Iwona GROBELNA, Michał GROBELNY
UNIWERSYTET ZIELONOGÓRSKI,
Podgórna 50, 65-246 Zielona Góra

Weryfikacja modelowa hierarchicznej specyfikacji sterownika logicznego

Dr inż. Iwona GROBELNA

Absolwentka Wydziału Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego oraz Fachhochschule Giessen-Friedberg (Niemcy). Od marca 2008 roku zatrudniona na stanowisku asystenta w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zainteresowania naukowe obejmują metody weryfikacji specyfikacji systemów wbudowanych.

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

Mgr inż. Michał GROBELNY*

Absolwent Wydziału Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego oraz Fachhochschule Giessen-Friedberg (Niemcy). Od października 2011 roku zatrudniony na stanowisku asystenta w Katedrze Mediów i Technologii Informacyjnych Uniwersytetu Zielonogórskiego. Zainteresowania naukowe obejmują metody specyfikacji osadzonych systemów sterowania.

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

Streszczenie

Specyfikacja zachowania projektowanego urządzenia powinna uwzględniać wszystkie elementy behawioralne. Z uwagi na złożoność projektowanych systemów szczególnie istotną rolę odgrywa możliwość dekompozycji. Z wykorzystaniem hierarchii można podzielić specyfikację na logiczne elementy połączone ze sobą na diagramach wyższego poziomu. W artykule przedstawiono zagadnienia związane z formalną weryfikacją hierarchicznych specyfikacji sterownika logicznego wyrażonych za pomocą interpretowanych sieci Petriego oraz diagramów aktywności języka UML.

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

Model checking of hierarchical logic controller specification

Abstract

Specification of a designed logic controller should include all behavioral aspects. By complex systems design decomposition is especially valuable. Specification can be divided into parts using hierarchy. Logical elements are joined together at higher-level diagrams. The paper focuses on formal verification [1] of logic controller hierarchical specification by means of UML activity diagrams and interpreted Petri nets. Although hierarchy itself is presented in the considered specification techniques in different ways (complex activities by UML activity diagrams and macroplaces/ macrotransitions by Petri nets), it is possible to use both techniques together in one project and to transform anytime one diagram into the another [5, 9, 10] (example in Figs. 1 and 2). In the transformation process, UML activity diagram actions correspond to Petri net transitions [7, 8]. Model checking [2, 3] of hierarchical specification can be performed step by step, e.g. by means of the NuSMV tool [11]. Rule-based specification (based on a Petri net) can be checked against behavioral properties [12, 13] expressed by temporal logic formulas [4]. Macroplaces can be verified separately (Fig. 3 considering local properties) and/or concurrently (Fig. 4, Fig. 5 considering mutual correlation and global properties). Next, the whole Petri net with macroplaces can be checked (Fig. 6). Sometimes it is convenient to verify a complete net (not hierarchical), like in [14]. Formal verification of specification can significantly increase its quality, and the support for hierarchy simplifies complex systems verification.

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

1. Wprowadzenie

Specyfikacja zachowania projektowanego urządzenia powinna zawierać wszystkie elementy behawioralne. Z uwagi na złożoność projektowanych systemów szczególnie istotną rolę odgrywa możliwość dekompozycji specyfikacji oraz implementacji. Z wykorzystaniem hierarchii można podzielić specyfikację na logiczne elementy połączone ze sobą na diagramach wyższego poziomu. Zarówno diagramy aktywności języka UML, jak i interpretowane sieci Petriego wspierają hierarchię. Wykorzystując rekonfigurowalne sterowniki logiczne i układy FPGA jako docelową platformę sprzętową, możliwa jest implementacja poszczególnych elementów składowych w różnych modułach. Umożliwi to późniejszą ewentualną wymianę części programu. Formalna weryfikacja specyfikacji [1] z wykorzystaniem techniki weryfikacji modelowej [2, 3] pozwala z kolei na sprawdzenie, czy sporządzona specyfikacja spełnia stawiane jej wymagania wyrażone za pomocą formuł logiki temporalnej [4].

W artykule przedstawiono sposób weryfikacji modelowej hierarchicznej specyfikacji przedstawionej w postaci interpretowanej sieci Petriego. Sieć ta może być także elementem pośrednim, otrzymanym na podstawie diagramu aktywności UML [5].

Artykuł podzielony jest następująco. Rozdział 2 przedstawia zagadnienia związane z formalną hierarchiczną specyfikacją sterownika logicznego. Rozdział 3 wprowadza nowatorską metodę weryfikacji modelowej hierarchicznej specyfikacji. Artykuł kończy się podsumowaniem oraz wnioskami.

2. Hierarchiczna specyfikacja sterownika logicznego

Specyfikacja zachowania urządzenia jest niezwykle istotna. Ważne jest, aby owa specyfikacja była zrozumiała i czytelna zarówno dla zleceniodawcy jak i dla projektanta i programistów. Specyfikacja musi przedstawiać wszystkie elementy behawioralne projektowanego systemu. Nawet najprostsze urządzenia w obecnych czasach realizują dziesiątki operacji. Każda taka operacja niejednokrotnie składa się z kilku lub kilkunastu akcji. Ważną rolę odgrywa tutaj możliwość podzielenia specyfikacji na mniejsze moduły. Z pomocą idą tutaj techniki hierarchicznej specyfikacji. Istnieją jednak pewne rozbieżności w przedstawianiu hierarchii w diagramach aktywności języka UML (w wersji 2.x) i sieciach Petriego, omówione w kolejnych podrozdziałach.

2.1. Hierarchia w diagramach aktywności

W diagramach aktywności języka UML 2.x hierarchia realizowana jest przez aktywności złożone. Specyfikacja systemu niejednokrotnie przedstawia skomplikowane czynności reprezentujące złożone procesy. Mogą one zostać zobrazowane w postaci jednej czynności [6], na którą składa się wiele czynności niższego poziomu lub wiele akcji (także czynności i akcji jednocześnie).

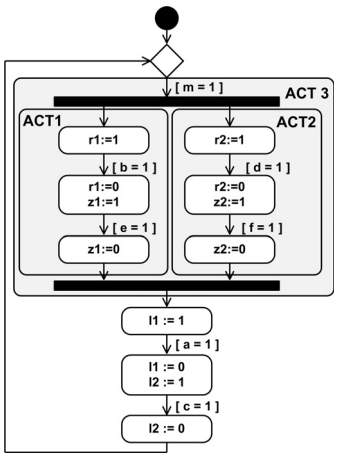
2.2. Hierarchia w sieciach Petriego

Hierarchiczne interpretowane sieci Petriego przedstawiają procesy sterowania obejmujące pewne wydzielone podprocesy. Z semantycznego punktu widzenia, są one jedynie inną formą graficznej reprezentacji, która jednakże rządzi się tymi samymi prawami co „zwykłe", niehierarchiczne sieci.

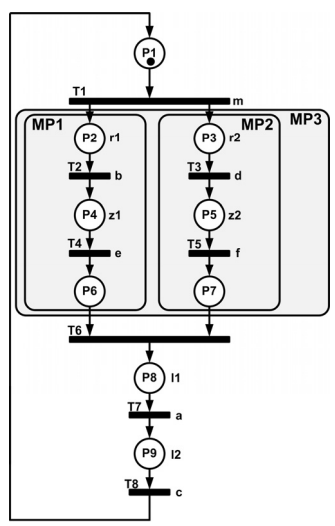
Możliwy jest dwoisty sposób reprezentacji struktur hierarchicznych w sieciach Petriego – przez makromiejsca i makrotranzycje. Oba sposoby różnią się nieznacznie pomiędzy sobą.

2.3. Transformacja pomiędzy wybranymi formami specyfikacji

Wykorzystanie transformacji diagramu aktywności UML do sieci Petriego sterowania stwarza możliwość weryfikacji modelowej wynikowej specyfikacji. Same sieci Petriego posiadają ponadto duże wsparcie ze strony formalnych technik weryfikacji i analizy. Przyjętą metodą transformacji diagramów aktywności UML 2.x do sieci Petriego sterowania jest ta polegająca na przypisaniu akcjom diagramu aktywności tranzycji sieci Petriego [7, 8]. I tak, przykładowa hierarchiczna interpretowana sieć Petriego z rys. 2 wynika z diagramu aktywności z rys. 1 [5].



Rys. 1. Przykładowy hierarchiczny diagram aktywności UML
Fig. 1. A sample UML hierarchical activity diagram



Rys. 2. Przykładowa hierarchiczna interpretowana sieć Petriego
Fig. 2. A sample hierarchical interpreted Petri net

Zastosowanie proponowanej metody transformacji pomiędzy wybranymi technikami specyfikacji umożliwia zarówno dogodne wykorzystanie ich obu w procesie projektowania sterownika logicznego [9], jak również sprawia, że metody weryfikacji interpretowanych sieci Petriego stają się osiągalne dla diagramów aktywności języka UML [10].

3. Weryfikacja modelowa hierarchicznej specyfikacji

Technika weryfikacji modelowej [2, 3] pozwala na automatyczną weryfikację behawioralnej specyfikacji systemu. Stosowana jest do weryfikacji systemów związanych ze sprzętem i oprogramowaniem. Przeprowadzana jest automatycznie przez narzędzia wnioskujące, przykładowo NuSMV [11]. Dostępne są mechanizmy pozwalające na sprawdzenie regułowej specyfikacji zachowania sterownika logicznego [12, 13]. Abstrakcyjna, regułowa specyfikacja tworzona jest na podstawie interpretowanej sieci Petriego sterowania. Taki behawioralny model nadaje się zarówno do formalnej weryfikacji, jak i do syntezy logicznej (przeprowadzanej w formie szybkiego prototypowania). Otrzymana implementacja jest w pełni zgodna ze zweryfikowaną specyfikacją.

3.1. Weryfikacja etapami

Weryfikację modelową hierarchicznej specyfikacji można wykonać w kilku etapach. Hierarchiczną interpretowaną sieć Petriego można podzielić na kilka części, wydzielając z niej odpowiednie makromiejsca. Każde makromiejsce zawierające przynajmniej kilka miejsc i tranzycji, może być zapisane w postaci regułowej, a następnie oddzielnie weryfikowane.

Sporządzony na podstawie makromiejsca model podsystemu będzie opisywał proste operacje w nim zachodzące (*MP1* z rys. 2 jako krok pierwszy i *MP2* jako krok drugi). Weryfikacja modelowa takiego makromiejsca potwierdzi, czy spełnia ono stawiane mu wymagania. Makromiejsca, które zawierają niewiele miejsc i tranzycji i nie wymagają osobnej weryfikacji mogą zostać rozwinięte do pełnych rozmiarów i zweryfikowane razem z pozostałą częścią sieci. Po pomyślnym zweryfikowaniu wybranych makromiejsc możliwe jest przeprowadzenie weryfikacji na całej sieci Petriego (z makromiejscem *MP3*). Zweryfikowane wcześniej makromiejsce *MP3* (zawierające makromiejsca *MP1* i *MP2*), w miarę możliwości, może być traktowane jako stan złożony bez wnikania w jego wewnętrzną strukturę. Możliwe jest wtedy sprawdzenie, czy cały zaprojektowany system spełnia stawiane mu wymagania behawioralne.

3.2. Weryfikacja makromiejsca

Opis modelu makromiejsca podlegający weryfikacji zawiera definicje występujących w nim miejsc, sygnałów wejściowych oraz wyjściowych sterownika logicznego. Przedstawiane są zmiany stanów wewnętrznych. Początek sieci posiada aktywne znakowanie (miejsce *P2* i sygnał *r1* są aktywne w momencie uruchomienia makromiejsca *MP1* odpowiadającego aktywności złożonej *ACT1*). Podsieć kończy się aktywnym znakowaniem ostatniego miejsca (miejsce *P6* makromiejsca *MP1*).

Po sporządzeniu opisu modelu danego makromiejsca należy przygotować listę stawianych mu wymagań. Są one definiowane z wykorzystaniem logiki temporalnej, np. liniowej logiki LTL. Przykład na rys. 3 określa, że nie jest dopuszczalna sytuacja jednoczesnej aktywności sygnałów wyjściowych *r1* i *z1*. W rzeczywistym środowisku można to zinterpretować jako jednoczesny ruch pojazdu w prawą stronę i otwarcie zsypu napełniającego pojemnik.

```
LTLSPEC !G (r1 & z1);
```

Rys. 3. Wymaganie dla makromiejsca *MP1*
Fig. 3. Requirement for macroplace *MP1*

3.3. Weryfikacja współbieżnych makromiejsc

Dodatkowo, podczas weryfikacji modelowej możliwe jest jednoczesne uruchomienie wielu procesów, czyli jednoczesne rozpatrywanie wielu makromiejsc. Definiując w modelu głównym (zawierającym kompletną sieć) osobny proces dla danego makromiejsca można sprawdzić jak zachowują się względem siebie procesy określone przez makromiejsca. Do definicji zmiennych należy wtedy dodać rozpatrywane procesy (jak na rys. 4, *mp1* i *mp2* oznaczają odpowiednio poszczególne zdefiniowane i zweryfikowane wcześniej moduły odpowiadające makromiejscom).

```
macro1: process mp1();
macro2: process mp2();
```

Rys. 4. Definicja procesów w opisie modelu
Fig. 4. Process definition in model description

Procesy będą działały w miarę naprzemiennie, tzn. w danej chwili może być uruchomiony tylko jeden proces, ale oba są sprawiedliwie wykonywane – żaden proces nie jest ani zbyt często pomijany, ani faworyzowany. Przykładowe wymaganie, zdefiniowane tym razem z wykorzystaniem rozgałęzionej logiki temporalnej CTL przedstawiono na rys. 5. Sprawdza ono, czy jest możliwe (dowolny taki stan), aby oba pojazdy poruszały się jednocześnie w prawą stronę.

```
CTLSPEC EF(macro1.r1 & macro2.r2);
```

Rys. 5. Wymaganie dla współbieżnych procesów
Fig. 5. Requirement for concurrent processes

3.4. Weryfikacja całej sieci

Po pomyślnej weryfikacji poszczególnych makromiejsc można sprawdzić kompletną sieć. Dla sieci z rys. 2 uwzględniane jest makromiejsce *MP3* zawierające dwa, zweryfikowane wcześniej, makromiejsca *MP1* i *MP2*. Miejsca wewnątrz tych makromiejsc, jak również sygnały wejściowe i wyjściowe nie są więcej rozpatrywane. Dla sieci z rys. 2 uwzględnione będą zatem miejsca *P1*, *MP3*, *P8* i *P9*. Przykładowe wymagania zdefiniowane w logice LTL zamieszczono na rys. 6. Dotyczą one struktury sieci (sprawdzane jest odpowiednie następstwo stanów) oraz zachowania projektowanego sterownika logicznego. Właściwość trzecia określa, że zawsze gdy pierwszy pojazd jedzie w lewą stronę i osiągnie swój punkt docelowy *a*, to w końcu drugi pojazd również rozpocznie swój przejazd w lewo. Ostatnia właściwość definiuje, że zawsze gdy drugi pojazd porusza się w lewą stronę, to w końcu jego przejazd się zakończy (odpowiedni napęd lub silnik zostanie wyłączony). Wymaganie to nie jest spełnione w rozpatrywanym opisie modelu. Generowany jest wtedy odpowiedni kontrprzykład, który pozwala zlokalizować miejsce błędu. Możliwe jest bowiem zapętlenie systemu. Sygnał wejściowy sterownika logicznego determinuje zakończenie ruchu, jest on jednak w pełni niezależny i w rzeczywistym środowisku pochodzi z zewnątrz. Może się zatem zdarzyć, że odpowiedni czujnik ulegnie awarii lub też pojazd zostanie zablokowany na swojej drodze. Kontrprzykład jasno wskazuje potencjalne niebezpieczeństwa.

```
LTLSPEC G (p1 & m -> X mp3);          -- 1
LTLSPEC G (mp3 -> X p8);              -- 2
LTLSPEC G (l1 & a -> F l2);           -- 3
LTLSPEC G (l2 -> F !l2);              -- 4
```

Rys. 6. Wymagania dotyczącej całej sieci
Fig. 6. Requirements for the whole net

3.5. Weryfikacja pełnej rozwiniętej sieci

Występują jednak takie sytuacje, gdy weryfikacja makromiejsc nie wnosi nic nowego do projektu (proste makromiejsca), a sieć Petriego z makromiejscami nie daje możliwości sprawdzenia wszystkich właściwości. Wtedy rozwiązaniem wydaje się być rozwinięcie wszystkich makromiejsc i weryfikacja kompletnej sieci Petriego (przykład weryfikacji sieci z wieloma procesami współbieżnymi o 20 miejscach lokalnych, 13 sygnałach wejściowych i 12 wyjściowych przedstawiono w pracy [14]). Niestety, w przypadku złożonych systemów o bardzo dużej liczbie stanów zadanie to staje się trudne i bardzo czasochłonne.

4. Podsumowanie i wnioski

W artykule zaproponowano metody weryfikacji modelowej hierarchicznych specyfikacji sterownika logicznego. Specyfikacja zapisana jest formalnie w postaci interpretowanej sieci Petriego lub diagramu aktywności języka UML. Możliwa jest oddzielna weryfikacja pojedynczych makromiejsc (sprawdzająca lokalne właściwości), jak również jednoczesna wielu współbieżnych makromiejsc (sprawdzająca globalne właściwości). Następnie, rozpatrywana jest cała sieć Petriego zawierająca makromiejsca, bez wnikania w ich wewnętrzną strukturę.

Zaletą proponowanego podejścia jest możliwość przeprowadzenia częściowej formalnej weryfikacji i wprowadzenia poprawek lub usprawnień tylko do wybranego fragmentu sieci, implementowanego następnie np. jako jeden moduł w układzie FPGA. Jednocześnie, nadal możliwa pozostaje weryfikacja ogólnej sieci na wyższym poziomie abstrakcji. Wadą jest niestety ścisła zależność modelu weryfikowalnego ze strukturą hierarchiczną sieci, a także trudność definicji wymagań na różnych poziomach szczegółowości (pewne wymagania dotyczą tylko wybranych makromiejsc i powinny być zdefiniowane wewnątrz nich).

* *Autor jest stypendystą w ramach Poddziałania 8.2.2 „Regionalne Strategie Innowacji", Działania 8.2 „Transfer wiedzy", Priorytetu VIII „Regionalne Kadry Gospodarki" Programu Operacyjnego Kapitał Ludzki współfinansowanego ze środków Europejskiego Funduszu Społecznego Unii Europejskiej i z budżetu państwa.*

5. Literatura

[1] Kropf T.: Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems. 1st edn, Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.
[2] Emerson E. A.: The Beginning of Model Checking: A Personal Perspective. w O. Grumberg, H. Veith (ed.), 25 Years of Model Checking, Vol. 5000 of Lecture Notes in Computer Science, Berlin, Heidelberg: Springer-Verlag, s. 27-45, 2008.
[3] Clarke E. M., Grumberg O., Peled D. A.: Model Checking. The MIT Press, 1999.
[4] Ben-Ari M.: Logika matematyczna w informatyce. Klasyka informatyki, Wydawnictwa Naukowo-Techniczne, 2005.
[5] Grobelny M., Grobelna I., Adamski M.: Hierarchical UML activity diagrams into control interpreted petri nets transformation. Mixed Design of Integrated Circuits and Systems - 19th international conference, Warszawa, Polska, s. 523-526, 2012.
[6] Wrycza S., Marcinkowski B., Wyrzykowski K.: UML 2.0 w modelowaniu systemów informatycznych, Wydawnictwo Helion, 2005.
[7] Grobelna I., Grobelny M., Adamski M.: Petri nets and activity diagrams in logic controller specification - transformation and verification. Mixed Design of Integrated Circuits and Systems – 17th International Conference, Wrocław, Polska, s. 607-612, 2010.
[8] Grobelny M., Grobelna I.: Diagramy aktywności języka UML i sieci Petriego w systemach sterowania binarnego - od transformacji do weryfikacji. Pomiary, Automatyka, Kontrola, Vol. 56, nr 10, s. 1154-1158, 2010.
[9] Grobelny M., Grobelna I., Adamski M.: Hardware behavioural modelling, verification and synthesis with UML 2.x activity diagrams. Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDeS), Brno, Czechy, s. 109-114, 2012.
[10] Grobelny M., Grobelna I.: Diagramy aktywności UML w projektowaniu rekonfigurowalnych sterowników logicznych. Pomiary, Automatyka, Kontrola, Vol. 58, nr 7, s. 596-598, 2012.
[11] Strona domowa narzędzia weryfikacji modelowej NuSMV: http://nusmv.fbk.eu/
[12] Grobelna I.: Weryfikacja modelowa specyfikacji sterowników logicznych. Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2012.
[13] Grobelna I.: Formal verification of embedded logic controller specification with computer deduction in temporal logic. Przegląd Elektrotechniczny, nr 12a, s. 40-43, 2011.
[14] Grobelna I.: Control interpreted Petri Nets - model checking and synthesis. Petri Nets - manufacturing and computetr science, P. Pawlewski (ed.), InTech, s. 177-192, 2012.

otrzymano / received: 20.05.2013
przyjęto do druku / accepted: 03.07.2013 *artykuł recenzowany / revised paper*