. Rozmiar przestrzeni stanów oraz czas kompilacji skryptu.

A	B	C	D
–	2^42.5984	2^18.1258	0'9,681"
1	2^40.5859	2^18.1258	0'2,925"
1–2	2^40.5859	2^15.808	0'4,297"
1–3	2^44.0453	2^15.808	0'9,741"
1–3, 5	2^35.653	2^16.6978	0'1,556"

Tabela 2. Czas kompilacji skryptu i weryfikacji formuły.

A	E	F	G	H
–	4'35,075"	4'38,000"	4'37,179"	8'42,022"
1	0'38,136"	0'38,203"	0'38,988"	1'10,131"
1–2	0'13,097"	0'13,104"	0'13,472"	0'21,159"
1–3	9'12,780"	9'11,512"	9'32,983"	16'26,541"
1–3, 5	0'25,455"	0'25,412"	0'25,751"	0'48,311"

Zastosowana optymalizacja (metody 1–3 i 5) zmniejszyła przestrzeń wszystkich stanów o 99,2%, w tym stanów osiągalnych o 62,8%; czas kompilacji skryptu skrócił się o 83,9%; czas kompilacji z weryfikacją przykładowej formuły skrócił się średnio o 90,9% dla badanych formuł. Należy zauważyć, że czas weryfikacji formuły byłby dłuższy, gdyby była fałszywa, o czas znalezienia przykładowej ścieżki stanów – dowodu fałszywości formuły.

Zastosowanie 5 metody (predestynacji), względem pozostałych, zmniejszyło przestrzeń wszystkich stanów o 99,7%, zwiększając (!) jednak przestrzeń stanów osiągalnych o 85,3%; czas samej kompilacji skrócił się o 84%, a czas kompilacji z weryfikacją formuły skrócił się średnio o 95,5%.

5.3. Przykłady zastosowania metod optymalizacji

Metoda 1. Analitycznie wyznaczono minimalny i maksymalny moment rozpoczęcia kołowania, czyli przyspieszania (`AccelerationInitTime`; linie 1–2), zamiast definiowania go jak zegara systemu, czyli od `0` do maksymalnego czasu realizacji scenariusza zdarzeń (`TimeMax`; linia 3).

```
1 VAR AccelerationInitTime :
2  AccelerationMinInitTime..AccelerationMaxInitTime;

3 VAR AccelerationInitTime : 0..TimeMax;
```

Metoda 2. Zdefiniowano stan końcowy `Final` wyrażeniem `DEFINE` (linie 1–3) i uniemożliwiono zmianę wartości zmiennych w ich funkcjach przejścia w wyrażeniach `TRANS` w przypadku jego osiągnięcia (przykład: linie 4–8). W funkcji przejścia w wyrażeniu `TRANS` zawierającym wyrażenie `case` ta blokada zmiany stanu jest pierwszym warunkiem.

```
1 DEFINE Final :=
2 (Position = RunwayThreshold) | (Position > 0 & Motion = None);
3  -- Samolot osiągnął próg pasa startowego lub przestał się poruszać.

4 TRANS next(Motion) in case
5 Final : Motion;
6  -- Jeśli osiągnięto stan Final, to ruch samolotu się nie zmieni.
7 -- Tu pozostałe przypadki.
8 esac;
```

Metoda 3. Wprowadzono zmienną `DecelerationInitVelocity` (linia 1) do zapamiętania momentu rozpoczęcia hamowania, ograniczoną przez maksymalną prędkość kołowania (`VelocityMax`). Wykorzystano ją w funkcji przejścia zmiennej `Motion` (rodzaj ruchu), wyznaczając moment zakończenia hamowania (linie 2–15).

```
 1 VAR DecelerationInitVelocity : 0..VelocityMax;

 2 TRANS next(Motion) in case
 3 -- Tu wcześniejsze przypadki.
 4 Motion = Deceleration
 5  -- Jeśli samolot hamuje,
 6  & DecelerationInitTime < AccelerationInitTime + Calculation1
 7  -- i hamowanie rozpoczęło się, zanim osiągnięto maksymalną prędkość
 8  -- kołowania VelocityMax,
 9 & Time * 10 + 10 >= DecelerationInitTime * 10 -
10 (DecelerationInitVelocity / DecelerationRate) * 10
11  -- i czas hamowania zakończy się w następnym stanie,
12 : None;
13  -- to samolot się zatrzyma.
14 -- Tu pozostałe przypadki.
15 esac;
```

Metoda 4. Nie zastosowano.

Metoda 5. W modelu bez predestynacji wartość `AccelerationInitTime` ustalana jest przy zmianie stanu z `Motion=None`, na `Motion=Acceleration` (linie 1–6), dokonywanej niedeterministycznie na podstawie innych parametrów czasowych w innej funkcji przejścia. W modelu z predestynacją wartość zmiennej `AccelerationInitTime` ustalana jest niedeterministycznie w stanie początkowym z zakresu Min–Max (linie 7–8), a od niej deterministycznie zależy zmiana wartości `Motion` (linie 9–14).

```
 1 -- Przypadek z wyrażenia TRANS next(AccelerationInitTime) in case:
 2 Motion != Acceleration & next(Motion) = Acceleration
 3  -- Jeśli samolot teraz nie przyspiesza, a w następnym stanie będzie,
 4 : Time + 1;
 5  -- to czas następnego stanu zapisany zostanie jako czas rozpoczęcia
 6  -- przyspieszania.

 7 VAR AccelerationInitTime:
 8  AccelerationMinInitTime..AccelerationMaxInitTime;

 9 -- Przypadek z wyrażenia TRANS next(Motion) in case:
10 Motion = None & Time + 1 = AccelerationInitTime
11  -- Jeśli samolot nie porusza się, a w następnym stanie nastąpi moment
12  -- rozpoczęcia kołowania,
13 : Acceleration;
14  -- to samolot w następnym stanie rozpocznie kołowanie.
```

6. Weryfikacja modelu w analizie hazardu

Weryfikowane własności systemu definiowane są jako: wyrażenia `COMPUTE`, obliczające minimalny i maksymalny rozmiar ścieżki między dwoma danymi stanami, i wyrażanie `CTLSPEC`, sprawdzające prawdziwość danych formuł CTL i TCTL. Wyrażenie zweryfikowane negatywnie potwierdzane jest przykładową (zwykle o minimalnej długości) ścieżką prowadzącą do stanu z nią sprzecznego, co umożliwia poszukiwanie ścieżek prowadzących do danego stanu.

6.1. Analiza osiągalności hazardu w modelu z predestynacją

Jednym z głównych celów projektu ACSA jest analiza hazardu, gdzie bada się m.in. osiągalność stanu hazardowego, czyli czy i kiedy ten stan może zostać osiągnięty.

Osiągalność stanu `s` opisuje formuła `EF(s)`, która jest prawdziwa, jeśli `s` jest osiągalny. W przypadku modelu z predestynacją, gdzie jest więcej niż jeden stan początkowy (ich liczba to liczba kombinacji wszystkich początkowych wartościowań wszystkich zmiennych), jej weryfikacja zwróci wynik negatywny nie tylko przy nieosiągalności `s`, ale też, jeśli `s` jest osiągalny nie dla każdego stanu początkowego; taki wynik jest nieprzydatny, więc zamiast osiągalności `s` bada się jego nieosiągalność.

Nieosiągalność stanu `s` opisuje formuła `!EF(s)`, która jest fałszywa, jeśli `s` jest osiągalny. Negatywny wynik jej weryfikacji nie tylko potwierdza osiągalność `s` dla przynajmniej jednego stanu początkowego, ale też opatrzony jest przykładową ścieżką

stanów od pewnego początkowego do `s`. Długość tej ścieżki to czas osiągnięcia `s` – dzięki zmiennej zegarowej `Time` z każdą zmianą stanu mija jedna jednostka czasu.

Najwcześniejszy moment osiągnięcia stanu `s` znajduje wyrażenie `COMPUTE MIN[p,s]`, gdzie `p` jest danym stanem początkowym; analogicznie najpóźniejszy moment znajduje wyrażenie `COMPUTE MAX[p,s]`.

Analizę możliwych ścieżek stanów można też prowadzić w trybie interaktywnym, krok po kroku, losowo lub z wyborem generując żądaną liczbę następnych stanów.

6.2. Przykład analizy osiągalności hazardu w modelu z predestynacją

Przykład definicji hazardu: wyrażenie `DEFINE HazardOfEnteredRunway` definiuje stan hazardu wkroczenia samolotu na próg pasa startowego (linie 1–4), czyli takiego, gdzie samolot znajduje się na progu pasa startowego (`Position=RunwayThreshold`) i nadal się porusza (`Motion!=None`). Osiągalność tego hazardu bada się wyrażeniem `CTLSPEC` (linia 5).

```
1 DEFINE HazardOfEnteredRunway :=
2  Position = RunwayThreshold & Motion != None;
3  -- Definicja hazardu:
4  -- gdy samolot znajduje się na progu pasa startowego i się porusza.

5 CTLSPEC !EF(HazardOfEnteredRunway)
```

To wyrażenie zostało zweryfikowane negatywnie, co oznacza, że hazard jest osiągalny dla pewnego stanu początkowego (dosłownie: nie jest nieosiągalny). Dodatkowo wygenerowana została ścieżka prowadząca do stanu hazardu, który może zostać osiągnięty w czasie 59 sekund.

Wykonanie wyrażenia `COMPUTE` (linia 1) wykazało, że najwcześniejszy czas osiągnięcia stanu hazardu od stanu początkowego (kiedy `Time=0`) to 55 sekund.

```
1 COMPUTE MIN[Time=0, HazardOfEnteredRunway]
```

6.3. Czas analizy osiągalności hazardu w różnych modelach

Dla obu powyższych wyrażeń (`CTLSPEC` i `COMPUTE`) zmierzono czas kompilacji skryptu i ich weryfikacji, kolejno i przyrostowo stosując metody optymalizacji 1–3 i 5. Wyniki pomiarów przedstawia Tabela 3.

Tabela 3. Czas kompilacji skryptu i weryfikacji wyrażeń CTLSPEC i COMPUTE.

A	CTLSPEC	COMPUTE
–	5'15,052"	4'38,924"
1	0'48,344"	0'38,556"
1–2	0'19,476"	0'13,696"
1–3	12'8,532"	8'47,752"
1–3, 5	0'29,216"	0'25,840"

Zastosowana optymalizacja (metody 1–3 i 5) zmniejszyła czas kompilacji skryptu z weryfikacją wyrażenia `CTLSPEC`, jak i z weryfikacją wyrażenia `COMPUTE`, o 90,7%.

Zastosowanie 5 metody (predestynacji), względem pozostałych, zmniejszyło ten czas odpowiednio o 96% i 95,1%. Wynika stąd, że optymalizacja w podobny sposób wpływa też na czas generacji ścieżki stanów.

7. Podsumowanie

Weryfikator NuSMV pozwala modelować i weryfikować scenariusze zdarzeń jako maszyny skończenie stanowe, z implementacją czasowych parametrów zdarzeń, czasu zmiany stanu i ich niedeterministycznego wyboru; dzięki czemu można odzwierciedlić rzeczywiste zachowanie badanego systemu. Poprawność działania NuSMV jest potwierdzone formalnie; wszystko zależy od poprawności modelowania systemu w pożądanym zakresie i z akceptowalną dokładnością. Natomiast weryfikacja zachowania systemu wykorzystuje formuły logiki temporalnej, czyli formalny język definicji czasowych relacji między zdarzeniami wyrażonymi jakościowo lub ilościowo; jest on niezależny od języka definicji maszyny stanowej.

Na podstawie kolejnych modelowanych scenariuszy zdarzeń tworzone są wzorce projektowe dla różnych aspektów zachowania aktorów lotniska (samolotu, wieży kontroli itp.), jak: radiowa komunikacja typu handshake, czy wstrzymanie kołowania lub odlotu samolotu. Powstają też wzorce formuł weryfikacyjnych do analizy hazardu, w tym do określenia możliwości i czasu rozpoczęcia hazardu i znalezienia zależności przyczynowo-skutkowych między zdarzeniami prowadzącymi do niego.

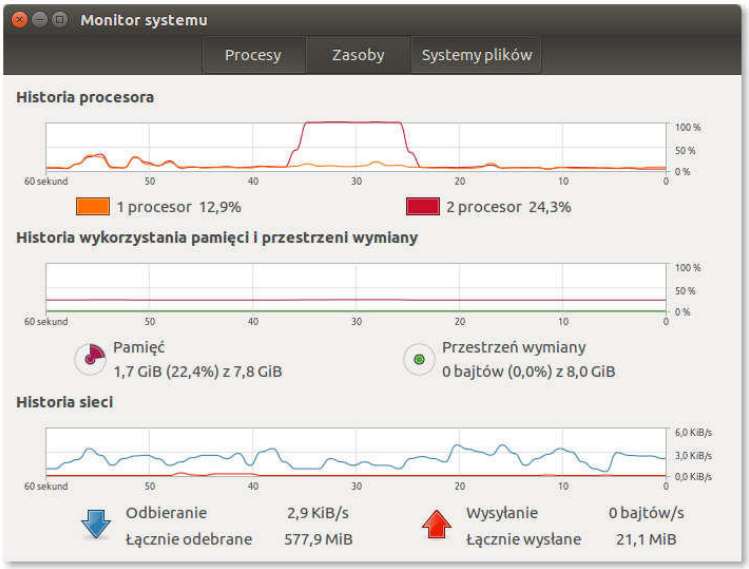
W celu skrócenia czasu kompilacji skryptu i weryfikacji własności modelu (w tym analizy hazardu), zwłaszcza w przypadku znajdywania przykładowej ścieżki stanów lub jej długości, zaproponowano metody optymalizacji skryptu NuSMV. Znacznie zmniejszają one przestrzeń stanów i, pośrednio, zauważalnie skracają ten czas (dla analizowanego przypadku średnio o 90,9%). Zauważyć można, że ograniczenie zakresu zmiennych i wprowadzenie stanów końcowych nie daje wystarczająco dobrego wyniku; osiągnięto go dzięki dodatkowemu losowaniu czasu zajścia zdarzeń już w stanie początkowym, co nazwano predestynacją.

Spośród zaprezentowanych metod optymalizacji jedynie metoda 4 może pogorszyć zgodność modelu NuSMV z modelowanym systemem, co potwierdziła weryfikacja odpowiednich formuł logiki temporalnej. Wynika to ze zmniejszenia dokładności pomiaru wielkości fizycznych (czas, odległość itp.). Stąd w przedstawionych badaniach nie korzystano z niej, a ograniczono się tylko do jej omówienia.

Czasowa optymalizacja skryptu nie wpływa zauważalnie na zajętość pamięci operacyjnej przez NuSMV, która i bez niej jest niska (1,7–3,7 GB w badanym przypadku). Stąd wniosek, że problemem jest złożoność czasowa algorytmu zaimplementowanego w NuSMV, który wykorzystuje tylko jeden rdzeń procesora i to w 100% (patrz trapezoidalny fragment wykresu na Rys. 1), więc zaleca się dodatkową sprzętową optymalizację przez zastosowanie szybciej taktowanego procesora.

Na koniec dodajmy, że przedstawione metody optymalizacji skryptu NuSMV mogą być łatwiej wykorzystywane przy modelowaniu przyszłych scenariuszy zdarzeń, jeśli

w procesie modelowania wezmą bezpośredni udział eksperci od analizowanego bezpieczeństwa. W tym celu opracowywany jest graficzny język modelowania, bazujący na UML, aby automatycznie tworzyć skrypt na podstawie diagramów opisujących strukturę i dynamikę modelu. Ułatwi to prawidłowe stosowanie zoptymalizowanych wzorców projektowych przy tworzeniu kolejnych modeli.



Rys. 1. Monitor systemu podczas kompilacji skryptu (24–37 s)

Bibliografia

[1] P. Głuchowski, NuSMV Model Verification of an Airport Traffic Control System with Deontic Rules, Dependability Engineering and Complex Systems (2016), 195–206.
[2] R. Cavada, A. Cimatti, Ch.A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, A. Tchaltsev, NuSMV 2.6 User Manual, FBK-irst, 2016.
[3] R. Cavada, A. Cimatti, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, NuSMV 2.6 Tutorial, FBK-irst, 2016.
[4] E.A. Emerson, Temporal and modal logic, Handbook of Theoretical Computer Science Vol.B (1990).
[5] R. Alur, C. Courcoubetis, D. Dill, Model checking in dense real-time, Information and Computation 104(1), (1993).
[6] R. Alur, C. Courcoubetis, D. Dill, Model Checking for Real-Time Systems, IEEE Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press., 1990.
[7] P. Wolper, Construting Automata from Temporal Logic Formulas: A Tutorial, Lectures on Formal Methods and PerformanceAnalysis, Vol. 2090 of LNCS, Springer, 2001.
[8] S.Yovine, Kronos: A verification tool for real-time systems, Springer International Journal of Software Tools for Technology Transfer, Vol. 1, Springer, 1997.
[9] K.L. McMillan, The SMV System, Symbolic Model Checking, Springer US, 1993.
[10] G. Holzmann, The model checker SPIN, IEEE Transactions on Software Engineering, Vol. 23(5), 1997.
[11] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, A. Zbrzezny, VerICS: A tool for verifying timed automata and Estelle specifications, Proc. of the 9th Int. Conf. on Tools

and Algorithms for the Construction and Analysis of Systems (TACAS'03). Vol. 2619 of LNCS, Springer, 2003.

[12] S. Campos, E. Clarke, W. Marrero, M. Minea, Verus: A tool for quantitative analysis of finite-state real-time systems, School of Computer Science Carnegie Mellon University1996.

Rozdział 9

Wybrane problemy wytwarzania systemów czasu rzeczywistego dla bezpilotowych statków powietrznych

1. Wprowadzenie

Wytwarzanie bezpilotowych statków powietrznych (BSP) stało się już istotną dziedziną wiedzy [1, 2, 3] a także gałęzią przemysłu światowego [4, 5] i krajowego [6, 7]. System bezpilotowego statku powietrznego (SBSP) standardowo składa się z obiektu latającego oraz stacji naziemnej (por. rys. 1). Wykonywanie misji obiektu latającego polega na autonomicznym odtworzeniu zaprogramowanej trajektorii lotu. We wskazanych fazach misji ładunek przenoszony przez obiekt realizuje zaplanowane działania. Ze względów bezpieczeństwa system powinien zapewnić możliwość przejęcia BSP przez operatora. Stacja naziemna służy zarówno do planowania, nadzorowania, a także, jeśli zachodzi taka potrzeba, bezpośredniego wykonywania misji. Systematycznie zwiększana złożoność misji zlecanych bezpilotowym statkom powietrznym oraz badania nad wprowadzeniem ich lotów do kontrolowanej przestrzeni powietrznej [8] wymuszają intensywny rozwój oprogramowania i struktury komunikacyjnej SBSP.

W pracy pokazane zostały problemy wiązane z wytwarzaniem oprogramowania oraz struktury komunikacyjnej rozwiązywane podczas konstruowania autorskiego SBSP. Został on stworzony na potrzeby realizacji kilku projektów badawczych [8, 17], które dotyczyły rozwoju systemów sterowania statków bezzałogowych, w tym algorytmów automatycznego startu, lądowania i poruszania się po płytach lotnisk.

Wytwarzanie oprogramowania systemów bezpilotowych statków powietrznych oraz dobór i konfigurowanie sieci komputerowej integrującej moduły sprzętowe wydaje się powielaniem standardowych rozwiązań opracowanych w szeregu publikacji dotyczących projektowania i implementacji systemów czasu rzeczywistego [10, 11, 12, 13, 14, 15]. Doświadczenia autorów niniejszego opracowania wskazują jednak, że w projektowaniu realnych systemów można napotkać na problemy, których rozwiązania są w sposób niepełny lub wcale nieopublikowane. W kolejnych podrozdziałach pokazane zostaną wybrane problemy implementacyjne systemów czasu rzeczywistego, które rozwiązano w czasie tworzenia autorskiego oprogramowania i konfiguracji sieciowej eksperymentalnego SBSP. Rozwiązania problemów mogą z powodzeniem, jak się wydaje, zostać zaimplementowane w szeregu innych aplikacjach czasu rzeczywistego.

Rys. 1. Podstawowe składniki systemu bezpilotowego statku powietrznego

W podrozdziale drugim zostanie omówiony eksperymentalny SBSP rozwijany przy współudziale autorów. Przedstawiona architektura systemu wynika z aktualnie realizowanych celów badawczych – opracowywania systemu automatycznego startu i lądowania BSP na lotniskach dostosowanych do przyjmowania komercyjnych lotów pasażerskich i towarowych. W podrozdziale trzecim zostaną wskazane dwa problemy implementacyjne rozwiązane podczas tworzenia oprogramowania czasu rzeczywistego autopilota – głównego podsystemu utrzymującego statek powietrzny w locie. Rozważone zostaną konsekwencje wyboru pewnego stylu programowania zadań cyklicznych przy nastąpieniu nadpisania aktywacji zadań (ang. task overriding) oraz szczególne rozwiązanie problemu wzajemnego wykluczania. W rozdziale czwartym zostaną wskazane kolejne problemy rozwiązywane w wytwarzanym systemie wynikające z zastosowania lokalnej sieci komputerowej Ethernet/IP/UDP jako jednego z mediów integrujących system czasu rzeczywistego. Dyskusji zostaną poddane konsekwencje zastosowania takiej metody komunikacji oraz wybrane metody konfiguracji sieci, programowania sieciowego i nadrzędnego, które mają zapewnić zwiększoną przewidywalność systemu. Ostatni podrozdział będzie zawierał podsumowanie oraz perspektywy dalszych prac rozwojowych nad systemem.

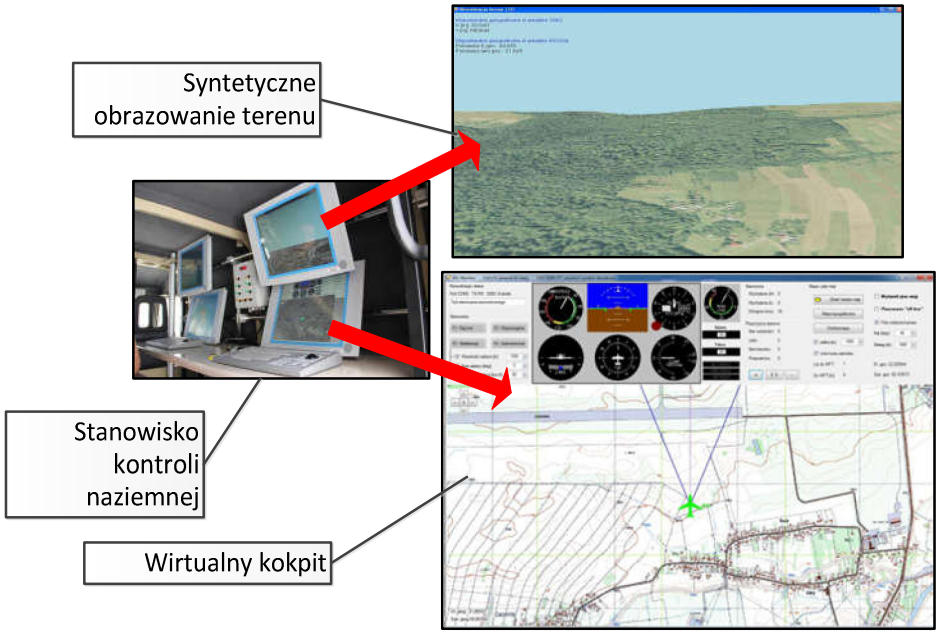
2. Eksperymentalny SBSP

Eksperymentalny system bezpilotowego statku powietrznego składa się z mobilnej stacji naziemnej zainstalowanej w samochodzie oraz zmodyfikowanego samolotu "MP-02 Czajka" [16, 17] (por. rys. 1). Stacja naziemna i samolot utrzymują cyfrowe połączenie radiowe.

Na pokładzie samolotu dokonano odpowiednich modyfikacji umożliwiających odbywanie lotów bezzałogowych. Wychylenie płaszczyzn sterowych oraz obroty silnika mogą być zmieniane za pomocą serwomechanizmów elektrycznych. Wartości te są zadawane przez komputer autopilota i obliczane na podstawie elektronicznego systemu odniesienia przestrzennego, pokładowych czujników oraz parametrów wynikających z planowanej trajektorii lotu. Autopilot utrzymuje również połączenie radiowe ze stacją naziemną. Osobne moduły mikroprocesorowe realizują algorytmy sterujące automatycznym startem i lądowaniem oraz autonomiczną nawigacją po płycie lotniska. Konfiguracja samolotu pozwala na odbywanie zarówno lotów załogowych jak i bezzałogowych. Częstą praktyką jest testowanie poszczególnych podsystemów autonomicznych w locie pod nadzorem pilota-oblatywacza.

Stację naziemną tworzy klaster komputerów klasy PC wspomagających planowanie, wykonywanie oraz rejestrację przebiegu misji (por. rys. 2). Planowanie misji odbywa się z zastosowaniem graficznego obrazowania terenu i polega na definicji punktów w przestrzeni oraz manewrów, które w danych punktach ma wykonać platforma latająca (np. dokonanie inspekcji pewnego obszaru w otoczeniu danego punktu). Oprogramowanie stacji naziemnej, oprócz podsystemu planowania misji, który nie ma charakteru czasu rzeczywistego, ma za zadanie w czasie rzeczywistym wizualizować bieżące parametry lotu (oferując wirtualny kokpit), obrazować położenie BSP na tle wirtualnej mapy oraz umożliwiać bieżące zadawanie parametrów lotu, jeśli operator przejął BSP do ręcznego sterowania. Operowanie BSP może być wspomagane z zastosowaniem syntetycznego obrazowania terenu otrzymywanego na drodze fuzji danych o jego orientacji przestrzennej, położeniu i wysokości z bazą danych trójwymiarowych map lotniczych. Osobny podsystem stacji naziemnej obsługuje ładunek BSP. Przykładowo, jeśli misja obejmuje inspekcję z zastosowaniem kamer, wydzielona konsola operatorska umożliwia wyświetlanie i archiwizację strumieni video.

Istotnym komponentem systemu jest cyfrowe łącze radiowe umożliwiające wymianę informacji pomiędzy stacją naziemną a platformą latającą. Komunikacja odbywa się z zastosowaniem radiomodemu o zasięgu do 80 km. Rozważania prowadzone w kolejnych podrozdziałach zakładają, że połączenie radiowe umożliwia wymianę komunikatów zgodnie z rodziną protokołów Ethernet/IP/UDP. W takim wypadku stacja naziemna oraz komponenty mikroprocesorowe będą połączone z zastosowaniem lokalnej sieci komputerowej.



Rys. 2. Interfejsy graficzne stacji naziemnej

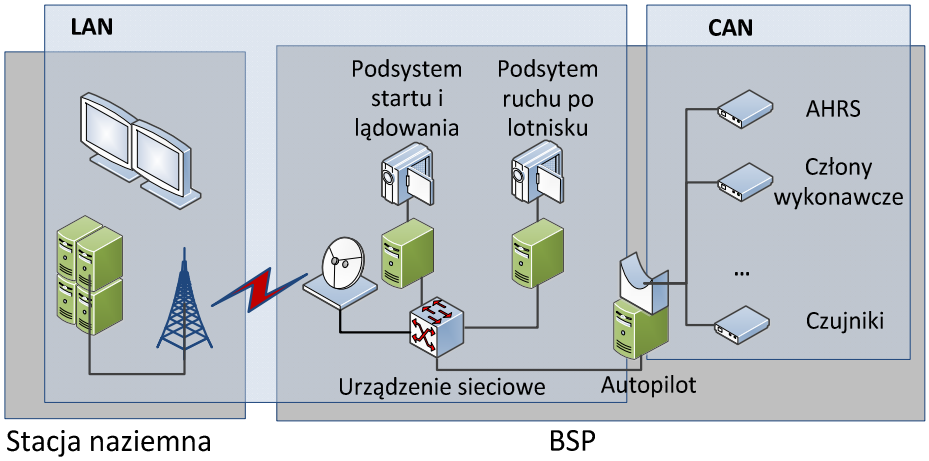
Na rys. 3 pokazano eksperymentalny SBSP z perspektywy komponentów sprzętowych połączonych w sieć komputerową. Komputer autopilota z jednej strony komunikuje się z układami pokładowymi samolotu poprzez magistralę CAN, z drugiej wymienia informacje z podsystemami automatycznego lądowania i startu, nawigacji lotniskowej i stacji naziemnej (za pomocą łącza radiowego) z zastosowaniem lokalnej sieci komputerowej (LAN) umożliwiającej wymianę danych zgodnie z rodziną protokołów Ethernet/IP/UDP.

Wydzielenie osobnych komputerów realizujących start i lądowanie, a także autonomiczne poruszanie się po lotnisku wynika z aktualnie realizowanego zadania badawczego. SBSP jest obecnie platformą testową dla wykonywania autonomicznych startów i lądowań oraz nawigacji po lotnisku. Poszczególne zadania projektowe są realizowane przez różne zespoły badawcze, które dostarczają wydzielone moduły sprzętowe sterujące wskazanymi fazami lotu. Integracja nowych modułów odbywa się z zastosowaniem sieci Ethernet/IP/UDP.

W konsekwencji, z punktu widzenia inżynierii oprogramowania system bezpilotowego statku powietrznego jest rozproszoną aplikacją czasu rzeczywistego. W lokalnych węzłach systemu wykonywane są zadania sterujące BSP na odpowiednich poziomach odbywania misji. Integrację głównych modułów sprzętowych można przeprowadzić z zastosowaniem technik programowania sieciowego.

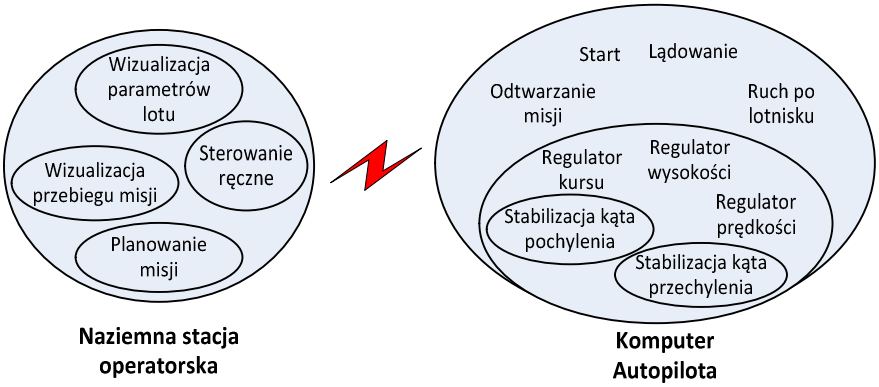
Na rys. 4 pokazano podstawowe moduły programowe umożliwiające skomponowanie SBSP. Głównym systemem czasu rzeczywistego BSP jest autopilot, który ma za zadanie utrzymywać orientację przestrzenną obiektu latającego. Jest to

oprogramowanie sterujące, które na podstawie sygnałów z modułu czujników orientacji przestrzennej stabilizuje kąty pochylenia i przechylenia obiektu latającego.



Rys. 3. Moduły sprzętowe i techniki komunikacji w eksperymentalnym SBSP

Podstawowe zadania stabilizujące służą jako podstawa do budowy kolejnych podsystemów sterujących umożliwiających lot w określonym kierunku (regulator kursu), na określonej wysokości (regulator wysokości) i z określoną prędkością (regulator prędkości) (por. [9]). Wymienione podsystemy uzupełnione kolejnym oprogramowaniem nadrzędnym umożliwiają z kolei odtwarzanie misji, czyli odbywanie lotu śledzącego zadaną trajektorię. Szczególnie wrażliwymi ze względu na bezpieczeństwo fazami lotu są start i lądowanie, a także (jeśli jest to wykonywane przez urządzenie autonomiczne) poruszanie się po lotnisku. Wykonywanie tych fragmentów misji powierza się z reguły kolejnym wydzielonym podsystemom czasu rzeczywistego, a w przypadku eksperymentalnego SBSP wykonywane są na osobnych modułach sprzętowych (por. rys. 3).

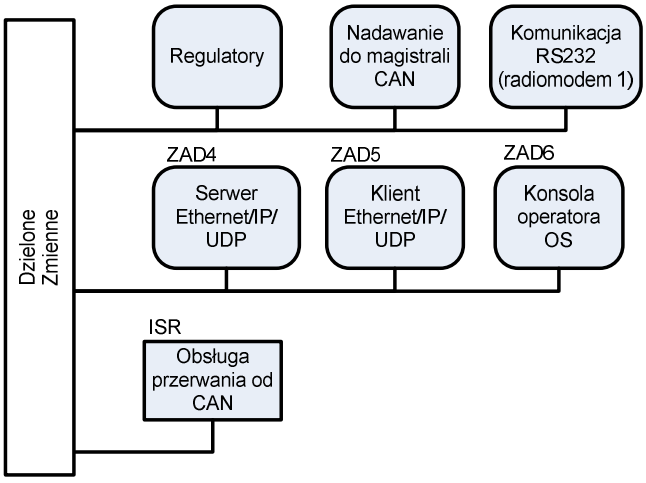


Rys. 4. Komponenty programowe systemu bezpilotowego statku powietrznego

Oprogramowanie BSP obejmuje również wymianę informacji pomiędzy platformą latającą a stacją naziemną. Kluczowymi danymi przesyłanymi do operatora są położenie, orientacja przestrzenna i stan obiektu latającego. Standardowy strumień danych wysyłany ze stacji operatorskiej zawiera z kolei komendy korygujące misję, chociaż oczekuje się również możliwości zdalnego sterowania obiektu latającego przez operatora. Oczekuje się również, że BSP będzie w stanie przesyłać strumienie danych z czujników i systemów wizyjnych zwykle służących do rejestracji danych i wspomagania nawigacji.

3. Wybrane problemy programowania zadań czasu rzeczywistego

Oprogramowanie autopilota – komputera pokładowego jest przygotowywane w języku C. Platformą sprzętową jest system uruchomieniowy ADS512101 z procesorem Freescale MPC5121e. System operacyjny komputera pokładowego to VxWorks 6.8 [18,19,20]. Zastosowanie systemu operacyjnego czasu rzeczywistego sprowadza projektowanie oprogramowania do dekompozycji modułów programowych na zbiór zadań (czasu rzeczywistego) pozostających ze sobą i otoczeniem w interakcji. Na rys. 5 pokazano zbiór zadań opracowywanego autopilota. Zadania te stanową pewnego rodzaju "ramę" dla implementowanych algorytmów sterowania czy protokołów komunikacyjnych. Przykładowo, kod w języku C realizujący algorytmy sterowania jest generowany automatycznie w środowisku Matlab-Simulink. Warstwa zadań pozwala na cykliczne wykonywanie algorytmów sterujących i efektywną wymianę informacji pomiędzy regulatorami a otoczeniem autopilota. Struktura zadań czasu rzeczywistego może być również wykorzystana do dalszej dekompozycji zadań sterujących (por. [9]).



Rys. 5. Zadania czasu rzeczywistego w oprogramowaniu autopilota

W wyidealizowanym modelu obliczeniowym systemów czasu rzeczywistego zbiór zadań jest szeregowany zgodnie z przyjętym algorytmem, zadania zawsze spełniają swoje ograniczenia czasowe i komunikują się z współdzielonymi zasobami z zasadą

wzajemnego wykluczania. Istnieją matematyczne techniki wykazywania szeregowalności zbioru zadań dla ustalonych architektur systemów mikroprocesorowych i mechanizmów wymiany informacji [10, 12, 13]. Wytwarzanie oprogramowania na ustaloną platformę sprzętową wymaga jednak czasami zmierzenia się z problemami rzadziej poruszanymi w publikacjach teoretycznych. W kolejnych podrozdziałach rozważone zostaną praktyczne problemy implementacyjne wynikłe z zastosowanej konkretnej architektury systemu mikroprocesorowego i systemu operacyjnego czasu rzeczywistego.

3.1. Problem poprawnego wznawiania obliczeń dla kolejnych instancji zadania

Oprogramowanie autopilota, to przede wszystkim realizacja algorytmów sterowania, czyli cyklicznie wykonywanych instancji zadań sterujących. Przy próbie klasyfikacji można takie oprogramowanie z powodzeniem zaliczyć do zbioru zadań sterujących wyzwalanych czasowo (por. [14]). W poprawnie zaprojektowanym systemie czasu rzeczywistego instancja każdego cyklicznego zadania kończy swoje obliczenia przed rozpoczęciem następnego. Schemat programowania tego typu zadań z zastosowaniem modułów czasowo-licznikowych watchdog pokazano na listingu 1.

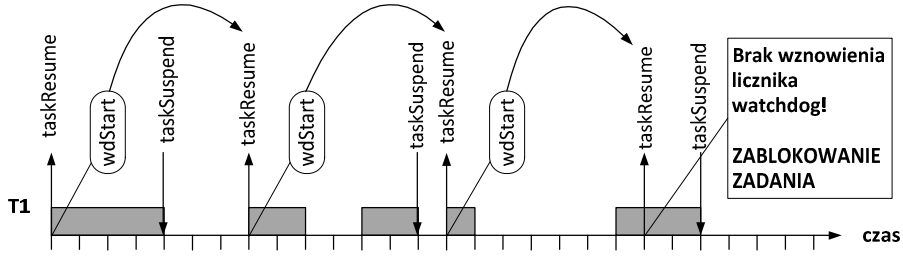
```
1.      int TaskID;
2.      WDOG_ID WatchdogID;
3.      //...
4.      void Regs_wd_fun(int a)
5.      { taskResume(TaskID);}
6.      void Regs(void)
7.      { // Obliczenia wykonywane tylko raz w zadaniu
8.        while(1)
9.        { if (wdStart (WatchdogID,
10.                      sysClkRateGet()/50,
11.                      (FUNCPTR)Regs_wd_fun,
12.                      1
13.           ) == ERROR)
14.         {logMsg("Watchdog %d Error",T_Regs);}
15.          //Obliczenia wykonywane cyklicznie
16.          taskSuspend(0);
17.     }}
```

Listing 1. Schemat programowania zadania cyklicznego w prawidłowo zaprojektowanym systemie

Podczas rozpoczynania obliczeń kolejnej instancji zadania cyklicznego incjalizowany jest programowy moduł czasowo-licznikowy (wywołanie funkcji wdStart()). Moduł, po upłynięciu zadanego przedziału czasowego (sysClkRateGet()/50), automatycznie uruchomi wskazaną funkcję (w przykładowym programie: Regs_wd_fun()). Funkcja ta dokona wznowienia procesu obliczeniowego zadania poprzez wywołanie funkcji taskResume(). Przy prawidłowym zaprojektowaniu systemu część cykliczna zadania powinna zawsze kończyć obliczenia (wywołanie funkcji taskSuspend()) przed ponownym wznowieniem.

W czasie przeprowadzania testów oprogramowania okazało się jednak, że w szczególnych okolicznościach wykonywania systemu – w chwili realizacji sporadycznych operacji wejścia-wyjścia na systemie plikowym – dochodzi do inwersji priorytetów (por. [10, 11, 12]). W konsekwencji jedno z zadań o wyższym priorytecie przestawało spełniać swoje ograniczenia czasowe i przy pokazanym na listingu

1 schemacie programowania dochodziło do jego permanentnego zablokowania. Zaproponowana technika programowania zadań cyklicznych okazała się wadliwa. Zaistniałe zjawisko pokazano schematycznie na rys. 6. Zadanie T1 prawidłowo wznawia swoje obliczenia, o ile nie nastąpi takie opóźnienie jego obliczeń, że w czasie ich wykonywania nastąpi przepełnienie licznika watchdog i w konsekwencji wysłanie rozkazu ponownego rozpoczęcia cyklu obliczeniowego (taskResume()). Zaproponowana na listingu 1 konstrukcja programowa powoduje w takim przypadku ignorowanie "wznowienia" obliczeń przez kolejną instancję zadania. Obliczenia bieżącego zadania są kontynuowane, a następnie zawieszane. Ponieważ wznawianie pracy licznika watchdog odbywało się w kodzie zadania na początku wykonywania cyklu, w zaistniałej sytuacji operacja ta nie ma miejsca. Z punktu widzenia systemu zadanie permanentnie przestaje wykonywać swoje obliczenia.



Rys. 6. Zadania czasu rzeczywistego w oprogramowaniu autopilota

W celu wyeliminowania wskazanego błędu w oprogramowaniu podjęto następujące kroki. Po pierwsze wyłączono powodujący nieprzewidywalne blokady w systemie podprogram obsługi systemu plików zastępując go inną techniką gromadzenia danych podczas wykonywania misji.

```
1.      int TaskID;
2.      WDOG_ID WatchdogID;
3.      int TaskOverrunRatio = 0;
4.      //...
5.      void Regs_wd_fun(int a)
6.      { if (wdStart (  WatchdogID,
7.                       sysClkRateGet()/50,
8.                       (FUNCPTR)Regs_wd_fun,
9.                       1) == ERROR)
10.       { logMsg("Watchdog %d Error",T_Regs); }
11.       if (!taskIsSuspended(TaskID))
12.       { TaskOverrunRatio++; }
13.       taskResume(TaskID);
14.     }
15.
16.     void Regs(void)
17.     {  // Obliczenia wykonywane tylko raz w zadaniu
18.       while(1)
19.       {
20.             //Obliczenia wykonywane cyklicznie
21.             taskSuspend(0);
22.     }}
```

Listing 2. Poprawiony schemat programowania zadania cyklicznego

Po drugie przeprogramowano mechanizm wznawiania obliczeń przez zadania cykliczne (por. listing 2). Pierwsza aktywacja licznika watchdog odbywa się podczas inicjalizacji systemu. Funkcja automatycznie uruchamiana po przepełnieniu licznika watchdog (w listingu 2: Regs_wd_fun()), oprócz dokonywania próby ponownego wznowienia obliczeń przez zadanie cykliczne (wywołanie funkcji taskResume()), przeprogramowuje układ czasowo-licznikowy i wskazuje siebie samą jako podprogram do uruchomienia przy ponownym przepełnieniu (wywołanie funkcji wdStart()). W konsekwencji niezależnie od fazy obliczeń zadania cyklicznego, co zadany przedział czasowy następuje próba wznowienia jego obliczeń. W idealnym przypadku zadanie jest wznawiane cyklicznie, co zadany przedział czasowy. Jeśli dochodzi do próby wznowienia jego obliczeń w chwili gdy wykonywana jest jeszcze jego poprzednia instancja, zadanie "ignoruje" to zdarzenie i kończy obliczenia bieżącej instancji. Jedna instancja obliczeniowa zostaje "opuszczona", a zadanie rozpoczyna obliczenia w następnym cyklu. Fakt nieudanej próby wznowienia obliczeń przez zadanie jest odnotowywany w liczniku TaskOverrunRatio (por. listing 2). Rozwiązanie takie zwiększa żywotność systemu, a z punktu widzenia aplikacji awionicznej – bezpieczeństwo odbywania przelotu. Licznik nieudanych wznowień może zostać zastosowany jako wskaźnik jakości działania systemu i aktywować automatyczne procedury "leczące" system.

3.2. Problem wzajemnego wykluczania w procedurach obsługi przerwań

Komputer autopilota w wytwarzanym systemie komunikuje się z urządzeniami pokładowymi przede wszystkim z zastosowaniem magistrali CAN [21] stosując standard komunikacji przyjęty w specyfikacji CANaerospace [22, 23] (por. rys. 3). Sterowniki magistrali CAN zaimplementowane w systemie VxWorks 6.8 dla mikrokontrolerów Freescale MPC5121e pozwalają na odbieranie komunikatów CAN w procedurze obsługi przerwania. Z kolei wywoływanie gotowych wysokopoziomowych funkcji sterownika powoduje wysyłanie komunikatów na magistralę (por. rys. 5). Odbierane dane gromadzone są we współdzielonych strukturach, z których korzystają zadania czasu rzeczywistego odpowiedzialne za obliczanie sterowania i komunikację z innymi komponentami systemu.

Wymiana informacji pomiędzy standardowymi zadaniami czasu rzeczywistego z zastosowaniem współdzielonego bufora jest dobrze udokumentowana. Wymaga jedynie rozwiązania typowego problemu wzajemnego wykluczania w oparciu o semafory wzajemnego wykluczania (ang. Mutual Exclusion Semaphores) (por. [18, 20]). Podstawowy schemat programowania pokazano na listingu 3, w którym zmienna "shared_data" jest modyfikowana tylko wtedy, gdy funkcja zadania (funcA() lub funcB()) zostanie "wpuszczona" do fragmentu programu chronionego przez semafor "mySem". Przejęcie semafora przez jedną z funkcji na czas modyfikacji współdzielonego zasobu uniemożliwia modyfikację tego zasobu przez inne procesy do momentu zwolnienia semafora.

```
1.      SEM_ID mySem;
2.      void sys_init(void){
3.       mySem = semMCreate (SEM_Q_PRIORITY);}
4.      int shared_data = 0;
5.      void funcA (void){
6.       semTake (mySem, WAIT_FOREVER);
```

```
7.        shared_data += 30;// Sekcja krytyczna
8.        semGive (mySem);}
9.       void funcB (void){
10.       semTake (mySem, WAIT_FOREVER);
11.       shared_data -= 15;//Sekcja krytyczna
12.       semGive (mySem);}
```

Listing 3. Schemat programowania rozwiązujący problem wzajemnego wykluczania z zastosowaniem semaforów

W wytarzanym systemie zachodzi jednak konieczność zapewnienia wzajemnego wykluczania w dostępie do danych współdzielonych pomiędzy procedurą obsługi przerwania od magistrali CAN a zadaniami czasu rzeczywistego. Dane odebrane w procedurze obsługi przerwania powinny zostać przekazane do współdzielonych struktur danych, z których z kolei korzystają zadania sterujące. W takim przypadku zastosowanie semaforów jest zabronione. Za błąd projektowy uznaje się włączenie funkcji próbującej przejąć semafor (semTake()) do ciała procedury obsługi przerwania.

W systemie VxWorks dla komputerów jednordzeniowych rozwiązanie problemu wzajemnego wykluczania podczas wymiany informacji pomiędzy procedurą obsługi przerwania a zadaniem można zrealizować z zastosowaniem mechanizmów blokujących system przerwań i przełączanie zadań (por. listing 4). Jeśli zadanie modyfikuje dane ze współdzielonego zasobu, to powinno najpierw zablokować możliwość wywłaszczenia siebie przez inne zadania (taskLock()), a następnie zablokować system obsługi przerwań (intLock()). W takim trybie pracy programu szeregującego i systemu obsługi przerwań w systemie VxWorks wykonywanym na procesorze jednordzeniowym uzyskuje się gwarancję, że operacje modyfikacji współdzielonych zmiennych nie zostaną przerwane. Na zakończenie tak skonstruowanej sekcji krytycznej przywraca się możliwość obsługi przerwań w systemie (intUnlock()) oraz odblokowuje możliwość wywłaszczenia zadania (taskUnlock()). Po stronie procedury obsługi przerwania należy zapewnić zablokowanie reakcji na przerwania zgłoszone przez inne komponenty systemu. Przełączenie procedury na inną, lub na zadanie staje się już wtedy niemożliwe.

```
1.       int lockKey;
2.       int shared_data = 0;
3.       Task_A ()
4.       { if (taskLock() == OK)
5.         { lockKey = intLock ();
6.              shared_data += 30; // sekcja krytyczna
7.           intUnlock (lockKey);
8.           taskUnlock();}
9.         else
10.        {  // zgłoszenie błędu lub
11.           // rozpoczęcie odzyskiwania
12.        }}
13.      ISR_B ()
14.      {  lockKey = intLock();
15.           shared_data -= 15;// sekcja krytyczna
16.         intUnlock (lockKey);}
```

Listing 4. Schemat programowania rozwiązujący problem wzajemnego wykluczania w komputerach jednoprocesorowych

Analiza dokumentacji systemu VxWorks dla procesorów wielordzeniowych wskazuje jeszcze jedną, bardziej ogólną metodę rozwiązania problemu wzajemnego wykluczania w wymianie informacji pomiędzy procedurami obsługi przerwania i zadaniami. Niezależnie od rodzaju procesora do programowania sekcji krytycznych

można zastosować wirujące blokady dostosowane do wywoływania z poziomu procedur obsługi przerwania (ang. ISR-Callable Spinlocks) (por. listing 5).

Schemat programowania jest podobny do typowych sekcji krytycznych wskazanych na listingu 3. Kluczową sprawą jest jednak implementacja tych mechanizmów wzajemnego wykluczania. Następuje tu, podobnie jak na w listingu 4, umiejętne blokowanie systemu obsługi przerwań oraz przełączania zadań, w razie potrzeby. Wirujące blokady kosztem obciążenia procesora zwiększają również responsywność systemu. Ostatecznie w implementacji wytwarzanego programowania autopilota zdecydowano się na wybranie właśnie tego mechanizmu dostępu do współdzielonych danych.

```
1.      spinlockIsr_t CAN_SpinLock;
2.      int shared_data = 0;
3.      void init_CAN_SpinLock(void)
4.      { SPIN_LOCK_ISR_INIT(&CAN_SpinLock,NULL);}
5.      Task_A ()
6.      { SPIN_LOCK_ISR_TAKE (&CAN_SpinLock);
7.              shared_data += 30; // sekcja krytyczna
8.        SPIN_LOCK_ISR_GIVE (&CAN_SpinLock);}
9.      ISR_B ()
10.     {  SPIN_LOCK_ISR_TAKE (&CAN_SpinLock);
11.             shared_data -= 15; // sekcja krytyczna
12.       SPIN_LOCK_ISR_GIVE (&CAN_SpinLock);}
```

Listing 5. Schemat programowania rozwiązujący problem wzajemnego wykluczania z zastosowaniem wirujących blokad dostosowane do wywołania z poziomu procedur obsługi przerwań

4. Wybrane problemy stosowania sieci Ethernet/IP/UDP w awionice

Struktura systemów sterowania większości statków powietrznych ma charakter rozproszony. Czujniki, systemy odniesienia przestrzennego, człony wykonawcze oraz autopilot są rozmieszczone w różnych punktach konstrukcyjnych obiektu latającego i połączone lokalną siecią komputerową. Najpopularniejszym medium komunikacyjnym w samolotach pasażerskich eksploatowanych obecnie jest sieć zgodna ze standardem ARINC 429 [25]. W samolotach mniejszych i profesjonalnych bezpilotowych statkach powietrznych są stosowane magistrale CAN [21] z protokołem CANaerospace [22, 23]. Przegląd innych popularnych magistrali komunikacyjnych stosowanych w awionice można znaleźć w [26].

Magistrale ARINC 429 i CAN są wystarczające do transferu typowych strumieni sterujących, jednak ich maksymalne przepustowości (100kbit/s dla ARINC 429 oraz do 1 Mbit/s dla CAN) stopniowo stają się barierą rozwojową, zwłaszcza kiedy dochodzi do zwiększenia liczby węzłów w sieci lub wzrostu oczekiwanej ilości przesyłanych danych. Przemysł lotniczy, podobnie jak i inne gałęzie przemysłu, zwraca swoje zainteresowanie w kierunku wprowadzenia sieci zgodnych ze standardem IEEE 802.3 [27]. W fazie wdrożeniowej są dwa awioniczne standardy komunikacji oparte na IEEE 802.3: ARINC specification 664 [28, 30] oraz TTEthernet [29]. Koszty instalacji wymienionych sieci są obecnie znaczące, stąd wdrożenie ich w mniejszych samolotach i SBSP odbędzie się za pewien czas.

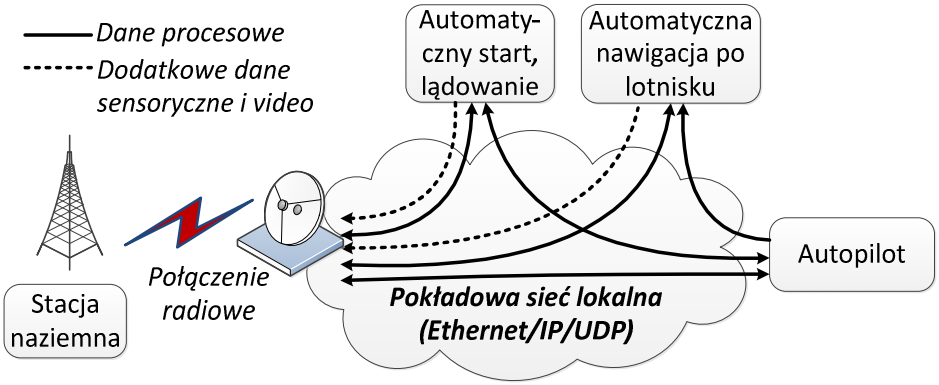
Powszechna dostępność "standardowej" technologii Ethernet/IP/UDP powoduje, że na pokładach obiektów latających wdraża się urządzenia sieci komputerowych zgodnych

z IEEE 802.3, oczywiście dostosowane mechanicznie i elektronicznie do warunków panujących podczas odbywania lotu (por. [31]). Uzyskuje się wtedy znaczące zwiększenie przepustowości pokładowej sieci komputerowej (w porównaniu do standardów CAN i ARINC 429). Przy przygotowywaniu takich wdrożeń należy jednak zwrócić uwagę na ograniczoną przewidywalność czasów transmisji w takiej sieci, która może zależeć przede wszystkim od jej obciążenia. Warto także zauważyć, że współczesne zaawansowane przełączniki Ethernet, w tym te oferowane dla przemysłu lotniczego, oferują dodatkowe mechanizmy zarządzania ruchem w sieci lokalnej. Na uwagę zasługują tu techniki stosujące standard znakowania ramek 802.1Q [32] oraz możliwość interpretacji pól usług zróżnicowanych w pakietach IP [38]. Mogłyby one posłużyć do zwiększenia przewidywalności przesyłania wydzielonych strumieni danych.

Zastosowanie sieci Ethernet/UDP/IP jako platformy integrującej komponenty systemu bezpilotowego statku powietrznego stawia również wyzwania w warstwie aplikacji. Ważnym staje się dobór wysokopoziomowych protokołów komunikacyjnych, a także ewentualne uzupełnienie oprogramowania w celu wykrywania przeciążeń w warstwie komunikacyjnej. Proponowane przez autorów wybrane techniki konfiguracji i programowania komunikacyjnego zostaną wskazane w kolejnych dwóch podrozdziałach.

4.1. Klasyfikacja strumieni danych

W komunikacji wybranych (nowych) komponentów eksperymentalnego systemu bezpilotowego statku powietrznego zastosowano lokalną sieć Ethernet/IP/UDP (por. rys. 3). Na rys. 7 pokazano strumienie danych, które będą w niej przekazywane. Kluczowymi, ze względu na poprawność działania systemu sterowania, są dane procesowe, a więc komunikaty przekazujące informacje o stanie obiektu sterowania (platformie latającej) oraz sygnały zadające bieżące wartości trajektorii lotu. W zależności od aktualnie realizowanego trybu pracy SBSP źródłem zewnętrznych sygnałów zadających mogą być stacja naziemna (tryb sterowania manualnego BSP), moduł automatycznego startu i lądowania (tryb start/lądowanie) lub moduł nawigacji po lotnisku (tryb TAXI). W trybie wykonywania misji autopilot dysponując jej cyfrowym planem może prowadzić BSP autonomicznie, przesyłając do stacji naziemnej jedynie komunikaty raportujące swój stan i położenie.



Rys. 7. Strumienie danych przekazywane w sieci Ethernet/IP/UDP SBSP

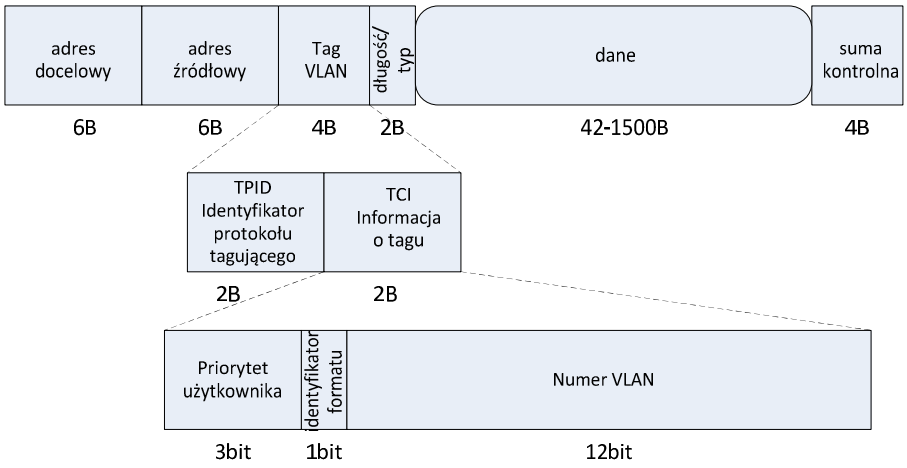
Oprócz danych procesowych w SBSP do stacji naziemnej przekazuje się informacje pochodzące z ładunku platformy latającej. Mogą to być strumienie danych z czujników, czy podsystemów wizyjnych. Nie mają one powiązania z systemami sterowania, ale stanowią dodatkowe obciążenie łączy komunikacyjnych.

Standardowo dane procesowe i dodatkowe strumienie sensoryczne przesyła się oddzielnymi łączami. W niektórych rozwiązaniach zakłada się jednak, że wszystkie strumienie przekazywane będą w obrębie jednej wspólnej cyfrowej sieci komputerowej. Nowym problemem, z punktu widzenia konstruowania takich systemów, jest zapewnienie jak najefektywniejszego sposobu przekazywania danych procesowych, dla których dodatkowe dane sensoryczne stanowią swego rodzaju zakłócenie.

W wytwarzaniu rozproszonych systemów sterowania stosujących standardowe urządzenia sieci Ethernet można zaobserwować różne podejścia. Niektórzy wytwórcy systemów stwierdzają, że sieci standardu Ethernet oferują tak wysoką przepustowość w stosunku do ilości przekazywanych danych procesowych, że nie ma potrzeby wprowadzania dodatkowych mechanizmów arbitrażu regulujących komunikację. Autorzy rozdziału stoją jednak na stanowisku, że współczesne urządzenia sieciowe i systemy operacyjne czasu rzeczywistego mogą być zastosowane do zwiększenia przewidywalności transmisji danych procesowych. Zwiększenie przewidywalności dostarczania danych procesowych odbywać się będzie kosztem wprowadzenia opóźnień w transmisji dodatkowych danych sensorycznych. Wśród standardów, na podstawie których można budować zaawansowane zarządzanie ruchem ramek w sieciach lokalnych obiecujące perspektywy daje IEEE 802.1Q. Współczesne konfigurowalne przełączniki mogą również szeregować ruch ramek interpretując pola protokołu IP.

4.2. Zastosowanie sieci 802.1Q do szeregowania strumieni danych sterujących

Ramka Ethernet w standardzie IEEE 802.1Q, w odniesieniu do typowych, zawiera dodatkowe pole "Tag VLAN" (por. rys. 8). Pierwsza część pola (TPID) jest identyfikatorem protokołu tagującego. Druga zaś (TCI) zawiera podpola priorytetu użytkownika oraz identyfikatora wirtualnej sieci.



Rys. 8. Pola ramki zgodna z IEEE 802.1Q

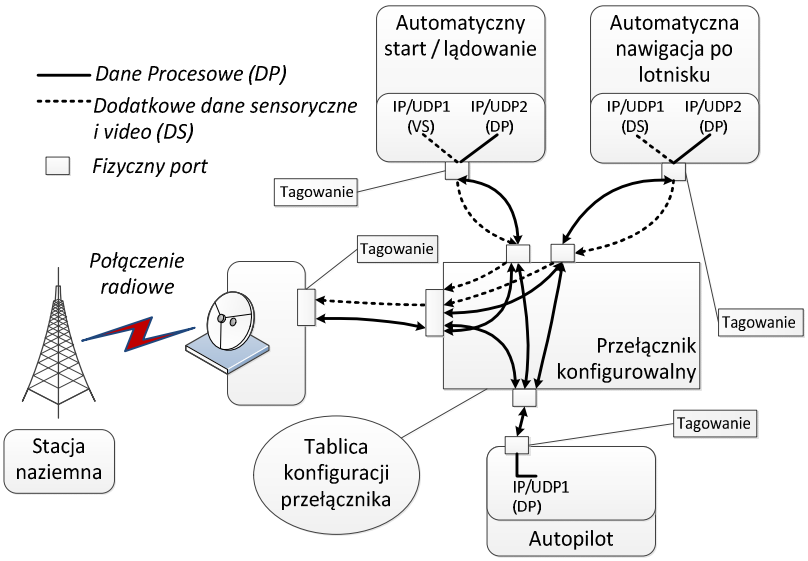
Podstawowym warunkiem uruchomienia komunikacji jest wsparcie wymienionego standardu zarówno po stronie przełączników, jak i węzłów końcowych sieci. Z poziomu programu konfiguracyjnego przełącznika można wtedy uaktywnić algorytmy analizujące zawartość rozszerzonych ramek wpływające na kolejkowanie przesyłanych porcji informacji. Z kolei oprogramowanie urządzeń końcowych powinno dawać możliwości wysyłania ramek już z określonymi znacznikami i priorytetami. W wytwarzanym systemie bezpilotowego statku powietrznego zarówno przełącznik jak i pozostałe urządzenia są wyposażone we wspomniane oprogramowanie.

Na rys. 9 pokazano schematycznie konfigurację sieci i urządzeń końcowych pozwalającą na rozróżnianie strumieni danych. Strumienie danych wprowadzane do sieci zostają oznakowane znacznikami (tagami). Odpowiednio skonfigurowane urządzenie sieciowe jest w stanie szeregować ramki w zależności od zawartych w nich znaczników.

Zastosowanie standardu komunikacji IEEE 802.1Q do eliminacji wpływu danych sensorycznych na dane procesowe może się odbyć, jak się wydaje, na dwa sposoby, które mają odniesienie w dostępnych konfiguracjach urządzeń sieciowych. W pierwszym podejściu w strumienie danych wysyłanych z węzłów sieci wprowadzać można tylko informacje o priorytecie danej ramki. Ramki przenoszące dane procesowe będą otrzymywały wyższe priorytety, od tych transportujących dane sensoryczne. Algorytmy szeregujące przełączników powinny wtedy szeregować ramki w buforach wyjściowych portów komunikacyjnych tylko na podstawie priorytetów i w ten sposób zabezpieczać terminowość przesyłania danych procesowych.

Szeregowanie ramek może się również odbywać z zastosowaniem zagregowanych informacji o priorytecie i numerze wirtualnej sieci. Strumienie danych procesowych można zaklasyfikować do innych wirtualnych sieci niż strumienie danych sensorycznych. Taka klasyfikacja doprowadzi do wirtualnego rozdzielenia ruchu. Poszczególnym sieciom wirtualnym można z kolei w ramach pojedynczego portu przydzielić pasmo przepuszczania. W ten sposób można zapewnić pewien statystyczny rozkład częstości przesyłania poszczególnych typów strumieni. Rozwiązanie takie powinno preferować

przesyłanie danych procesowych przy zarezerwowaniu ustalonych zasobów na dostarczanie danych sensorycznych.

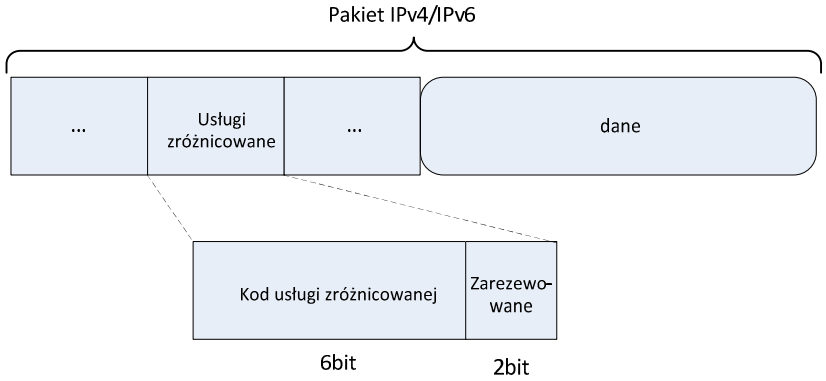


Rys. 9. Konfiguracja przełącznika i urządzeń końcowych umożliwiająca identyfikację strumieni danych w sieci

4.3. Zastosowanie pola usług zróżnicowanych protokołu IP do szeregowania strumieni danych sterujących

Szeregowanie ramek we współczesnych konfigurowanych przełącznikach (i routerach) może odbywać się również na podstawie interpretacji danych zgromadzonych w nagłówkach pakietu IP. Interpretacji zostaje poddane pole nagłówka o nazwie *usługi zróżnicowane*. Nowa interpretacja pola została wprowadzona w dokumencie RFC 2474 [38]. Sześć starszych bitów pola zawiera kod usługi zróżnicowanej, który może być interpretowany jako swego rodzaju priorytet pakietu (por. rys. 10).

Struktura pola jest identyczna w protokołach IPv4 i IPv6. Jeśli do struktury sieci komputerowej wprowadzone zostaną pakiety z odpowiednio wypełnionymi polami usług zróżnicowanych i urządzenia sieciowe będą odpowiednio skonfigurowane, to pakiety zostaną uszeregowane w buforach przełącznika zgodnie ze swoimi "priorytetami". Warto zwrócić uwagę na to, że szeregowanie na podstawie pola usług zróżnicowanych pakietu IP może się odbywać w konfigurowanych przełącznikach a zakres funkcji takich urządzeń sięga więc do wyższych warstw modelu sieci ISO/OSI.



Rys. 10. Interpretacja pola usługi zróżnicowane w pakietach IPv4 i IPv6 zgodnie z RFC 2474

4.4. Wysokopoziomowe problemy integracji z zastosowaniem sieci Ethernet/IP/UDP

Z punktu widzenia programisty, wymiana informacji pomiędzy węzłami w sieci polega na przygotowaniu komunikatu i przekazaniu go interfejsu sieciowego oraz utworzeniu podsystemu odbierającego komunikaty od innych. Dobrze ugruntowanym standardem programowania sieciowego jest interfejs gniazd [33] (ang. Sockets). Wciąż oferuje on uniwersalną metodę integracji wymiany informacji, zwłaszcza w systemach heterogenicznych. W realizacji prezentowanego w publikacji zadania projektowego został on przyjęty jako podstawa to wytwarzania modułu komunikacji sieciowej.

W dokumentach zawierających zalecenia co do przesyłania danych procesowych w awionicznych systemach sterowania z zastosowaniem sieci Ethernet [28,34] preferowanym protokołem jest UDP [35]. Przyjmuje się, że w komunikacji decydujące znaczenie ma regularne przesyłanie aktualnego stanu komponentów systemu. Za dopuszczalne przyjmuje się możliwość utraty pewnej puli komunikatów, a za niekorzystną sytuację uznaje się próbę "odzyskiwania" utraconych przestarzałych danych, co może nastąpić przy zastosowaniu protokołu TCP [36].

Wymiana informacji pomiędzy stacją naziemną a urządzeniami pokładowymi obejmuje informowanie operatora o stanie BSP (położenie i orientacja przestrzenna, realizowany etap misji, stan ładunku) oraz możliwość przesyłania do platformy latającej planu misji lub sygnałów sterujących. Owocem wcześniejszych prac twórców SBSP prezentowanego w publikacji jest autorski protokół komunikacyjny pomiędzy stacją naziemną a BSP [17]. Rozbudowa systemu na potrzeby nowego projektu i konieczność jego integracji z komponentami dostarczanymi przez innych producentów oraz wprowadzenie sieci Ethernet/IP/UDP jako medium wymiany informacji spowodowały konieczność poszukiwania nowego protokołu. Bardzo obiecującym "kandydatem" okazał się standard komunikacji służący do wymiany informacji pomiędzy systemami naziemnymi a BSP opracowany przez NATO - STANAG 4586 [34]. Wymiana informacji w standardzie odbywa się przez przekazywanie komunikatów UDP o ściśle ustalonym formacie. STANAG 4586 nie zastępuje rodziny Ethernet/IP/UDP, oferuje tylko strukturę i logikę komunikatów na kolejnym, wyższym poziomie abstrakcji. Komunikaty zostały

pogrupowane ze względu na treść (komendy, misja, ładunek, stan BSP). Ponadto dla części komunikatów zaproponowano możliwość "odsyłania" informacji do stacji naziemnej na żądanie, co pozwala redukować zajętość łącza komunikacyjnego. Przewidziano również możliwość wymiany komunikatów z systemami zarządzania ruchem lotniczym oraz zaawansowane komendy nakazujące dokonanie rekonesansu nad wskazanym obszarem. Każdy komunikat otrzymuje również pieczątkę czasową, która może posłużyć do identyfikacji ew. nieprzewidywanych opóźnień w komunikacji.

Przyjęcie gotowego wysokopoziomowego standardu komunikacji w systemie nie rozwiązuje jednak wszystkich problemów komunikacyjnych. Specyfika realizowanego zadania projektowego, w którym w zależności od trybu pracy systemu źródłem danych zadających położenie BSP może być odpowiednio: stacja naziemna, system automatycznego startu i lądowania oraz system automatycznej nawigacji po lotnisku, wymaga opracowania dodatkowej warstwy oprogramowania i dokumentów uzgadniających stosowanie standardu w tym szczególnym przypadku projektowym. W konfiguracji systemu będzie trzeba przewidzieć mechanizmy "przejmowania" modułu autopilota przez różne źródła sygnałów zadających parametry lotu. "Bieżące" źródło danych będzie zależeć od fazy wykonywania misji, bądź aktualnej sytuacji operacyjnej w czasie wykonywania misji.

Niezależnie od technik zwiększania przewidywalności transferu danych procesowych (por. podrozd. 4.2 i 4.3) prowadzonych na poziomie ruchu ramek i pakietów w sieci, również oprogramowanie systemu sterowania może posłużyć do kontroli terminowości otrzymywania danych. Częsta utrata danych procesowych oraz zbyt długie opóźnienia w transferze danych mogą być wykrywane na kilka sposobów. Jednym z podejść, wskazanym w pracy [37] jest rozbudowa systemu sterowania o komponenty oceniające obliczone na podstawie otrzymywanych komunikatów sygnały sterujące. Wykrycie zjawisk oscylacji lub zbyt dużego sygnału błędu na poziomie oprogramowania sterującego może świadczyć o problemach z terminowym i odpowiednio częstym otrzymywaniem komunikatów sieciowych. Zastosowanie transferu danych z zastosowaniem standardu Ethernet/IP/UDP pozwala również, bez wprowadzania opóźnień i dodatkowego ruchu w sieci, na uzupełnienie danych procesowych dodatkowymi atrybutami, które mogłyby być zastosowane do detekcji potencjalnych błędów w czasie wykonywania się systemu. Dane procesowe uzupełnić można dodatkowymi znacznikami wskazującymi na nadawcę lub pieczątkami czasowymi. Dodatkowe oznakowanie nadawcy może być gwarantem odbierania danych od ustalonego węzła sieci (por. [37]). Pieczątki czasowe mogą z kolei posłużyć do wyznaczania rzeczywistych opóźnień w komunikacji, co z kolei mogłoby wykorzystane do ewentualnej adaptacji parametrów algorytmów sterujących.

5. Podsumowanie

W pracy zaprezentowano eksperymentalny system bezpilotowego statku powietrznego z perspektywy inżynierii oprogramowania systemów czasu rzeczywistego. Omówiono wybrane praktyczne przypadki wytarzania oprogramowania oraz konfiguracji lokalnej sieci komputerowej, które miały na celu zwiększenie przewidywalności, zachowanie żywotności oraz usprawnienie procesu integracji komponentów systemu.

Skomentowano również dobór standardów i protokołów komunikacyjnych w systemie czasu rzeczywistego integrowanym z zastosowaniem konfigurowalnych urządzeń sieciowych zgodnych z IEEE 802.3Q i RFC 2474.

Prezentowany w rozdziale SBSP przechodzi systematyczną ewolucję. Opracowywane oprogramowanie i konfiguracja sieci jest wciąż przedmiotem intensywnych badań i eksperymentów. Autorzy w najbliższej przyszłości zamierzają dokonać między innymi analizy terminowości przesyłania danych procesowych zakłócanych dodatkowymi danymi sensorycznymi w zależności od rodzaju i konfiguracji urządzeń sieciowych. Osobną ścieżką badawczą jest opracowanie wykonywalnego modelu matematycznego warstw oprogramowania czasu rzeczywistego i sieci komputerowej z zastosowaniem sieci Petriego. Mógłby on posłużyć do wspomagania wytwarzania systemów o podobnych charakterystykach przetwarzania danych i wymiany informacji.

Bibliografia

[1] R. Lozano (Editor), Unmanned Aerial Vehicles: Embedded Control, Wiley-ISTE, 2010.
[2] K. P. Valavanis (Editor), Advances in Unmanned Aerial Vehicles, State of the Art and the Road to Autonomy, Springer, 2007.
[3] T. M. Lam (Editor), Aerial Vehicles, In-Tech, 2009
[4] Strona producenta SBSP http://www.avinc.com/uas/
[5] Strona producenta SBSP http://www.ga-asi.com/products/aircraft/
[6] Strona producenta SBSP http://www.eurotech.com.pl/produkty-i-uslugi/2/lotnictwo-i-awionika
[7] Strona producenta SBSP http://www.wb.com.pl/Rozwiazania,Systemy-C4ISR,Systemy-rozpoznania.html
[8] Strona projektu Europejskiego ERA http://www.era-research-project.org/.
[9] S. Samolej, T. Rogalski, D. Nowak, Problemy implementacyjne systemu sterowania lotem na platformę systemu operacyjnego czasu rzeczywistego VxWorks, Zeszyty Naukowe Politechniki Rzeszowskiej 288, Mechanika 85, RUTMech, t. XXX, z 85 (3.13), s. 515-524, październik grudzień 2013,
[10] G. C. Buttazzo, Hard Real Time Computing Systems, Predictable Scheduling Algorithms and Applications, Springer Science+Business Media, LLC 2011.
[11] A. Burns, A. Wellings, Real-Time Systems and Programming Languages (Fourth Edition), Ada 2005, Real-Time Java and C/Real-Time POSIX, Addison Wesley Longmain, April 2009.
[12] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, A Practitioner's Handbook for Real-Time Analysis, Guide to Rate monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers, 1993.
[13] P. Pop, P. Eles, Z. Peng, Analysis and Synthesis of Distributed Real-Time Embedded Systems, Kluwer Academic Publishers, 2004.
[14] R. Obeimaisser, Event-Triggered and Time-Triggered Control Paradigms, Springer Science + Business Media, Inc., 2005.
[15] C. Wehner, Tornado and VxWorks, What's not in the Manual, 2003.
[16] MP-02 Czajka -strona producenta: http://www.mp-02.pl/
[17] D. Nowak, T. Rogalski, Ł. Wałek, System lot jako latające laboratorium, Technika Transportu Szynowego 12/2015.
[18] VxWorks Kernel Programmer's Guide, 6.8, Wind River Systems, 2009.
[19] Wind River Network Stack Programmer's Guide, Volume 3: Interfaces and Drivers 6.8, Wind River Systems, 2009.
[20] P. Szymczyk, Systemy operacyjne czasu rzeczywistego, Uczelniane Wydawnictwa Naukowo-Dydaktyczne, Kraków, 2003.
[21] M. D. Natale, H. Zeng, P. Giusto, A. Ghosal, Understanding and Using the Controller Area NetworkCommunication Protocol, Theory and Practice, Springer Science+Business Media, LLC 2012.
[22] CAN Areospace, Interface specification for airborne CAN applications V 1.7, http://www.canaerospace.net/tl_files/downloads/canaerospace/canas_17.pdf

[23] ARINC 825, General Standardization of CAN (Controller Area Network) Bus Protocol For Airborne Use, 2015.
[24] L. Buckwalter, Ed., Avionics Databuses, Third Edition, Avionics Communications Inc. 2008.
[25] ARINC Specification 429 Part 1-17: Mark 33 Digital Information Transfer System (DITS), 1996.
[26] L. Buckwalter, Avionics Databuses, Airline Avionics, 3rd edition, 2005.
[27] IEEE 802.3 ETHERNET WORKING GROUP, http://www.ieee802.org/3/.
[28] Aircraft Data Network Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specication 664p7, 2005.
[29] Time-Triggered Ethernet, Certfiable Ethernet Solution for Fault-Tolerant Systems, https://www.tttech.com/technologies/deterministic-ethernet/time-triggered-ethernet/
[30] S. Samolej, T. Rogalski, G. Kopecki, A. Tomczyk, The Integration of a Prototype Pitch Control Application with IMA2G Devices, Automatyka/Automatics, Uczelniane Wydawnictwa Naukowo-Dydaktyczne Akademia Górniczo-Hutnicza w Krakowie, Vol. 17 No. 1, 93-102, 2013.
[31] M. Southworth, Choosing a Rugged Ethernet Switch/Router Solution, Technology White Paper, Curtiss-Wright Defense Solutions, 2015.
[32] IEEE 802.3q IEEE Standards for Local and metropolitan area networks Virtual Bridged Local Area Networks , 2003.
[33] M. P. Donahoo, K. L. Calvert, TCP/IP Sockets in C, Practical Guide for Programmers, Elsevier, 2001.
[34] STANAG 4586 (EDITION 3) Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability, NSA/1235(2012) 4586, 2012.
[35] User Datagram Protocol, RFC 768, 1980.
[36] Transmission Control Protocol, RFC 793, 1981.
[37] T. Rogalski, S. Samolej, A. Tomczyk, ARINC 653 Based Time-Critical Application for European SCARLETT Project, AIAA Guidance, Navigation, and Control Conference, 08 - 11 August, 2011.
[38] Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474

Autorzy i afiliacje

WSTĘP
Lech Madeyski
Katedra Informatyki, Wydział Informatyki i Zarządzania
Politechnika Wrocławska,
Lech.Madeyski@pwr.edu.pl
Piotr Kosiuczenko
Zakład Inżynierii Systemów Informatycznych, Wydział Cybernetyki,
Wojskowa Akademia Techniczna
piotr.kosiuczenko@wat.edu.pl
Marek Bolanowski
Zakład Systemów Złożonych, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
marekb@prz.edu.pl

ROZDZIAŁ 1
Zbigniew Świder
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
swiderzb@prz.edu.pl
Leszek Trybus
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
ltrybus@prz.edu.pl

ROZDZIAŁ 2
Maciej Koryl
Asseco Poland S.A.
Rzeszów
Maciej.Koryl@asseco.pl
Paweł Dymora
Zakład Systemów Złożonych, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
pawel.dymora@prz.edu.pl
Mirosław Mazurek
Zakład Systemów Złożonych, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
miroslaw.mazurek@prz.edu.pl

ROZDZIAŁ 3
Michał Madera
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
michalmadera@gmail.com
Tomasz Śliwa
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
tomeks@kia.prz.edu.pl

ROZDZIAŁ 4
Grzegorz Dec
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
grzegorz.dec@kia.prz.edu.pl
Piotr Woźniak
Politechnika Rzeszowska
ptrwoz@gmail.com

ROZDZIAŁ 5
Dominik Strzałka
Zakład Systemów Złożonych, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
strzalka@prz.edu.pl

ROZDZIAŁ 6
Bartosz Trybus
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
btrybus@prz.edu.pl
Leszek Trybus
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
ltrybus@prz.edu.pl

ROZDZIAŁ 7
Dariusz Rzońca
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
drzonca@kia.prz.edu.pl
Andrzej Stec
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
astec@prz.edu.pl
Bartosz Trybus
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
btrybus@prz.edu.pl

ROZDZIAŁ 8
Paweł Głuchowski
Wydział Elektroniki
Politechnika Wrocławska,
pawel.gluchowski@pwr.edu.pl

ROZDZIAŁ 9
Sławomir Samolej
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
ssamolej@prz.edu.pl
Tomasz Rogalski
Katedra Awioniki i Sterowania, Wydział Budowy Maszyn i Lotnictwa
Politechnika Rzeszowska
orakl@prz.edu.pl
Dariusz Rzońca
Katedra Informatyki i Automatyki, Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska
drzonca@prz-rzeszow.pl

Rada Naukowa
Polskiego Towarzystwa Informatycznego
ul. Solec 38 lok. 103
00-394 Warszawa
ISBN 978-83-946253-3-7