Grzegorz ŁABIAK, Marian ADAMSKI
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Zastosowanie języka UML w modelowaniu sterowania dyskretnego

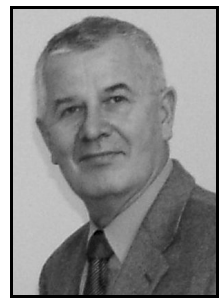
Dr inż. Grzegorz ŁABIAK

Dr inż. Grzegorz Łabiak pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego jako adiunkt. Zainteresowania naukowe koncentrują się wokół nowoczesnych metod projektowania specjalistycznych układów cyfrowych, technik programowania oraz metod formalnych. Jest autorem i współautorem licznych publikacji o zasięgu krajowym i międzynarodowym.

e-mail: G.Labiak@iie.uz.zgora.pl

Prof. dr hab. inż. Marian ADAMSKI

Dyrektor Instytutu Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zainteresowania badawcze obejmują projektowanie systemów cyfrowych realizowanych w postaci mikrosystemów cyfrowych oraz formalnych metod programowania sterowników logicznych. Członek IEEE, IEE, ACM, Polskiego Towarzystwa Elektrotechniki Teoretycznej i Stosowanej oraz Polskiego Towarzystwa Informatycznego.



e-mail: M.Adamski@iie.uz.zgora.pl

Streszczenie

Referat omawia sposób wykorzystania języka UML w modelowaniu sterowania dyskretnego, które jest jednym z etapów projektowaniu sterowników dyskretnych. Proces modelowania sterowania zestawiony jest ze znaną z literatury metodyką modelowania i projektowania programów. W zestawieniu tym szczególny nacisk położony jest na różnice. Główne różnice sprowadzają się do analizy i celu modelowania. W przypadku projektowania oprogramowania główną rolę odgrywa analiza obiektowa, której celem jest sporządzenie modelu danych programu. W przypadku projektowania sterownika dyskretnego, główną czynnością jest analiza zachowania, której celem jest sporządzenie formalnej specyfikacji zachowania.

Słowa kluczowe: UML, sterownik dyskretny, metodyka, analiza zachowania.

UML methodology usage in discrete control modeling

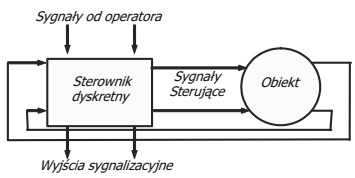
Abstract

The paper describes usage of UML methodology in discrete control modeling, which is one of few stages of discrete controller design. The discrete control modeling process is compared with traditional and well known in literature software development methodology. In the comparison the differences are particularly emphasized. The main differences reduce to analyzing process and modeling aims. In case of software development crucial role plays object analysis which is meant to bring creation of data model. In case of discrete controller design main activity in modeling is behavior analysis which is aimed to formally and precisely specify behavior.

Keywords: UML, discrete controller, methodology, behavior analysis.

1. Wstęp

W wielu niebanalnych projektach inżynierskich często występuję zagadnienie sterowania, w którym układ sterownika kieruje działaniem obiektu sterowanego (rys. 1). Sterowniki takie mogą być projektowane, jako sterowniki dyskretne, tzn. takie, które na swoich wejściach i wyjściach operują wartościami dyskretnymi (w szczególności binarnymi) i implementowane jako układy PLC lub układy cyfrowe realizowane w technologiach programowalnych.



Rys. 1. Układ sterowania dyskretnego
Fig. 1. Discrete control system

Projektowanie takiego układu sterującego może odbywać z wykorzystaniem metod tradycyjnych, w zależności od wybranej technologii.

W przypadku układów PLC podstawowym środkiem specyfikacji zachowania jest sieć SFC [1], a w przypadku układów cyfrowych są to najczęściej: układ sekwencyjny [9], sieć Petriego [5] czy diagram statechart [4, 6, 7].

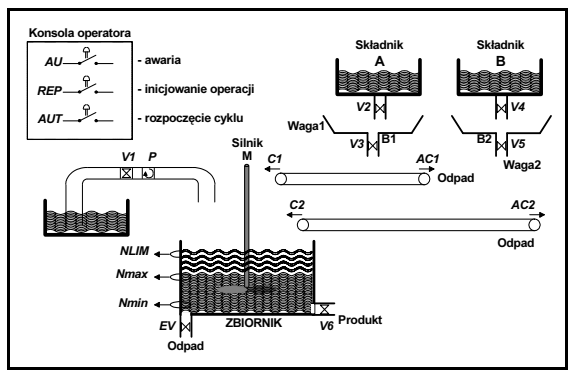
Niezależnie od wybranej technologii w procesie projektowym istniej potrzeba specyfikacji funkcjonalnej sterownika, dokładnego sformułowania zachowania, jasnego i zrozumiałego przedstawienia wybranych aspektów funkcjonowania układu sterowania czy dokumentowania całego projektu. Doskonałym językiem nadającym się do zaspokojenia tych potrzeb jest język UML [3, 10], wywodzący się dziedziny programowania obiektowego. Język UML jest definiowany jako „język do specyfikacji, konstruowania, wizualizacji i dokumentowania wytworów silnie wykorzystujących oprogramowanie" [3, 10]. W swym pierwotnym założeniu twórcy języka ograniczali się do wytwarzania oprogramowania, lecz wkrótce okazała się jego znaczna przydatność w szerokich zastosowaniach inżynierskich [2, 4, 7, 8].

W niniejszej publikacji skupiono się na przedstawieniu praktycznego zastosowania diagramów języka modelujących działanie prostego reaktora chemicznego, którego układ sterujący, za pośrednictwem diagramów statechart lub sieci Petriego, może być implementowany w strukturach programowalnych [5, 6].

2. Rozpoznanie i funkcje obiektu sterowanego

Pierwszym krokiem w projektowaniu sterownika jest rozmowa ze zleceniodawcą projektu, której celem jest rozpoznanie obiektu sterowanego – czyli określeniu czym ten obiekt jest i co ma robić. Polega to na identyfikacji obiektów składowych oraz na ustaleniu podstawowych funkcji obiektu.

W przypadku reaktora chemicznego (rys. 2) podstawowymi obiektami składowymi są: dwie *wagi*, dwa *taśmociągi*, *pompa*, *silnik*, *zbiornik główny*. Ze względów metodologii projektowania warto jest również wyróżnić dwa dodatkowe obiekty, nie należące do klasy obiektów sterowanych, mianowicie obiekt *operatora* i obiekt samego *sterownika*.



Rys. 2. Schemat ideowy reaktora
Fig. 2. Schematic diagram of reactor

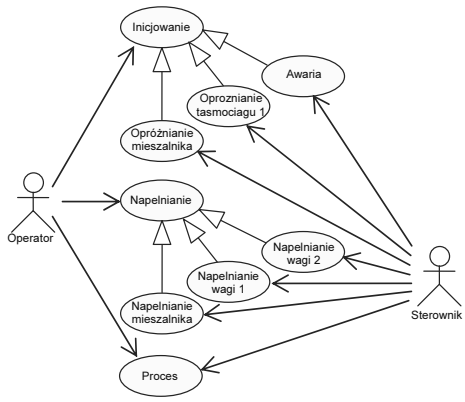
Na tym etapie można wykorzystać diagramy klas i obiektów. Funkcje obiektów są uwarunkowane przebiegiem procesu technologicznego. Najprościej jest zacząć opis działania reaktora od abstrahowania – tzn. od posługiwania się pojęciami ogólnymi, oderwanymi od szczegółów technologicznych. W przypadku, gdy formalny opis działania sterownika będzie specyfikowany notacją uwzględniającą hierarchię, jak np. diagramy statechart, abstrahowanie poczynione na tym etapie, znajdzie swoje dalsze przełożenie.

Cykl technologiczny składa się z trzech etapów: *inicjowania*, *napełniania* i *procesu*. Na etapie *inicjowania* ma miejsce uprzątnięcie reagentów pozostałych po poprzednim cyklu: taśmociągi przekazują zawartość w kierunku wstecznym (*AC1*, *AC2*), a zbiornik główny odprowadza odpad (*EV*). Na etapie *napełnia* odmierzane są składniki reakcji: wagi 1 i 2 są napełniane substratami (*V2*, *V4*) do momentu przekroczenia zadanej masy (*B1*, *B2*), pompa (*V1*, *P*) tłoczy środowisko wodne do zbiornika głównego w zależności od poziomu wody i piany. Na etapie procesu właściwego odmierzone substraty są wprowadzane do zbiornika głównego (*C1*, *C2*, *V3*, *V5*), mieszane (*M*), a powstała substancja jest odprowadzana jako produkt (*V6*). Nadzór na całym cyklem technologicznym sprawuje operator, który: rozpoczyna cykl (*AUT*), inicjuje instalację chemiczną, wstrzymuje działanie obiektu celem obsługi sytuacji awaryjnej (*AU*).

W języku UML diagramem, który opisuje funkcje systemu, a dokładniej w tym przypadku funkcje obiektu sterowanego, jest diagram przypadków użycia (ang. *use case diagram*). Warto zwrócić uwagę, że przy projektowaniu sterowników, jest nieco inna rola przypadków użycia niż ma to miejsce w trakcie projektowania oprogramowania. Otóż przypadki użycia, zaznaczone jako owale, nie oznaczają funkcji projektowanego systemu, lecz oznaczają funkcje obiektu sterowanego.

Kolejną istotną różnicą, między projektowaniem oprogramowania a projektowaniem algorytmu sterowania, są cele działań projektowych. W przypadku projektowania oprogramowania celem jest stworzenie, w oparciu o analizę obiektową, modelu danych (np. w językach *C*/*C++*, *Java*, *C#*), czyli określenie klas i powiązań między nimi, które w dalszych etapach projektowych będą rozwijane w sposób klasyczny przez programistę. Natomiast w przypadku projektowania sterownika, celem jest stworzenie modelu zachowania. Nie ma więc potrzeby przeprowadzania analizy obiektowej, gdyż obiekty (składowe obiektu sterowanego) są określone wprost. Problemem jest jednak przeprowadzenie takiej analizy zachowania obiektu, aby można było stworzyć jak najdokładniejszy model zachowania. W języku UML do analizy zachowania są wykorzystywane m.in. następujące diagramy: *sekwencji*, *współpracy* i *aktywności*, zaś sam model zachowania można formalnie i jednoznacznie opisywać np. diagramami *statechart* lub sieciami *Petriego*.

Rys. 3 przedstawia przypadki użycia z dwóch perspektyw: z perspektywy operatora – spojrzenie bardziej ogólne, oraz z perspektywy sterownika – spojrzenie bardziej szczegółowe. Jak to jest zaznaczone na rysunku, między przypadkami użycia operatora a przypadkami użycia dla sterownika zachodzi relacja generalizacja-specjalizacja (przypadki operatora są ogólne, a przypadki sterownika szczegółowe).



Rys. 3. Diagram przypadków użycia
Fig. 3. Use case diagram

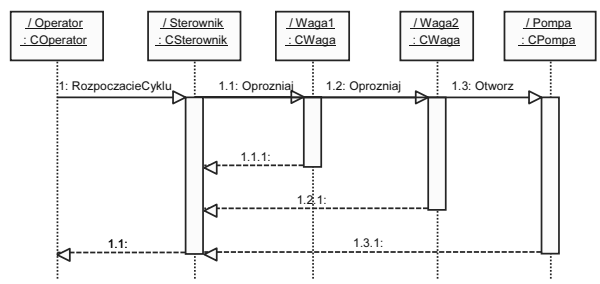
Wydaje się, że przy projektowaniu sterowników, takie związki między przypadkami użycia dla operatora sterownika są czymś naturalnym, gdyż z jednej strony dąży się do maksymalnego uproszczenia pracy operatora (co prowadzi do minimalizacji jego przypadków użycia), a z drugiej zaś strony złożoność sterownika uwarunkowana jest zadaną złożonością sterowanego obiektu. Stąd, przy podległej zależności sterownika względem operatora (patrz rys. 1), zależności między odpowiednimi przypadkami użycia przekładają się na związek typu generalizacja-specjalizacja.

3. Przypadki użycia w działaniu

Po rozpoznaniu obiektu sterowania i określeniu jego funkcji można przejść do bardziej szczegółowej analizy zachowania. Na tym etapie podstawą mogą być wcześniej określone przypadki użycia. Dla każdego z wyodrębnionych przypadków użycia można sporządzić diagram sekwencji (ang. *sequential diagram*).

Diagram sekwencji posiada dwa wymiary. W poziomie umieszczone są obiekty, które biorą udział w realizacji modelowanego przypadku użycia, zaś w wymiarze pionowym reprezentowany jest czas. Strzałki oznaczają operacje jakie są wywoływane na rzecz obiektów biorących kolejno udział w realizacji przypadku użycia. Prostokąty umieszczone na linii życia obiektu symbolizują czas przetwarzania danej operacji przez obiekt.

Rys. 4 przedstawia przykładowy diagram sekwencji dla przypadku użycia *napełnianie*, przy czym na diagramie uwzględniono wszystkie modyfikacje tego przypadku, odnoszące się zarówno do punktu widzenia operatora jak i sterownika. Całość sekwencji rozpoczyna się od operatora, który sygnałem *AUT* (patrz rys. 2) na rzecz sterownika wywołuje operację *RozpoczecieCyklu*. Następnie za kolejne komunikaty odpowiedzialność przejmowana jest przez sterownik, który wysyłając sygnały *V1*, *V2*, *V4* oraz *P* wywołuje na rzecz podstawowych obiektów sterowanych: *Waga1*, *Waga2* i *Pompa* odpowiednio dwie metody *Oprozniaj* i *Otworz*. Na rysunku widać, że strzałki odpowiadające wywoływanym metodom pokrywają się, co oznacza, że metody te są wywoływane w tym samym momencie czasu, lecz ich zakończenie występuje już w różnych momentach, w zależności od przebiegu indywidualnych czynności.



Rys. 4. Diagram sekwencji dla przypadku napełnianie
Fig. 4. Sequential diagram for filling case

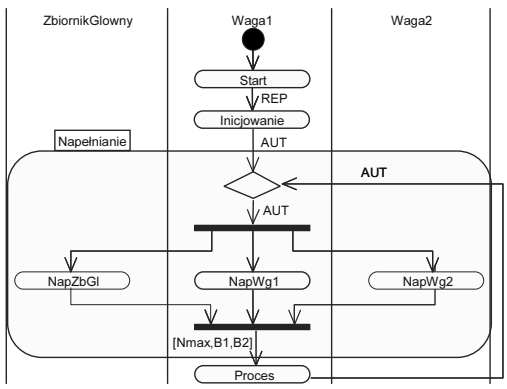
Ten rodzaj diagramów w sposób bardzo czytelny i niewymagający wiedzy inżynierskiej obrazuje szczegóły techniczne przebiegu procesu technologicznego. Z tego powodu może służyć w rozmowach ze zleceniodawcą, który nie musi posiadać wiedzy inżyniera-automatyka. Diagramy te są na tyle jednoznaczne, że mogą służyć jako uzupełnienie dokumentacji projektowanego urządzenia. Kolejnym zastosowaniem diagramów sekwencji obrazujących konkretne przypadki użycia, jest testowanie gotowego sterownika. Zamodelowana sekwencja komunikatów stanowi gotowy scenariusz testowy, częściowo weryfikujący sterownik pod kątem określonego przypadku użycia.

4. Aktywności w układzie sterowania

Diagramy aktywności (ang. *activity diagram*), obok diagramów sekwencji, umożliwiają bardziej szczegółową analizę zachowania. W tym przykładzie nie koncentrują się na wybranym przypadku użycia, co jest ich najczęstszym wykorzystaniem. Ich zadanie nie sprowadza się tylko do modelowania zachowania pojedynczego obiektu, lecz może służyć opisowi aktywności całego złożonego

obiektu sterowanego. W przeciwieństwie do bardzo podobnych diagramów statechart, które to najczęściej skupiają się na obiekcie poddawanym jakiemuś procesowi (lub na samym procesie postrzeganym jako obiekt), diagramy aktywności koncentrują się na przepływie aktywności pojawiających w całym systemie, a w tym przypadku aktywności występujących w reaktorze.

Rys. 5 przedstawia diagram aktywności dla prawidłowego cyklu pracy reaktora, przy czym przepływ aktywności pokazany jest z punktu widzenia operatora. Operator rozpoczyna pracę reaktora od wysłania z konsoli sygnału *REP*, co powoduje że obiekt znajduje się w przypadku *inicjowanie*. Następnie operator wysyła sygnał *AUT*, który sprawia, że na rzecz obiektu sterownika wywoływana jest metoda *RozpoczęcieCyklu* (patrz rys. 4) i jest realizowany przypadek użycia *Inicjowanie*. Na diagramie przypadek ten jest reprezentowany przez stan złożony, w którym za pomocą elementów rozwidlającego i łączącego wyróżniono trzy stany aktywne jednocześnie, odpowiadające napełnianiu wag oraz zbiornika głównego. Dodatkowo informacja o współbieżności wzmocniona jest obecnością tzw. torów pływacki (ang. *swimlane*), które jawnie przyporządkowują wyróżnione części diagramu odpowiednim obiektom. Kolejne cykle bezawaryjnej pracy obiektu są wznawiane przez operatora sygnałem *AUT*, w wyniku którego sterownik na rzecz obiektów wag oraz pompy wywołuje operacje *Oprozniaj* oraz *Otworz* (patrz rys. 4).



Rys. 5. Diagram aktywności dla pracy bezawaryjnej
Fig. 5. Activity diagram for failure-free work

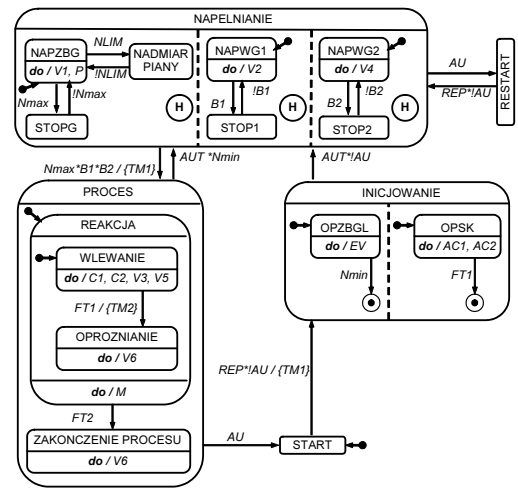
5. Formalny model zachowania

Rozpoznanie obiektów składowych, ustalenie funkcji obiektu, wizualizacja wybranych przypadków użycia, analiza zachowań z różnych perspektyw – wszystkie te czynności mają służyć projektantowi stworzeniu formalnej specyfikacji zachowania sterownika, na tyle precyzyjnej i jednoznacznej, aby mogłaby być podstawą realizacji fizycznej. Autorzy przebadali wykorzystanie dwu następujących notacji: diagramów statechart i sieci Petriego.

Rys. 6 przedstawia przykład pełnego diagramu statechart dla sterownika reaktora chemicznego. Diagram opisuje wszystkie aspekty działania sterownika i zawiera więcej informacji niż wszystkie pozostałe diagramy UML, aczkolwiek w celu zachowania przejrzystości predykaty są tak uproszczone aby oddać sens opisu i przez co mogą występować konflikty między tranzycjami.

Wyróżnionym przypadkom użycia, ogólnym (zanotowanym w wyniku abstrahowania) i szczegółowym, odpowiadają elementy diagramu. I tak np. ogólnemu przypadkowi użycia *Napelnianie* odpowiada stan złożony *NAPELNIANE*, a szczegółowemu *NapelnianieWagi1* automat składający się ze stanów *NAPWG1* i *STOP1*.

Model budowany przy użyciu diagramów UML nie jest ścisłą specyfikacją zachowania, gdyż zakłada się, że model ma stanowić uproszczenie rzeczywistości. Liczba i szczegółowość zastosowanych diagramów w modelu, nie jest w żaden sposób konkretnie ograniczona. Można by podać więcej użytecznych opisów w UML, które tu nie zostały zamieszczone, jak np. diagramy obiektów, diagramy sekwencji dla pozostałych przypadków użycia, czy diagram aktywności dla cyklu pracy z awarią. Ich liczba i jakość, jak się wydaje, uwarunkowana jest przede wszystkim subiektywnym kryterium czytelności i zrozumienia modelowanego systemu, za wyjątkiem jednak diagramów statechart, które całkowicie i jednoznacznie powinny specyfikować zachowanie.



Rys. 6. Diagram statechart dla sterownika reaktora chemicznego
Fig. 6. Statechart diagram for chemical reactor controller

6. Podsumowanie

Zastosowanie UML w modelowaniu sterowania dyskretnego, z pewnymi różnicami, jest bardzo podobne do zastosowania UML w modelowaniu programów. Modelowanie to może przebiegać następująco:

- identyfikacja obiektów składowych obiektu sterowanego – stosuje się diagramy klas i obiektów,
- ustalenie funkcji obiektu sterowanego – diagramy przypadków użycia,
- analiza zachowania obiektu – diagramy statechart, sekwencji, aktywności i współpracy.

Celem nadrzędnym modelowania, oprócz dokumentowania i wizualizacji, jest precyzyjna specyfikacja zachowania w formalnej notacji np. diagramami statechart lub siecią Petriego.

Przedstawiony sterownik został zaimplementowany w układzie FPGA XCV50BG256 z rodziny VIRTEX, zajmując 63 komórki *Slice*, co stanowi ok. 4% układu [4, 7].

Praca finansowana ze środków KBN (nr 4 T11C 006 24).

7. Literatura

[1] M. Adamski, M. Chodań, Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC, Zielona Góra, Oficyna PZ, 2000
[2] G. Bazydło, M. Adamski, Graficzna specyfikacja programów dla sterowników logicznych z wykorzystaniem języka UML, Reprogramowalne Układy Cyfrowe – RUC 2005, VIII Krajowa Konferencja Naukowa, Szczecin, Polska 2005, ss. 65-71
[3] G. Booch, J. Rumbaugh, I. Jacobson, UML przewodnik użytkownika, Wydawnictwa Naukowo-Techniczne, Warszawa 2001
[4] HiCoS, www.uz.zgora.pl/~glabiak
[5] T. Kozłowski, E. L. Dagless, J. M. Saul, M. Adamski, J. Szajna, Parallel controller synthesis using Petri nets, IEE Proceedings-E, Computers and Digital Techniques, Vol.142, No.4, ss.263-271, July 1995
[6] G. Łabiak, M. Adamski, Hierarchiczny diagram stanów i jego odwzorowanie w strukturze logicznej FPGA, Reprogramowalne Układy Cyfrowe - RUC 2004, VII Krajowa Konferencja Naukowa, Szczecin, Polska, 2004, ss. 103–110
[7] G. Łabiak, Wykorzystanie hierarchicznego modelu współbieżnego automatu w projektowaniu sterowników cyfrowych, Oficyna Wydaw. Uniwersytetu Zielonogórskiego (oraz http://zbc.uz.zgora.pl), 2005
[8] Mrozek Z. , Metodyka wykorzystania UML w projektowaniu mechatronicznym, Pomiary Automatyka Kontrola Nr 1, str:.25-28, styczeń 2002
[9] W. Traczyk, Układy cyfrowe. Podstawy teoretyczne i metody syntezy, WNT, Warszawa, 1986
[10] http://www.omg.org

Artykuł recenzowany