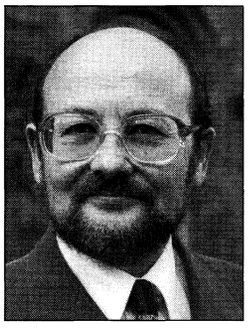
Zbigniew Mrozek
POLITECHNIKA KRAKOWSKA

Metodyka Wykorzystania UML w projektowaniu mechatronicznym

Dr inż. Zbigniew MROZEK



Jest adiunktem na Wydziale Inżynierii Elektrycznej i Komputerowej Politechniki Krakowskiej. Jego specjalnością jest mechatronika oraz modelowanie i symulacja obiektów ciągłych i dyskretnych. Odbywał staże krajowe oraz zagraniczne w Anglii, Belgii, Danii, Grecji, Hiszpanii i Kanadzie
Współpracuje z uczelniami w kraju i za granicą. Uczestniczył w realizacji wielu granatów krajowych i międzynarodowych. Był koordynatorem kilku programów TEMPUS. Jego dorobek naukowo-badawczy obejmuje kilkadziesiąt pozycji w języku polskim i angielskim, głównie z zakresu modelowania i symulacji komputerowej.

Streszczenie

Przekazywanie informacji odgrywa istotną rolę w działaniu urządzeń mechatronicznych i może być łatwo przedstawione na diagramach UML. Zdaniem autora, terminologia i notacja używana w UML może być zaadoptowana do projektowania interdyscyplinarnych systemów mechatronicznych oraz jako narzędzie do sporządzania dokumentacji na wszystkich etapach projektowania.

Abstract

Information transfer plays an important role in operation of mechatronic system. This can be easily presented on UML (Unified Modelling Language) diagrams. Author believes that terminology and notation of visual modelling with UML can be adopted as common language for design of the mechatronic systems and as documentation tool on every design phase.

1. Wstęp

Istotnymi czynnikami warunkującymi sukces nowych produktów jest połączenie wysokich walorów użytkowych i jakości oraz istnienie (bądź stworzenie) zapotrzebowania na dany produkt. Przykładami takich produktów są przykładowo: kserokopiarka, kompaktowy aparat fotograficzny, telefon komórkowy i fax. Wykorzystanie zaawansowanych technologii i nowatorskie podejście w określeniu funkcji nowego wyrobu może zadecydować o jego sukcesie na rynku, ale wymaga interdyscyplinarnego podejścia przy projektowaniu wyrobu i opracowaniu technologii jego produkcji [15]. Oznacza to, że konstrukcja i technologia produkcji wyrobu jest opracowana z uwzględnieniem różnych systemów i technologii, aby uzyskać efekt synergii (synergia: współdziałanie kilku czynników dające łączny efekt skuteczniejszy niż suma ich oddzielnych działań).

Członkowie interdyscyplinarnego zespołu mają różne umiejętności. Posługują się odmiennymi sposobami projektowania i opisu uzyskanych rezultatów. Daje to z jednej strony szansę stworzenia wyrobu, którego nie mógłby zaprojektować żaden z pracujących oddzielnie fachowców. Z drugiej strony stwarza to potrzebę określenia wspólnego języka (terminologii i notacji), który posłuży im do precyzyjnego, łatwego i jednoznacznego porozumiewania się i dokumentowania realizowanych etapów projektu. Powinien on też ułatwić precyzyjne określenie przeznaczenia, funkcji i własności projektowanego produktu. Takim językiem może być opisany dalej UML.

2. Język UML

UML (*unified modeling language*) stworzono w celu modelowania systemów informatycznych. Jest to uniwersalny, obiektowo zorientowany język przydatny do projektowania systemów niezależnie od ich przeznaczenia i od języka, w jakim będzie zaimplementowany docelowy system. UML jest efektem ujednolicenia trzech znanych wcześniej języków: OMT2 (*object modelling technique*) Jamesa Rumbaugha, metody Booch'a (Grady Booch) oraz OOSE (*object oriented software engineering*) Ivara Jacobsona. Zawiera on też pewne elementy języka SDL (*Specification and Design Language*, 1976 CCITT) oraz metodyki E-R (*entity relationship*). Rezultatem pracy dwu pierwszych autorów był UML 0.8 [2-3], opublikowany w roku 1995. Język UML był wielokrotnie ulepszany i dopiero wersja UML 1.3 została zaaprobowana w roku 1999 przez OMG (*Object Management Group*) jako propozycja standardu. Aktualnie dostępna specyfikacja to wersja 1.4 draft [9]. Ponadto są już przygotowywane dokumenty do języka UML 2.0 [14].

Model projektowanego systemu ma postać diagramów graficznych. Każdy rodzaj diagramu pokazuje te elementy przyszłego systemu, które są istotne z wybranego punktu widzenia, a pomija bądź upraszcza inne elementy. Wysoki poziom abstrakcji pozwala na zwięzły opis nawet bardzo dużych systemów. Jest on uzupełniany o opis werbalny w postaci scenariuszy oraz komentarze i inne informacje. Oczywistym jest, że użycie tylko jednego typu diagramu nie jest wystarczające. Praktycznie zawsze wykorzystuje się opisany dalej dwa **diagramy statyczne**:

- diagram przypadków użycia (use cases)
- diagram klas.

Wybór dodatkowych diagramów jest uzależniony od aktualnych potrzeb i doświadczenia projektantów. Są to: **diagramy dynamiczne** (behawioralne):

- diagramy aktywności (*activity*)
  - diagram sekwencyjny
  - diagram współpracy (*collaboration*), zwany też diagramem czynności
- diagram stanu (*statechart*)

diagramy implementacyjne:

- diagram komponentów
- diagram konfiguracji (*deployment*), zwany też diagramem wdrożeniowym

Diagramy i ich elementy można grupować w pakietach i w podsystemach. Pakiety i podsystemy mogą wchodzić w skład innych pakietów i podsystemów.

Pakiety CASE (*computer aided system engineering*) oferują narzędzia wspomagające kolejne etapy komputerowego wspomagania projektowania z wykorzystaniem UML. Umożliwiają automatyczne dokumentowanie prac i wprowadzanych zmian. CASE sprawdza też formalną poprawność i zgodność wykonywanych diagramów z diagramami wcześniej przygotowanymi. Pozwala to na automatyczne wychwycenie błędów formalnych i braku spójności różnych elementów projektu. W rezultacie koszt i strata czasu na dokonanie poprawek i modyfikację projektu są relatywnie niskie, gdyż znaczną ich część wykonuje się na wczesnym etapie realizacji projektu. Stosowanie narzędzi CASE poprawia komfort pracy projektantów i wydaje się niezbędne nawet przy tworzeniu niewielkich systemów. Dodatkowym walorem jest automatycznie generowanie dokumentacji budowanego systemu przez narzędzia CASE.

3. Projektowanie obiektu mechatronicznego w języku UML

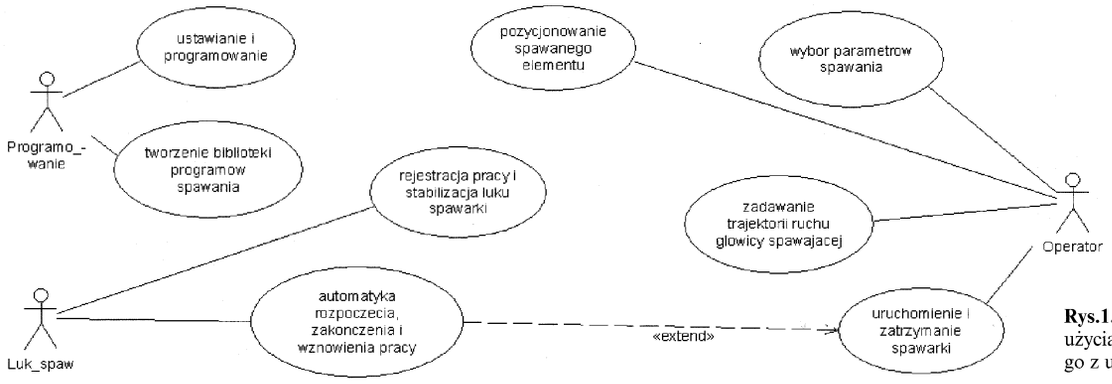
Precyzja i uniwersalność notacji języka UML powoduje, ze ponawiane są próby zastosowania go w dziedzinach innych niż informatyka. Już w roku 1998 wykorzystano UML do projektowania systemu sterowania taśmociągiem i sortownikiem pojemników

[10]. Poniżej podano przykłady wybranych diagramów opisujących spawanie łukowe z użyciem robota.

3.1. Przykład metodyki organizacji prac projektowych

Podstawą sukcesu projektu jest dobry, a więc kompetentny i zgrany zespół projektantów, stosujący narzędzia i metodykę organizacji prac projektowych dobrane odpowiednio do postawionego zadania. Stosowana metodyka winna określać etapy realizacji projektu. Mogą to być tak zwane kamienie milowe czyli warunki, których spełnienie pozwala na przejście do następnego etapu. W realizacji projektu systemu mechatronicznego można wyróżnić następujące fazy (etapy):
– analiza i formalna weryfikacja specyfikacji docelowego systemu
– prototypowanie i testowanie
– wdrożenie

Podczas realizacji kolejnych faz projektu tworzone są modele projektowanego systemu, zazwyczaj w postaci różnych diagramów UML. Są one na bieżąco weryfikowane pod kątem wewnętrznej zgodności i kompletności oraz przez porównanie ze specyfikacją projektowanego systemu. Wszystkie te czynności są w istotny sposób wspomagane przez odpowiednie oprogramowanie CASE i inne narzędzia zależne od specyfiki projektowanego wyrobu. Po zrealizowaniu projektu, zleceniodawca powinien otrzymać gotowy produkt wraz z dokumentacją techniczną i technologią jego produkcji.



Rys.1. Diagram przypadków użycia dla spawania łukowego z użyciem robota

3.2 Analiza i formalna weryfikacja specyfikacji systemu

Etap analizy rozpoczyna się od uzgodnienia ze zleceniodawcą wymagań i wizji przyszłego wyrobu. Obejmuje to określenie granic systemu jako całości oraz zebranie i opisanie najważniejszych jego cech i scenariuszy jego działania. Wszelkie późniejsze zmiany specyfikacji wprowadzane przez zleceniodawcę lub projektantów powinny być właściwie udokumentowane, pisemnie uzasadnione i uzgodnione przez obie strony. Zmiana któregokolwiek z parametrów w dowolnym diagramie jest automatycznie przenoszona do wszystkich innych diagramów, w których występuje dany element. Przykładowo nazwę *aktora* „operator" (rysunek 1) można zmienić na „operator_spawarki". Zmiana ta będzie automatycznie zaktualizowana na wszystkich innych diagramach.

Oprogramowanie RtS [13] standardowo zapamiętuje datę i godzinę oraz autora ostatniej modyfikacji każdego diagramu (patrz rysunek 2), lecz informacja ta jest tracona po każdej następnej zmianie. Specjalnym narzędziem służącym do zarządzania zmianami jest *Rational ClearCase*, a do zarządzania wymaganiami służy *Rational RequisitePro* [12].

3.2.1 Diagram przypadków użycia

W oparciu o wymagania zleceniodawcy i przyszłych użytkowników tworzy się diagramy przypadków użycia. Pozwalają one na graficzne pokazanie własności systemu tak jak są one widziane po stronie użytkownika. W języku UML definiuje się *aktorów*, którzy z zewnątrz oddziaływają na pracę systemu. *Aktorem* może być zarówno człowiek (użytkownik lub operator systemu) jak i dowolne urządzenie znajdujące się na zewnątrz projektowanego systemu. Symbolem *aktora* jest uproszczony rysunek przedstawiający człowieka. Możliwości oddziaływania *aktora* na system (i odwrotnie) są pokazane na diagramie *przypadków użycia* (*use case* diagram). Zdefiniowanie *aktorów* pozwala na precyzyjne określenie granicy pomiędzy projektowanym systemem i jego otoczeniem. Przykładem *aktora* jest „łuk spawalniczy", jeśli będzie traktowany jako zjawisko zewnętrzne względem spawarki (rysunek 1). Parametry łuku są mierzone i rejestrowane przez system, a w wypadku niezajarzenia się łuku lub jego zgaśnięcia, uruchamiane są procedury obsługi wznowienia pracy spawarki.

Wysoki poziom abstrakcji i możliwość pominięcia niepotrzebnych na tym etapie szczegółów pozwalają na sporządzenie prostego i przejrzystego diagramu *przypadków użycia* dla bardzo skomplikowanych systemów. Diagramy przypadków użycia i opisane dalej *scenariusze* stanowią podstawę wszystkich następnych etapów projektowania, testowania i odbioru gotowego projektu. Staranne przygotowanie diagramu *przypadków użycia* we współpracy ze zleceniodawcą pozwala na uszczegółowienie i uściślenie wymagań oraz ułatwia usunięcie ewentualnych sprzeczności już w początkowym etapie realizacji projektu. Przykładowo zleceniodawca może zauważyć brak funkcji: awaryjne zatrzymanie spawarki (diagram *przypadków użycia*, rysunek 1).

Podczas przygotowywania *przypadków użycia* powinna także powstać specyfikacja testów funkcjonalnych. Odłożenie tworzenia tej specyfikacji na później stwarza ryzyko, że testy nie będą obiektywne. Przykładowo wykonawca może dopasować testy do uzyskanych rezultatów, a z drugiej strony zleceniodawca może dążyć do poszerzenia zakresu testów o własności nie ujęte w zleceniu.

3.2.2 Scenariusz

Scenariusz to opisana w języku naturalnym wybrana sekwencja zachowań i komunikatów wymienianych pomiędzy *aktorami* a projektowanym systemem oraz werbalny opis reakcji systemu i *aktorów* na te komunikaty. Jest to poszerzenie i uszczegółowienie czynności opisanych przez diagram *przypadków użycia*. Dla każdego przypadku użycia można podać wiele różniących się pomiędzy sobą *scenariuszy*. Przykładowy scenariusz może wyglądać następująco:
Operator spawarki mocuje na pozycjonerze elementy do spawania. Wybiera z menu program spawania i zadaje trajektorię ruchu głowicy spawającej. Następnie uruchamia automat spawalniczy. Automat włącza wydawanie gazu, wody chłodzącej, obcina końcówkę drutu elektrody, po czym rozpoczyna spawanie. Jeśli łuk elektryczny nie zajarzy się lub zgaśnie podczas spawania, automatycznie ponawia się próbę jego zapalenia..

Scenariusze są używane do tworzenia i weryfikacji diagramu przypadków użycia, diagramu sekwencyjnego, diagramu współpracy i diagramu stanów.

3.2.3 Diagram architektury systemu.

Specyfika systemów czasu rzeczywistego pociąga za sobą potrzebę poszerzenia standardu języka UML. Przykładem jest diagram *architektury systemu*, będący rozszerzeniem diagramu *konfiguracji*. Język UML w wersji 2.0 będzie lepiej dostosowany do projektowania tych systemów.

W odróżnieniu od systemów informatycznych, systemy mechatroniczne mają często dedykowany hardwarowy interfejs użytkownika w postaci specjalizowanych wyświetlaczy, przycisków pokręteł i innych elementów. Własności takiego interfejsu są istotnym poszerzeniem informacji zawartych w diagramie *przypadków użycia* i powinny być uwzględnione podczas analizy specyfikacji projektowania systemu. Tego typu informacje można przedstawić na diagramie *architektury systemu*.

Pokazany na rysunku 2 diagram *architektury systemu* dla spawania łukowego z użyciem robota jest fragmentem wygenerowanej automatycznie dokumentacji systemu projektowanego z użyciem oprogramowanie RtS [13]. Na diagramie pokazano trzy podsystemy dla spawania łukowego z użyciem robota spawalniczego: (1) pozycjoner, (2) szafa sterująca robota z zasilaczem spawarki oraz (3) robot z głowicą spawającą i systemem podawania drutu spawalniczego, gazu i wody. Pokazano też zewnętrznych *aktorów* i połączenia z wykorzystaniem sieci CAN. Klikając w dowolny element tego schematu otwiera się okienko właściwości (*properties*), w którym podane się parametry tego elementu (na przykład nazwa elementu, numer przerwania i adres oraz jego opis werbalny). Dane te są dostępne również we wszystkich innych diagramach, w których wystąpi dany element.

3.2.4 Obiekty i klasy

Kolejnym krokiem będzie identyfikacja obiektów i klas, co posłuży do określenie struktury wewnętrznej systemu, wymaganej dla realizacji czynności opisanych przez przypadki użycia. Analiza rzeczowników, czasowników i innych części mowy występujących w scenariuszu, pomaga identyfikować obiekty i klasy potrzebne do przygotowania diagramów, a także ich atrybuty i metody. Etapu tego nie pokazano w niniejszej pracy.

3.2.5 Diagramy dynamiczne

Diagramy dynamiczne opisują własności behawioralne, czyli działanie projektowanego systemu. Zalicza się do nich diagram *sekwencyjny* i diagram *współpracy* oraz diagram *stanu* (*statechart*). Diagram *sekwencyjny* i diagram *współpracy* są uszczegółowieniem diagramu przypadków użycia. Pokazują one pewną wybraną sekwencję działań aktora oraz reakcję obiektów projektowanego systemu na te działania, zgodnie z wybranym scenariuszem. Diagram *sekwencyjny* i diagram *współpracy* są przydatne zarówno na etapie analizy jak i podczas prototypowania i testowania [11]. Pozwalają one na wykrycie niekompletności i niezgodności specyfikacji wymagań bądź scenariuszy. Pozwalają też zweryfikować poprawność identyfikacji obiektów w systemie. Będą one omówione w oddzielnym opracowaniu.

3.3 Prototypowanie, testowanie i implementacja

Profesjonalne systemy CASE posiadają możliwość symulacji pracy projektowanego systemu, co może być wykorzystane do jego prototypowania i testowania. Możliwość generowania kodu w języku C, C++ lub Java, a często i w innych językach jest wykorzystana przy implementacji systemu. Specyfika projektowanego systemu mechatronicznego może wymagać użycia innych specjalistycznych narzędzi jak SIMULINK i STATEFLOW [11, 14]. Ich użycie będzie znacznie ułatwione poprzez możliwość wykorzystania diagramów UML sporządzonych na wcześniejszych etapach projektowania systemu.

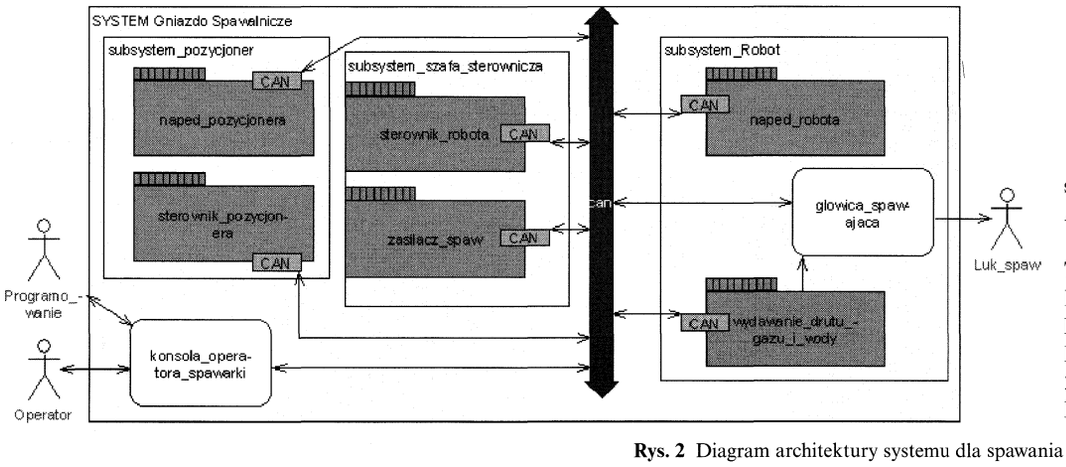
4. Wnioski

W systemie mechatronicznym współpracują elementy i podsystemy o różnej naturze fizycznej. Są to przykładowo układy mechaniczne, elektryczne, elektroniczne oraz oprogramowanie. O jakości całego systemu zadecyduje współpraca jego elementów składowych (daje to efekt synergii) oraz parametry najsłabszego elementu. Złożoność współczesnych konstrukcji oraz konieczność (ze względu na konkurencyjność rynku) skrócenia czasu powstawania produktu powoduje, że projekty nowych wyrobów są zazwyczaj tworzone przez wieloosobowe zespoły fachowców różnych specjalności.

Powszechne jest wykorzystywanie komputerów – nie tylko w części merytorycznej, ale też do zarządzania wymaganiami użytkowników i pracą członków zespołu projektującego. Proces projektowania jest wspierany przez narzędzia CAD i CASE oraz przez odpowiednią organizację pracy zespołu projektantów. Narzędzia CASE istotnie zwiększają efektywność i produktywność, pozwalając na skrócenie czasu tworzenia końcowego produktu. Ułatwiają też wprowadzenie sterowania jakością, zgodnie z normami ISO 9000.

W niniejszej pracy pokazano tylko część możliwości języka UML. Bardziej szczegółowe opisy można znaleźć w cytowanej literaturze, a w szczególności w pracach [1] oraz [4-13].

Podziękowanie: Autor składa podziękowanie firmom *ARTiSAN Software Tools, Inc (GB); Rational Software Corporation (USA) i Premium Technology Sp zoo* za bezpłatne udostępnienie do testowania programów: *Real-time Studio, Rational Rose Suite i Rational Rose RT*.



Rys. 2 Diagram architektury systemu dla spawania łukowego z użyciem robota

Literatura:

1. Bonfe M, Fantuzzi C *Mechatronics objects encapsulation in IEC 1131-3 norm*, Proc 2000 IEEE Int. Conf. on Control Applications, Anchorage, Alaska USA, Sept. 2000,
2. Booch, G. and Rumbaugh, J. Unified Method for Object-Oriented Development v. 0.8. Rational Software Corp., 1995.
3. Booch, G., Jacobson, I., and Rumbaugh, J. *The Unified Modelling Language for Object-Oriented Development* v. 0.9. Rational Software Corp., 1996,
4. Booch, G. Rumbaugh, J. Jacobson I *The Unified Modelling Language User Guide'*, Addison Wesley, 1999 (tłumaczenie: UML: przewodnik użytkownika, WNT 2001)
5. Bruegge B, Dutoit A, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*, Prentice-Hall, 1999,
6. Douglas, B. P. Real-time UML: *Developing Efficient Objects for Embedded Systems*. Addison Wesley, 1998,.
7. Kobryn Cris, *UML 2001 A Standardization Odyssey*, Communications of the ACM October 1999, Vol. 42, No. 10, pp. 28-37
8. Kobryn, C. *Modelling Software Architectures with UML*. Addison Wesley Longman, 2000.
9. OMG Unified Modeling Language Specification (draft), Version 1.4, February 2001.
10. McLaughlin Michael J., Moore Alan, *Real-Time Extensions to UML, Timing, concurrency, and hardware interfaces*, Dr. Dobb's Journal December 1998
11. Mrozek Z, *UML as integration tool for design of the mechatronic system*, in Second Workshop on Robot Motion and Control, pp 189-194, ed. Kozlowski K, Galicki M, Tchoń K, Oct 18-20, 2001, Bukowy Dworek, Poland
12. *Rational Rose Suite; Rational Rose RT* oraz inne oprogramowanie oferowane przez Rational Software Corporation,
13. *Real-time Studio (RtS)*, ARTiSAN Software Tools, Inc. July 2001
14. Uhl T (*edytor*), *Wybrane problemy projektowania mechatronicznego*, Katedra Robotyki i Dynamiki Maszyn, AGH Kraków 1999.

RECENZJE

Wiesław Winiecki, Jacek Nowak, Sławomir Stanik

Graficzne zintegrowane środowiska programowe do projektowania komputerowych systemów pomiarowo-kontrolnych

Wydawnictwo MIKOM, Warszawa X 2001, Wydanie I
Format B5, 280 stron + 8 stron (wkładka), rys. 290, tabel 1, bibliografia poz. 22.

We współczesnej technice często wykorzystywane są komputerowe systemy pomiarowe. Początkowo systemy pomiarowe programowane były przy wykorzystaniu języków wysokiego poziomu, później z użyciem programów zorientowanych obiektowo. Ostatnie lata należą do specjalistycznych, zintegrowanych środowisk programowania. Dzięki nim odpowiedni program może być przygotowany w krótkim czasie przez osoby nie znające klasycznych języków programowania i nie posiadające wiedzy informatycznej. Program taki posiada estetyczny, przyjazny dla użytkownika interfejs, zbliżony wyglądem do panelu tradycyjnych urządzeń pomiarowych. W programie mogą być wykorzystane gotowe procedury złożonych algorytmów obliczeniowych, obsługi modułów pomiarowych i interfejsów standardowych oraz sieci komputerowych.

Omawiana książka zawiera opis pięciu środowisk do przygotowania oprogramowania systemów pomiarowo-kontrolnych (LabWindows/CVI, LabVIEW, VEE, TestPoint, DasyLab) i trzech do oprogramowania systemów automatyki (BridgeVIEW, Lookout, GeniDAQ).

Spośród pięciu przedstawionych środowisk do programowania systemów pomiarowo-kontrolnych, trzy środowiska można zaliczyć do grupy środowisk graficznych (LabVIEW, VEE i DasyLab), w których przygotowywany program powstaje na płaszczyźnie diagramu przez łączenie wybranych obiektów zgodnie z kierunkiem przepływu sygnałów.

Programowanie przy użyciu pozostałych dwóch środowisk odbywa się w inny sposób. Mimo, że nie jest to klasyczne programowanie graficzne zgodne z tytułem książki, bardzo dobrze, że autorzy ze względu na popularność tych środowisk, zamieścili również ich opis. W przypadku LabWindows/CVI, po wyborze z menu odpowiednich poleceń, generowany jest fragment kodu źródłowego w języku C. Środowisko to lubiane jest szczególnie przez informatyków, ponieważ daje im możliwość ingerencji w kod programu. Jeszcze inaczej programuje się w środowisku TestPoint, gdzie wybiera się odpowiednie obiekty i określa jakim działaniom mają one podlegać.

Jeżeli w przypadku środowisk do programowania systemów pomiarowo-kontrolnych przedstawiono najczęściej spotykane i używane w Polsce i na świecie, to inaczej jest dla automatyki przemysłowej. Tutaj autorzy ograniczyli się jedynie do trzech przykładowych, pomijając inne, używane w Polsce równie często. Celem książki nie jest jednak szczegółowe omawianie oprogramowania układów automatyki, a pokazane przykłady służą do przedstawienia problemów, jakie pojawiają się w systemach automatyki w porównaniu z systemami pomiarowymi.

Opis każdego ze środowisk składa się z przedstawienia jego możliwości, sposobu posługiwania się nim oraz z przygotowanych przy jego użyciu przykładowych aplikacji. Jest to zaletą książki – autorom udało się uniknąć samego powielenia informacji zawartych w dokumentacjach programów i uzupełnili je o własne doświadczenia z pracą przy użyciu tych środowisk i opisem przygotowanych przy ich użyciu prostych programów.

Kilka środowisk przedstawiono korzystając z wersji demonstracyjnych, posiadających ograniczone możliwości w stosunku do wersji pełnych. Były one jednak wystarczające do zaprezentowania działania danego środowiska.

Książka jest bogato ilustrowana, większość opisywanych czynności została zilustrowana obrazem odpowiedniego okna opisywanego programu.

Ze względu na zebranie w jednej książce opisów kilku programów, nie można jej traktować jako podręcznika do nauki programowania w danym środowisku. Pozwala natomiast na zapoznanie się z możliwościami danego środowiska i ułatwia pierwszy kontakt z nim. Użytkownicy danego środowiska, w celu pełnego wykorzystania jego możliwości będą zmuszeni do sięgnięcia do dokumentacji programu.

Książka umożliwia porównanie różnych środowisk, ułatwiając decyzję dotyczącą wyboru danego środowiska do dalszej pracy. Przyszli użytkownicy muszą jednak pamiętać, by dokonując wyboru uwzględnili posiadany (czy planowany do zakupu) sprzęt pomiarowy.

Ze względu na często pojawiające się nowe wersje omawianych programów, przygotowanie książki przedstawiającej najnowsze wersje środowisk wymagało od autorów wiele pracy i wiąże się z koniecznością modyfikacji przy kolejnych wydaniach.

Omawiana książka stanowi pozycję długo oczekiwaną przez studentów, pracowników naukowych i inżynierów — projektantów systemów pomiarowych. Z powodu braku konkurencyjnych książek w języku polskim, z pewnością będzie pozycją cieszącą się dużym zainteresowaniem specjalistów.

dr inż. Dariusz Świsulski
Politechnika Gdańska