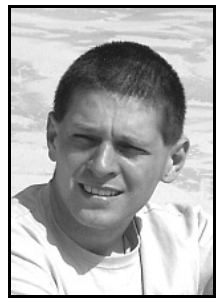
Michał GROBELNY, Iwona GROBELNA
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI,
ul. Szafrana 2, 65-246 Zielona Góra

Diagramy aktywności języka UML i sieci Petriego w systemach sterowania binarnego – od transformacji do weryfikacji

Mgr inż. Michał GROBELNY

W roku 2007 ukończył studia na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego. Absolwent Zintegrowanych Studiów Zagranicznych Uniwersytetu Zielonogórskiego i Fachhochschule Giessen-Friedberg (Niemcy). Obecnie uczestnik studiów doktoranckich. Zainteresowania naukowe obejmują metody specyfikacji osadzonych systemów sterowania. Członek Polskiego Towarzystwa Informatycznego.

e-mail: M.Grobelny@weit.uz.zgora.pl



Mgr inż. Iwona GROBELNA

Absolwentka Wydziału Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego oraz Fachhochschule Giessen-Friedberg (Niemcy). Od marca 2008 zatrudniona na stanowisku asystenta w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Studentka studiów doktoranckich. Zainteresowania naukowe obejmują metody weryfikacji specyfikacji systemów osadzonych. Członek Polskiego Towarzystwa Informatycznego.

e-mail: I.Grobelna@iie.uz.zgora.pl

Streszczenie

Język UML jest technologią powszechnie stosowaną w świecie naukowym oraz w przemyśle. Sieci Petriego są modelem matematycznym ogólnego zastosowania ugruntowanym od wielu lat. Obie te techniki doskonale nadają się do specyfikacji procesów sterowania. Jednakże jako odmienne, każda z nich posiada unikatowe właściwości. Technika weryfikacji modelowej jest jedną z metod formalnej weryfikacji specyfikacji pozwalającą na zdiagnozowanie błędów w specyfikacji wymagań albo w opisie modelu. Artykuł przedstawia metodę transformacji pomiędzy obiema wymienionymi technikami specyfikacji w celu formalnej weryfikacji projektu sterowania opisanego w języku UML.

Słowa kluczowe: diagramy aktywności UML, sieci Petriego, weryfikacja modelowa.

UML activity diagrams and Petri nets in binary control systems – from transformation to verification

Abstract

Unified Modeling Language (UML) [1-3, 5, 6-8] is commonly used in scientific and industrial world. Petri nets [9] are mathematical model used for a long period of time. Both techniques are well suited for control processes specification. However, they are quite different. Each technique has its own unique properties. Model checking technique [14-17] is one of formal verification methods [18] for specifications. It allows detecting errors either in requirements specification or in model description. The paper presents the method for transformation between both mentioned specification techniques – from UML activity diagram (Fig. 1) to Petri net (Fig. 4), using some defined rules [10, 11]. Mapping of particular elements is presented in Table 1. Petri net after direct transformation may include redundant places which can be afterwards removed. Then, it is possible to formally verify control process described in UML. The proposed model checker tool is NuSMV [20]. NuSMV (Fig. 5) compares model description (Fig. 6 – 8) and requirements (Fig. 9) which have to be fulfilled. The requirements are defined using temporal logic. If a specified requirement may not be fulfilled, appropriate counterexamples are generated (Fig. 10) which allow detecting an error source. Then, the specification can be corrected and model checking process can start again, sometimes including only the particular part of a designed system.

Keywords: UML activity diagrams, Petri nets, model checking.

1. Wprowadzenie

Język UML jest technologią powszechnie stosowaną w świecie naukowym oraz w przemyśle. Jego niewątpliwymi zaletami są przejrzystość oraz intuicyjność diagramów, które są zrozumiałe dla szerokiego grona osób. Diagramy aktywności UML w wersji 2.x wzorowane były na sieciach Petriego, posiadają zatem wiele podobieństw, ale także różnic.

Sieci Petriego mogą zostać formalnie zweryfikowane. Technika weryfikacji modelowej pozwala na wykrycie błędów w specyfikacji behawioralnej procesu sterowania. Właściwości systemu definiowane są przy użyciu formuł logiki temporalnej i jej podstawowych operatorów.

Artykuł przedstawia metodę transformacji diagramów aktywności języka UML do sieci Petriego sterowania oraz przedstawia możliwość weryfikacji modelowej wynikowego diagramu. Takie postępowanie umożliwia badanie spełnialności założeń projektu specyfikowane przy pomocy języka UML z wykorzystaniem powszechnie dostępnych narzędzi weryfikacji używanych przy pracy z sieciami Petriego.

Artykuł podzielony jest następująco. W rozdziale drugim zaprezentowane zostały dwa sposoby specyfikacji procesów sterowania – diagramy aktywności języka UML i sieci Petriego – oraz sposób transformacji pomiędzy nimi wraz z odwzorowaniem poszczególnych elementów obu typów diagramów. Rozdział trzeci prezentuję technikę weryfikacji modelowej projektu sterowania popartą analizowanym przykładem. Rozdział czwarty podsumowuje artykuł oraz przedstawia plany dalszych badań.

2. Specyfikacja procesu sterowania

Specyfikacja procesu sterowania stanowi fundament projektu urządzenia i z tego powodu jest jednym z najbardziej kluczowych momentów cyklu projektowania. Na tym etapie projektu określany jest sposób zachowania sterownika oraz właściwości jego pracy. Diagramy aktywności języka UML oraz sieci Petriego są jednymi z możliwości graficznej reprezentacji specyfikacji procesu sterowania.

2.1. Diagramy aktywności (czynności) języka UML

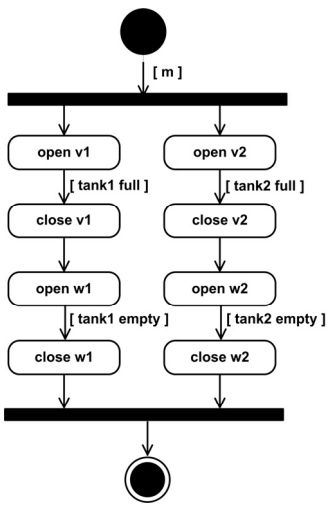
Notacja UML (ang. *Unified Modeling Language*) [1, 2, 3] została wprowadzona przez konsorcjum OMG (Object Management Group, [4]) jako język służący do specyfikacji, wizualizacji oraz dokumentacji oprogramowania. Technologia usprawnia przepływ informacji pomiędzy członkami zespołu i umożliwia lepsze zrozumienie zachowania systemu. Behawioralny projekt systemu osadzonego [5] można sporządzić przy wykorzystaniu wybranych typów diagramów UML – diagramów aktywności, maszyn stanów czy też diagramów sekwencji. Od pewnego czasu UML jest językiem dynamicznie wkraczającym w coraz to nowe dziedziny nauki i przemysłu. Pojawia się w obszarach zupełnie niezwiązanych, czasami odległych [6], od pierwotnie dedykowanych zastosowań, takich jak inżynieria oprogramowania czy projektowanie relacyjnych baz danych.

Notacja UML w wersji 2.x wprowadziła nowe typy diagramów, a także wiele usprawnień do już istniejących diagramów aktywności, które stały się semantycznie podobne do sieci Petriego.

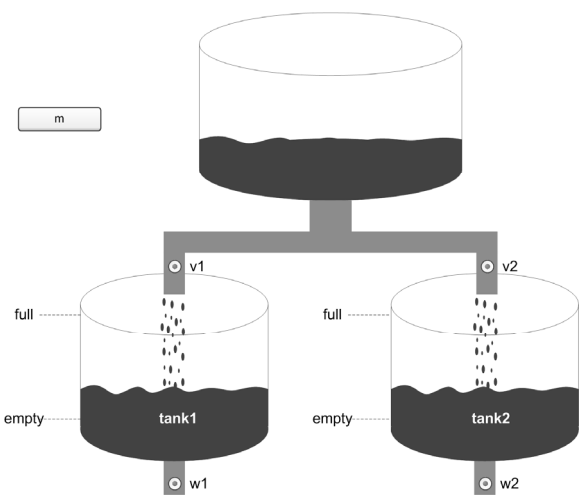
Diagramy aktywności są powszechnie wykorzystywane w dziedzinie biznesu czy modelowaniu przepływu informacji [6, 7],

w behawioralnym projektowaniu oprogramowania oraz systemów osadzonych, jak również we współbieżnym projektowaniu sprzętu i oprogramowania (software/hardware co-design) [8]. Systemy osadzone są zwykle definiowane jako zbiór aktywności wykonywanych przez zewnętrznych aktorów lub stany wewnętrzne.

Specyfikacja procesu sterowania może zostać zapisana przy wykorzystaniu diagramu aktywności UML (rys. 1). W przykładzie rozpatrywany jest prosty osadzony system do napełniania dwóch zbiorników pewną cieczą (rys. 2). Początkowo, oba zbiorniki są puste. Po naciśnięciu przycisku *m*, otwierają się zawory *v1* i *v2*. Proces napełniania się zbiorników jest procesem współbieżnym. W momencie, gdy dany zbiornik jest już pełny, odpowiedni zawór wlewający ciecz zostaje zamknięty, otwiera się natomiast zawór wylewający ciecz (odpowiednio *w1* lub *w2*), który z kolei będzie zamknięty, gdy dany zbiornik zostanie opróżniony.



Rys. 1. Diagram aktywności UML
Fig. 1. UML activity diagram



Rys. 2. Model rzeczywisty
Fig. 2. Real model

2.2. Sieci Petriego

Sieci Petriego są modelem matematycznym ogólnego zastosowania wprowadzonym na początku lat sześćdziesiątych ubiegłego stulecia (Carl Adam Petri, 1962). Opisują relacje pomiędzy warunkami i zdarzeniami. Obecnie są wykorzystywane w wielu gałęziach przemysłu, do planowania i kontrolowania przepływu produkcji, projektowania i programowania sterowników logicznych oraz syntezy oprogramowania systemowego.

Przy wykorzystaniu podstawowych elementów sieci (takich jak miejsca czy też tranzycje) możliwe jest określanie takiego zachowania jak równoległość i współbieżność, wybór, synchronizacja czy też współdzielenie zasobów [9].

Sieci Petriego są bardzo potężnym narzędziem do modelowania. Ich niewątpliwą zaletą jest możliwość weryfikacji założeń przy wykorzystaniu mechanizmów formalnych. Projekty, które są wyspecyfikowane przy wykorzystaniu sieci Petriego mogą zostać poddane badaniu spełnialności założeń postawionych im w początkowych fazach projektowania. Taki proces nazywa się weryfikacją formalną. Jedną z tego typu technik jest weryfikacja modelowa. Po przeprowadzeniu weryfikacji projektant otrzymuje raport, w którym wypunktowane są potencjalne miejsca odbiegające od założeń projektowych.

2.3. Transformacja

Diagramy aktywności języka UML mogą zostać przetransformowane do sieci Petriego. Transformacja odbywa się zgodnie z ściśle określonymi regułami [10, 11]. Akcjom diagramu aktywności (przedstawionym w zaokrąglonych prostokątach) odpowiadają w tym przypadku tranzycje sieci Petriego, natomiast warunki wejściowe do akcji (przedstawione w kwadratowych nawiasach) tłumaczone są na tranzycje z warunkiem odpalenia [12, 13]. Takie traktowanie możliwe jest przy założeniu, że warunki wejściowe do akcji traktuje się identycznie jak bloki decyzyjne umieszczone przed wspomnianą akcją. Zestawienie odwzorowania poszczególnych węzłów diagramu aktywności w elementach sieci Petriego zostało przedstawione w Tabeli 1. Miejsca w sieciach Petriego są elementami pośrednimi pomiędzy tranzycjami i jako takie nie posiadają bezpośredniej interpretacji w diagramach aktywności. W diagramach aktywności akcje wykonywane są bezpośrednio jedna po drugiej. W przypadku oczekiwania procesu na jakieś sygnały, proces jest „zawieszony" pomiędzy akcjami. Semantyka sieci Petriego wymusza naprzemienne stosowanie tranzycji oraz miejsc (nie jest możliwe połączenie ze sobą dwóch jednakowych elementów sieci Petriego).

Należy także zauważyć pewną specyfikę elementów diagramów aktywności. Mowa jest tutaj o synchronizacji procesów współbieżnych w węzłach *join*. Składnia języka UML wymusza synchronizowanie procesów we wspomnianych węzłach, jednakże nie wymusza jawnego specyfikowania miejsc oczekiwania na poszczególne podprocesy.

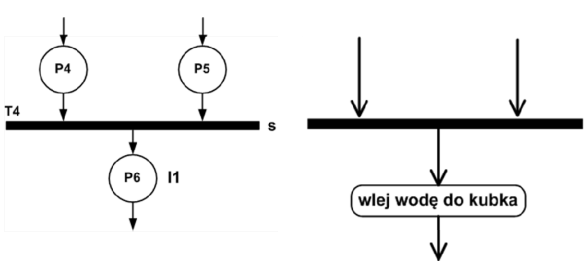
W sieciach Petriego koniec procesu współbieżnego opisywany jest przez tranzycję z wieloma wejściami i jednym wyjściem. Składnia przewiduje wykonanie tranzycji tylko i wyłącznie wtedy, kiedy wszystkie miejsca dołączone do tranzycji zostaną osiągnięte. Należy podkreślić, iż współbieżność w sieciach Petriego jest z założenia synchronizowana. Z tego powodu w specyfikacji procesów współbieżnych miejsca tuż przed tranzycją łączącą są miejscami oczekiwania (miejsca *P4* i *P5* na rys. 3).

Diagramy aktywności nie posiadają miejsc oczekiwania, stąd przy transformacji takie miejsca muszą zostać dodane do wynikowej sieci Petriego.

Przy transformowaniu obu typów diagramów mogą powstawać miejsca nadmiarowe. Mowa tutaj o elementach, które pojawiają się zgodnie z zasadami transformacji, jednak nie posiadają wpływu na specyfikowany proces. Wymusza to przeprowadzenie redukcji miejsc nadmiarowych tuż po bezpośrednim przetransformowaniu diagramu aktywności na sieć Petriego. Sieć wynikowa zazwyczaj opisuje proces sterownia krok po kroku, a w rzeczywistości wiele czynności w procesie sterowania wykonywanych jest jednoczenie i nie potrzeba stanów pośrednich. Taki sposób transformacji w jasny i przejrzysty sposób przedstawia sekwencję sygnałów wejściowych oraz wyjściowych. Jednakże, liczba miejsc i tranzycji staje się przez ten fakt dość duża, co może utrudniać analizę zachowania bardziej złożonych procesów sterowania.

Tab. 1. Odwzorowanie elementów diagramów aktywności języka UML do sieci Petriego sterowania
Tab. 1. Mapping of UML activity diagram elements to control Petri nets

Diagram aktywności	Sieć Petriego
Początek Koniec	P1 Zapętlenie
akcja Akcja	T1 Tranzycja
Przepływ	Przepływ
Brak odpowiednika	P1 Miejsce
Rozdzielenie (fork)	T1 Tranzycja początek procesu współbieżnego
Złączenie (join)	T10 Tranzycja koniec procesu współbieżnego
Decyzja	P2 T5 T6 P3 P4 Decyzja
Złączenie (merge)	P2 P3 T2 T3 P2 Złączenie



Rys. 3. Synchronizacja w sieciach Petriego oraz diagramach aktywności
Fig. 3. Synchronization in Petri nets and activity diagrams

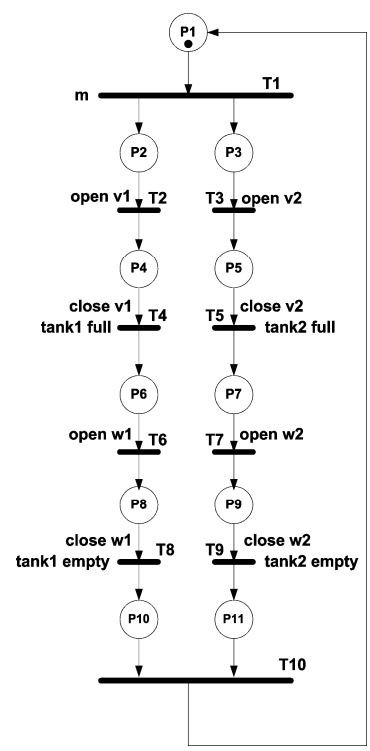
Rozpatrywany przykład prezentuje podstawowe zasady transformacji. Wejściowy diagram aktywności języka UML (rys. 1) transformowany jest do sieci Petriego (rys. 4). Akcje diagramów aktywności w trakcie transformacji traktowane są jako tranzycje [12, 13].

Sieć Petriego po bezpośredniej transformacji przedstawiona jest na rys. 4. Sieć zawiera 11 miejsc i 10 tranzycji. Sieć przedstawia krok po kroku zachowanie procesu sterowania bazując na jego interpretacji w postaci diagramu aktywności.

Punkt początkowy diagramu aktywności posiada odpowiadające mu miejsce *P1* w sieci Petriego. Diagram aktywności zawiera punkt końcowy procesu, zaś sieć Petriego ma charakter cykliczny. Stąd też ostatnia tranzycja *T10* połączona jest z miejscem początkowym *P1*, co odpowiada elementowi końcowemu diagramu aktywności UML. Dodatkowo warunki wejściowe do akcji potraktowane zostały tak, jakby były zapisane jako bloki decyzyjne przed akcją. Stąd każdemu warunkowi wejściowemu z diagramu aktywności w sieci Petriego przypisana została tranzycja z odpowiednim warunkiem odpalenia. Przykładowo akcja `open v1` odpowiada tranzycji *T2*, zaś akcja `close w2` odpowiada tranzycji *T9*. Całe zestawienie odwzorowania znajduje się w tabeli 2. Inny charakter mają tranzycje *T1* i *T10*. Nie posiadają one swoich odpowiedników w diagramie aktywności w postaci akcji, tylko odpowiadającym im węzłom rozpoczynającym i kończącym proces współbieżny. Tranzycji *T1* odpowiada węzeł *fork*, zaś tranzycji *T10* węzeł *join*.

Tab. 2. Odwzorowanie akcji diagramu aktywności języka UML w tranzycjach wynikowej sieci Petriego
Tab. 2. Mapping of UML activity diagram action to Petri net transitions

Diagram aktywności UML	Sieć Petriego
`open v1`	*T2*
`open v2`	*T3*
`close v1`	*T4*
`close v2`	*T5*
`open w1`	*T6*
`open w2`	*T7*
`close w1`	*T8*
`close w2`	*T9*
węzeł rozgałęzienia (*fork*)	*T1*
węzeł złączenia (*join*)	*T10*



Rys. 4. Sieć Petriego
Fig. 4. Petri net

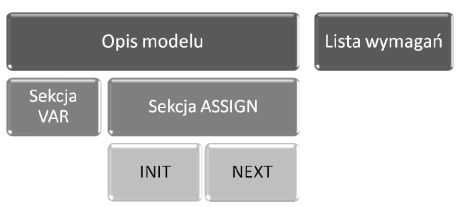
Zauważyć także należy dodatkowe miejsca synchronizacji (miejsca *P10* oraz *P11*) w wynikowej sieci Petriego. Są to elementy jawnie nieobecne w diagramie aktywności, jednakże składnia UML wymusza synchronizację w węźle *join*, stąd konieczność uwzględnienia tych elementów.

3. Weryfikacja modelowa

Technika weryfikacji modelowej [14, 15, 16, 17] jest jedną z metod formalnej weryfikacji specyfikacji (przegląd metod przedstawiony jest w pracy [18]) i można ją potraktować jako alternatywę lub uzupełnienie dla interpreterów czy maszyn wirtualnych [19] sprawdzających poprawność specyfikacji metodą symulacji. Weryfikacja przeprowadzana jest automatycznie przez narzędzia wnioskowania komputerowego (ang. *model checker*). Danymi wejściowymi jest opis modelu oraz lista wymagań stawianych projektowanemu systemowi. Wymagania zdefiniowane są przy pomocy logiki temporalnej. Należy mieć na uwadze fakt, że tylko wyspecyfikowane właściwości zostaną sprawdzone. Narzędzie weryfikujące przeprowadza weryfikację systemu poprzez sprawdzenie czy dany model spełnia stawiane mu wymagania oraz zwraca odpowiedź, czy model i jego specyfikacja są spójne, zaś w przypadku gdy tak nie jest – dodatkowo wygenerowany zostaje kontrprzykład. Kontrprzykłady pozwalają na ręczne przeanalizowanie nieprawidłowych sytuacji. Formalna metoda weryfikacji modelowej pozwala na zdiagnozowanie błędów w specyfikacji wymagań albo w opisie modelu.

Specyfikacja procesu sterowania w postaci sieci Petriego została zweryfikowana przy wykorzystaniu temporalnej logiki rozgałęzionej CTL i narzędzia NuSMV [20] w aktualnej wersji 2.4.3. Temporalna logika rozgałęziona CTL jest częściej wykorzystywana w przemyśle niż liniowa logika temporalna LTL [17].

Plik wejściowy do narzędzia NuSMV (rys. 5) zawiera opis modelu oraz listę stawianych mu wymagań. Opis modelu zawiera kolejno definicję zmiennych oraz możliwych wartości jakie mogą przyjąć (sekcja `VAR`, rys. 6), przypisania początkowych wartości zmiennym (sekcja `ASSIGN`, słowo kluczowe `INIT`, rys. 7) oraz przypisania następnych wartości zmiennym (sekcja `ASSIGN`, słowo kluczowe `NEXT`, rys. 8). W ostatnim przypadku z lewej strony dwukropka podane są warunki, które muszą zostać spełnione, aby zmienna przyjęła wartość podaną z prawej strony dwukropka. Wartość `1` podana jako warunek oznacza wszystkie nie uwzględnione wyżej sytuacje, zaś nazwa zmiennej z prawej strony oznacza, że zmienna nie zmienia w danym przypadku swojej wartości.



Rys. 5. Weryfikacja w NuSMV
Fig. 5. Verification in NuSMV

```
VAR
   tank1 : {empty, ok, full};
   tank2 : {empty, ok, full};
   v1    : {open, closed};
   v2    : {open, closed};
   w1    : {open, closed};
   w2    : {open, closed};
   m     : {true, false};
```

Rys. 6. Deklaracja zmiennych w opisie modelu
Fig. 6. Variable definition in model description

```
ASSIGN
      init(tank1) := empty;
      init(tank2) := empty;
      init(v1)    := closed;
      init(v2)    := closed;
      init(w1)    := closed;
      init(w2)    := closed;
      init(m)     := false;
```

Rys. 7. Przypisania początkowych wartości zmiennym w opisie modelu
Fig. 7. Assignment of initial values to variables in model description

```
...
next(tank1) := case
     tank1 = empty & v1 = closed : empty;
     tank1 = empty & v1 = open : ok;
     tank1 = ok & v1 = open : {ok, full};
     tank1 = full & w1 = open : ok;
     tank1 = ok & w1 = open : {ok, empty};
     1 : tank1;
esac;
...
```

Rys. 8. Przypisania następnych wartości zmiennym w opisie modelu (fragment)
Fig. 8. Assignment of next values to variables in model description (fragment)

Lista wymagań (rys. 9) zawiera żądane właściwości systemu, które będą następnie weryfikowane. Pośród przedstawionych wymagań znajdują się zarówno wymagania dotyczące bezpieczeństwa (sytuacje, które nie mogą się zdarzyć), jak i wymagania dotyczące żywotności (sytuacje, które muszą się zdarzyć). Przykładowo, pierwsza zdefiniowana właściwość określa, że nigdy nie powinna mieć miejsca sytuacja, gdy zawór wlewający i wylewający ciecz do/ze zbiornika pierwszego będą jednocześnie otwarte.

```
CTLSPEC AG !(v1 = open & w1 = open);
CTLSPEC AG (v1=open -> AF (tank1 = ok | tank1 =
full));
CTLSPEC A [v1 = open -> v1 = open U !(tank1 = full)];
CTLSPEC A [w1 = open -> w1 = open U !(tank1 = emp-
ty)];
CTLSPEC EF tank1 = full;
```

Rys. 9. Lista wymagań
Fig. 9. Requirements list

Weryfikacja modelowa polega na porównaniu opisu modelu z listą wymagań. Jeżeli którekolwiek wymaganie nie jest spełnione, generowany jest odpowiedni kontrprzykład. Przykładowe wymaganie, które nie może być spełnione dla omawianego procesu sterowania, wraz z wygenerowanym kontrprzykładem przedstawione jest na Rys. 10. Badana jest właściwość określająca, że zawsze, jeżeli pierwszy zbiornik będzie pusty, to zawór wlewający do niego ciecz będzie otwarty. Z analizy kontrprzykładu wynika, że już w momencie inicjalizacji systemu wymagania te nie są spełnione.

```
-- specification AG (tank1 = empty -> v1 = open)  is
false
-- as demonstrated by the following execution se-
quence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  tank1 = empty
  tank2 = empty
  v1 = closed
  v2 = closed
  w1 = closed
  w2 = closed
  m = false
```

Rys. 10. Kontrprzykład
Fig. 10. Counterexample

Weryfikacja pozwala na wykrycie błędów zarówno w specyfikacji wymagań, jak również w modelu systemu. Dlatego też na podstawie wygenerowanego kontrprzykładu użytkownik musi podjąć decyzję, czy należy zmienić opis systemu, czy może listę właściwości.

4. Podsumowanie

W artykule zarysowano sposób i podano przykład transformacji diagramów aktywności języka UML do sieci Petriego.

Język UML daje możliwość sprawnej specyfikacji procesu sterowania, zwłaszcza przez osoby nie w pełni zapoznane z technikami informatycznymi.

Sieci Petriego dysponują natomiast sprawdzonym i rozbudowanym aparatem matematycznym oraz dużą ilością narzędzi dedykowanych dla tej technologii spełniających pomocnicze zadania w fazie projektowania sterowników logicznych. Do tego typu narzędzi zaliczane są omawiane w artykule narzędzia i techniki weryfikacji modelowej.

Technika weryfikacji modelowej pozwala na wykrycie błędów w specyfikacji procesu sterowania przedstawionego za pomocą sieci Petriego, czy też algorytmicznych maszyn stanów [21].

Zastosowanie języka UML, transformacji diagramu do sieci Petriego oraz docelowo weryfikacji modelowej podniesie jakość projektu i usprawni proces specyfikacji behawioralnej sterownika logicznego. Omawiany przykład obrazuje cały proces od specyfikacji procesu w postaci diagramu aktywności języka UML po wynikowy raport weryfikacji modelowej przeprowadzonej przy pomocy logiki temporalnej rozgałęzionej CTL oraz narzędzia NuSMV.

Dalsze badania obejmować będą realizację mechanizmów pozwalających na odwzorowywanie diagramów aktywności języka UML z zastosowaniem dekompozycji oraz specyfikacji zawierających opis zachowania w sytuacjach wyjątkowych. W ten sposób zdefiniowane zasady wykorzystane zostaną przy tworzeniu aplikacji do automatycznej transformacji obu typów specyfikacji. Usprawni to pracę projektantów poprzez możliwość sprawnej weryfikacji większej grupy projektów urządzeń osadzonych. Korzystając z tego będzie możliwe szersze zastosowanie istniejących i sprawdzonych mechanizmów weryfikacji modelowej, która jest w stanie w znaczny sposób poprawić jakość projektowanych urządzeń.

5. Literatura

[1] Graessle P., Baumann H., Baumann P.: UML 2.0 w akcji. Przewodnik oparty na projektach, Helion 2006, str. 17-38.

[2] Miles R., Hamilton K.: UML 2.0. Wprowadzenie, Helion 2007.

[3] Wrycza S., Marcinkowski B., Wyrzykowski K.: UML 2.0 w modelowaniu systemów informatycznych, Helion 2005.

[4] Oficjalna strona internetowa konsorcjum Object Management Group, http://www.omg.org

[5] Adamski M. A., Karatkevich A., Węgrzyn M. (ed.): Design of embedded control systems, Springer Science+Business Media, Inc., 2005.

[6] Yen-Liang C., Sammy C., Chyun-Chyi C., Irene C.: Workflow Process Definition and Their Applications in e-Commerce, IEEE 2000, str. 193 – 200.

[7] Eshuis R., Wieringa R.: A Comparison of Petri Net and Activity Diagram Variants, Proc. of 2nd Int. Coll. on Petri Net Technologies for Modelling Communication Based Systems 2001, str. 93 – 104.

[8] Schattkowsky T.: UML 2.0 – Overview and Perspectives in SoC Design, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05).

[9] David R., Alla H.: Petri Nets & Grafcet. Tools for modeling discrete event systems, Prentice Hall 1992.

[10] Grobelny M.: Transformacja diagramów aktywności UML 2.0 do sieci Petriego w systemach sterowania binarnego, Pomiary, Automatyka, Kontrola, 2009, nr 7, str. 498 – 500.

[11] Yao S., Shatz S.M.: Consistency Checking of UML Dynamic Model Based on Petri Net Techniques, Proceedings of the 15th International Conference on Computing, IEEE 2006.

[12] Staines T. S.: Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets, 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008, str. 191 200.

[13] Tričković I.: Formalizing activity diagram of UML by Petri nets, Novi Sad J. Math, Vol. 30, No. 3, 2000, pp. 161 171.

[14] Clarke E. M., Burch J. R., Grumberg O., Long D. E., McMillan K. L.: Automatic verification of sequential circuit designs, Phil. Trans. R. Soc. Lond. A (1992) 339, str. 105 – 120.

[15] Clarke E. M., Emerson E. A., Sistla A. P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986, str. 244 – 263.

[16] Emerson E. A.: The Beginning of Model Checking: A Personal Perspective, Lecture Notes in Computer Science, 25 Years of Model Checking: History, Achievements, Perspectives, 2008, str. 27 – 45.

[17] Rice M. V., Vardi M. Y.: Branching vs. Linear Time: Final Showdown, Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TA-CAS 2001, LNCS Volume 2031, Springer-Verlag 2001, str. 1 – 22.

[18] Clarke E. M., Wing J. M. et al.: Formal methods: State of the Art and Future Directions, ACM Computing Surveys, Vol. 28, No. 4, 1996.

[19] Crane M. L., Dingel J.: Towards a UML virtual machine: implementing an interpreter for UML 2 actions and activities, Proceedings of the 2008 conference of the center for advanced studies on collaborative research, IEEE 2008.

[20] Cavada R. et al. : NuSMV 2.4 User Manual, http://nusmv.irst.itc.it, 2005.

[21] Grobelna I.: Formalna analiza interpretowanych algorytmicznych maszyn stanów ASM z wykorzystaniem narzędzia model checker, Metody Informatyki Stosowanej nr 3/2008, Tom 16, str. 107 – 124.

otrzymano / received: 13.05.2010
przyjęto do druku / accepted: 01.09.2010 *artykuł recenzowany*

INFORMACJE

Informacja redakcji dotycząca artykułów współautorskich

W miesięczniku PAK od numeru 06/2010 w nagłówkach artykułów współautorskich wskazywany jest autor korespondujący (Corresponding Author), tj. ten z którym redakcja prowadzi wszelkie uzgodnienia na etapie przygotowania artykułu do publikacji. Jego nazwisko jest wyróżnione drukiem pogrubionym. Takie oznaczenie nie odnosi się do faktycznego udziału współautora w opracowaniu artykułu. Ponadto w nagłówku artykułu podawane są adresy korespondencyjne wszystkich współautorów.

Wprowadzona procedura wynika z międzynarodowych standardów wydawniczych.

Redakcja