AUTOMATYKA • 2008 • Tom 12 • Zeszyt 2

Konrad Kułakowski*, Marek Kostrzewa*

Modelowanie systemów czasu rzeczywistego w UML

1. Wprowadzenie

Pierwsze systemy czasu rzeczywistego były rozwiązaniami głównie sprzętowymi. Z biegiem czasu rola oprogramowania rosła, by w końcu stać się dominującym składnikiem takich systemów, decydującym o ich zastosowaniu i funkcjonalności. Przykładem może być szeroko rozumiana „elektronika użytkowa", wszechobecna pod postacią telefonów komórkowych, kamer, aparatów cyfrowych, urządzeń wyposażonych we wszelkiego rodzaju programatory i układy sterujące. Wszystkie te urządzenia podejmując działanie na zlecenie użytkownika, zobowiązane są dostarczyć mu wynik w konkretnym skończonym czasie. W większości z nich oprócz warstwy sprzętowej, obecne jest działające w czasie rzeczywistym oprogramowanie decydujące często w znacznym stopniu o możliwościach danego urządzenia. Implementacja części funkcjonalności systemu na poziomie oprogramowania przynosi wiele korzyści: pozwala na zmniejszenie kosztów produkcji produkcji samego urządzenia, szybkie reagowanie na zapotrzebowanie rynku poprzez implementacje nowych cech produktu oraz na łatwość usuwania usterek dostrzeżonych po wprowadzeniu urządzenia do sprzedaży. Wraz ze wzrostem znaczenia oprogramowania w konstrukcjach czasu rzeczywistego, wzrosło znaczenie metod inżynierii oprogramowania zajmujących się jego wytwarzaniem. Najnowsze trendy w tym zakresie koncentrują się na obiektowych metodach modelowania, wytwarzania i analizy systemów czasu rzeczywistego. Na szczególną uwagę zasługują konstrukcje oparte na zyskującej coraz większy rozgłos platformie *Real Time Java* [4]. Przykładem może być przedstawione w niniejszej pracy środowisko *RAT*.

1.1. Reaktywne systemy czasu rzeczywistego

Pojęcie *systemu reaktywnego* zostało wprowadzone przez Davida Harela oraz Amira Pnueliego [7]. Zwykle tym mianem określa się autonomicznie działający system, potrafiący komunikować się z otoczeniem. Niektóre ważniejsze cechy wymieniane w kontekście systemów reaktywnych to: ciągła interakcja z otoczeniem, obsługa synchronicznego oraz asynchronicznego wejścia/wyjścia, możliwość istnienia ograniczeń czasowych związanych

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

z reakcją na zachowanie otoczenia, potencjalnie duża ilość scenariuszy działania systemu wynikająca z różnorodnych zachowań otoczenia systemu, możliwość pracy ciągłej, współbieżność. Często systemy reaktywne są równocześnie systemami czasu rzeczywistego. Dzieje się tak wówczas, kiedy istnieją konkretne wymagania dotyczące prędkości reakcji takiego systemu na zdarzenia pochodzące z jego otoczenia. Zatem oprócz wymagań ogólnych dotyczących jakości budowanego oprogramowania, w przypadku reaktywnych systemów czasu rzeczywistego dochodzą wymagania szczegółowe, związane zarówno ze specyfiką systemów reaktywnych jak i systemów czasu rzeczywistego. Sprostanie tym wszystkim wymaganiom wymusza na projektantach systemu poszukiwanie metod i narzędzi wspierających wytwarzanie tego typu oprogramowania.

1.2. Metody formalne

Jednym ze sposobów na zapewnienie wysokiej jakości oprogramowania czasu rzeczywistego jest wykorzystanie modeli zbudowanych w oparciu o metody formalne. Do najbardziej znanych formalizmów używanych do modelowania systemów czasu rzeczywistego należy zaliczyć rozszerzenia czasowe różnego rodzaju algebr procesów.

Tabela 1
Rozszerzenia czasowe głównych algebr procesów

Algebra procesów	Wybrane rozszerzenie czasowe
CCS (*Calculus of Communicating Systems*)	SCCS, Timed CCS, RT CCS
CSP (*Calculus of Sequential Processes*)	Timed CSP, RT CSP
ACP (*Algebra of Communcating Processes*)	Timed ACP, RT ACP
LOTOS (*Language Of Temporal Ordering Specification*)	RT LOTOS, ET LOTOS

Praktycznie każda z głównych algebr procesów wykształciła swoje własne, czasowe odmiany (tab. 1). Rozszerzenia czasowe algebr procesów pozwalają na modelowanie czasu zarówno lokalnie (opóźnienie czasowe związane jest z lokalnym przejściem procesu do kolejnego stanu) jak i globalnie (istnieje globalny zegar regulujący upływ czasu dla całego modelowanego systemu). Przykładem pierwszego podejścia mogą być algebry Timed CCS, Enhanced Timed LOTOS, przykładem drugiego Synchronous CCS, gdzie każdy z procesów jest zobowiązany do wykonania jakiejś akcji w każdym takcie globalnego zegara [5]. W tej klasie formalizmów wartości oznaczeń czasowych mogą pochodzić ze zbiorów dyskretnych lub ciągłych [9]. W pierwszym przypadku mówi się często o **rozszerzeniach czasowych** (*timed*) algebr procesów, w drugim o **rozszerzeniach czasu rzeczywistego** (*real time*).

Kolejną klasą metod formalnych wykorzystywaną w modelowaniu systemów czasu rzeczywistego są **sieci Petriego**. Również w tym przypadku istnieje kilka odmian czasowych tego formalizmu. Najbardziej popularne to **proste sieci czasowe** oraz **przedziałowe**

sieci czasowe [17]. Rozwinięciem prostych sieci Petriego są **kolorowane sieci Petriego**, które również posiadają własne rozszerzenia czasowe. Wyróżnić tu można **czasowe sieci kolorowane Petriego** oraz **kolorowane sieci Petriego czasu rzeczywistego** (*RTCP-sieci*) [16, 17]. Modelowanie czasu w obrębie tego formalizmu związane jest zwykle z pojęciem zegara globalnego synchronizującego upływ czasu dla całego systemu.

Popularnymi metodami formalnymi specyfikacji i badania własności systemów reaktywnych są **logiki temporalne** [11]. Ich użycie często jest połączone ze stosowaniem innych metod, na przykład algebr procesów. W szczególności niektóre algebry procesów wykształciły własne odmiany logik temporalnych uzupełniające pierwotne formalizmy. Stąd tworząc model systemu np. w algebrze procesów CCS, projektant może go rozszerzyć o wymagania zapisane w logice HML (*Hennesy-Milner Logic*) [15] itp.

Formalny model systemu, po stworzeniu, zostaje poddany weryfikacji w odpowiednim dla analizy danej klasy modeli formalnych programie. W wyniku analizy twórcy oprogramowania otrzymują informacje o własnościach powstałego modelu na podstawie których podejmują dalsze decyzje projektowe.

Trudno przecenić zalety stosowania metod formalnych do modelowania systemów informatycznych. Pozwalają one na; dokładne zbadanie poprawności działania takich systemów, analizę własności czasowych, symulację zachowania na stworzonym uprzednio modelu. Pewnym niedostatkiem może być fakt, iż dla niewielu z nich stworzono narzędzia wspierające automatyzację procesu wytwarzania oprogramowania na etapie przejścia od modelu formalnego do rzeczywistego, działającego systemu informatycznego. W praktyce częstym problemem jest też mała liczba osób posiadających wiedzę pozwalającą na modelowanie z wykorzystaniem metod formalnych. W konsekwencji proces projektowy wykorzystujący metody formalne jest drogi, a przez to stosowany tylko w projektach, w których poprawność wykonania systemu informatycznego jest absolutnie krytyczna.

2. UML w modelowaniu systemów czasu rzeczywistego

2.1. UML

Język UML jest szeroko uznanym standardem specyfikacji różnego rodzaju systemów o skomplikowanym zachowaniu i strukturze. Do jego powstania przyczynił się szereg wcześniejszych prac definiujących strukturalne metody specyfikacji systemów czasu rzeczywistego, z których do najważniejszych trzeba zaliczyć metodę **Warda-Mellora** i metodę **Hatleya-Pirbhaiego**, szczegółowo opisane w literaturze [3, 25]. Pierwsza wersja języka końcowy kształt otrzymała w wyniku prac trzech głównych metodologów zajmujących się metodami obiektowymi, tj. **G. Boocha**, **J. Rumbaugha** i **I. Jacobsona**. Pierwotna składnia języka UML to w dużej mierze kompilacja metod modelowania zaproponowanych przez jego twórców.

Choć dostępne wówczas diagramy pozwalały na projektowanie znacznej klasy systemów obiektowych, to niestety projektantom doskwierał brak możliwości zwiększenia pre-

cyzji tworzonych modeli. Stąd z czasem, język UML został uszczegółowiony i wzbogacony o dodatkowe diagramy. W efekcie umożliwia on lepsze precyzowanie pojęć w odniesieniu do konkretnej dziedziny zastosowań. Temu ostatniemu służy mechanizm profili rozszerzających podstawową semantykę języka.

2.2. UML czasu rzeczywistego

Rozszerzenie możliwości modelowania w UML jest możliwe poprzez użycie już istniejących bądź zdefiniowanie nowych profili języka. Głównym zadaniem profilu jest dostarczenie projektantowi gotowych pojęć z dziedziny problemu wraz z opisem, w jaki sposób działają, jak można je łączyć i w jaki sposób przebiega pomiędzy nimi interakcja. W ten sposób standardowym elementom języka UML zostaje nadane znaczenie specyficzne dla dziedziny problemu. Istnieją różne profile języka UML. Aktualną ich listę wspierana prze OMG (*Object Management Group*) można znaleźć na stronach OMG[1)].

Oficjalnym profilem zalecanym przez OMG, służącym do modelowania systemów czasu rzeczywistego jest *UML Profile for Schedulability, Performance and Time* [24]. Wspiera on następujące aspekty modelowania systemu:

- modelowanie zasobów systemu,
- modelowanie czasu,
- modelowanie współbieżności,
- modelowanie szeregowania procesów,
- modelowanie wydajności.

Ogólne wskazówki zawarte w dokumencie specyfikacji profilu zostały opatrzone przykładem modelu aplikacji w technologii Real-Time CORBA proponowanej przez OMG. W przygotowaniu jest kolejny profil standardowy UML (*UML Profile for Modeling and Analysis of Real-Time and Embedded Systems*)[2)] [23]. Profil ten oferuje projektantowi pomoc przy modelowaniu systemów czasu rzeczywistego, a także, co jest nowością, przy ich analizie. W przyszłości ma on zastąpić i uzupełnić dotychczasowy profil *UML Profile for Schedulability, Performance and Time*.

2.3. MDA

Opracowano wiele różnych metodyk sprzyjających budowie systemów informatycznych wysokiej jakości. W wielu z nich etap modelowania systemu odgrywa kluczową rolę. Spośród tych metod, warto zwrócić uwagę na podejście MDA (*Model Driven Architecture*) [10, 19]. Opracowane przez OMG, MDA jest propozycją pewnej metodologii i technologii wytwarzania oprogramowania, w której kluczową rolę odgrywa modelowanie systemu w języku UML. Cechą charakterystyczną tego podejścia jest odseparowanie aspektu funk-

---

1) *http://www.omg.org/technology/documents/profile_catalog.htm*

2) W momencie pisania tekstu dostępna jest specyfikacja w wersji beta 1.

cjonalnego od aspektu implementacyjnego. Taki podział pozwala na łatwiejsze zrozumienie istoty rozwiązywanego problemu, oczekiwań klientów, celowości istnienia tworzonego systemu. Umożliwia także położenie większej wagi na problemy architektoniczne tworzonego rozwiązania. Proces wytwarzania oprogramowania w metodzie MDA składa się z kilku etapów, w trakcie których tworzone są coraz bardziej szczegółowe modele całego systemu. Wyróżnia się trzy następujące poziomy abstrakcji modeli:

- CIM (*Computation Independent Model*), w którym przedstawione są wymagania systemu. Opisuje on otoczenie w którym system będzie użyty. Często ten model określa się jako model dziedziny bądź też model biznesowy konstruowanego systemu.
- PIM (*Platform Independent Model*), w którym przedstawiony jest model systemu pozbawiony aspektów implementacyjnych.
- PSM (*Platform Specific Model*), w którym opis systemu dodatkowo wzbogacony jest o aspekty implementacyjne, związane z konkretną, docelową platformą uruchomieniową.

Przejście pomiędzy poszczególnymi modelami powinno być możliwie automatyczne. Zastosowanie MDA podczas tworzenia oprogramowania może wpłynąć korzystnie na szereg czynników związanych z szeroko rozumianą jakością, między innymi na:

- zwiększenie przenośności (*portability*) pomiędzy różnymi platformami technicznymi,
- zwiększenie produktywności (*productivity*),
- istotne zmniejszenie kosztów poprzez łatwiejszą pielęgnację (*maintenance*),
- wzrost powtórnego wykorzystania stworzonych komponentów (*reuseability*),
- minimalizacje czasu potrzebnego do wprowadzenia produktu na rynek,
- zmniejszenie ryzyka związanego ze zmianami wymagań biznesowych oraz rozwoju technologii.

Zalecanym językiem modelowania w ramach podejścia MDA jest UML. Ponad to MDA wykorzystuje następujące standardy: MOF (*Meta Object Facility*) [22], XMI (*XML Metadata Interchange*) [21], CWM (*Common Warehouse MetaModel*) [20].

2.4. Executable UML

Executable UML, często skracany jako xUML lub xtUML (*executable and translatable UML*) jest rozwinięciem obiektowym metody Shlaera–Mellora. Jedną z głównych idei, która przyświecała twórcom xUML-a jest możliwość stworzenia takiego obiektowego modelu systemu, który można by bezpośrednio uruchamiać, pomijając w ten sposób żmudny i obarczony sporym ryzykiem proces ręcznej implementacji oprogramowania. W kontekście podejścia MDA, xUML jest techniką pozwalającą na stworzenie warstwy PIM a następnie automatyczną translację takiego modelu do postaci wykonywalnej PSM, (w kontekście xUML translacja do postaci zależnej od platformy jest nazywana kompilacją modelu). W celu umożliwienia tworzenia „uruchamialnego projektu" xUML wprowadza

do języka UML precyzyjną semantykę akcji [26]. Zaprojektowany system podejmuje działania uruchamiając akcje zapisane w jednym ze specjalnych języków (np. OAL – *Object Action Language*, ASL – *Action Specification Language*, SMALL – *Shlaer-Mellor Action Language*).

3. Istniejące rozwiązania

3.1. Podejście MDA

Istnieje wiele narzędzi i metod tworzenia uruchamialnych modeli systemów reaktywnych. Dzięki ich zastosowaniu użytkownicy i projektanci mogą w łatwy sposób sprawdzić czy model spełnia postawione wymagania, przeprowadzić symulację działania systemu, wygenerować działający program, a w przypadku konieczności wprowadzenia zmian, w prosty sposób nanieść je na powstający projekt. Grupą narzędzi wspomagających szybkie tworzenie systemów RT przy wykorzystaniu języka UML są systemy implementujące model MDA. Przykładem może być tu CoSMIC (*Component Synthesis using Model Integrated Computing*) [6]. Projekt ten jest zbiorem narzędzi do tworzenia systemów DRE (*Distributed Real-Time and Embedded*) opartych o technologię CORBA. Głównym jego zadaniem jest modelowanie i analiza wymagań QoS (*Quality of Service*) dla klasy systemów DRE. Kolejnym przykładem oprogramowania wspierającego tworzenie systemów czasu rzeczywistego jest FUJABA [28]. Narzędzie to wspiera projektowanie i weryfikację z wykorzystaniem UML/RT (*UML for Real Time*). W tym podejściu duży i skomplikowany model systemu jest dekomponowany do mniejszych, specyficznych dla danej dziedziny zastosowań, fragmentów (wzorców). Poprawność każdego z otrzymanych po dekompozycji elementów może być analizowana oddzielnie. Weryfikacja polega na transformacji maszyn stanu czasu rzeczywistego RTSC (*Real Time Statecharts*) do HTA (*Hierarhical Timed Automata*), które następnie są analizowane w środowisku formalnej weryfikacji modeli Uppaal [14]. Niestety z uwagi na konieczność korzystania z diagramów RTSC, które nie są standardowymi elementami języka UML, możliwości tworzenia modeli korzystających z proponowanej metody są ograniczone do środowiska FUJABA. Systemem wspierającym podejście MDA, generującym kod uruchamialny w RT Java jest HIDOORS (*High Integrity Distributed Object-Oriented Real-Time System*) [18]. Środowisko HIDOORS zapewnia zestaw narzędzi do tworzenia i weryfikacji modelu, a także specjalny profil HIDOORS UML dostarczający wygodnych abstrakcji projektowych.

3.2. *Reactive Appliance Toolkit*

RAT (*Reactive Appliance Toolkit*) powstał w 2006 roku jako narzędzie wczesnej fazy analizy wymagań dla systemów reaktywnych czasu rzeczywistego [12, 13]. W skład pakietu RAT wchodzą następujące komponenty: biblioteka RAT API, generator kodu, konsola zarządzająca. Pozwala on na tworzenie systemów czasu rzeczywistego z wykorzystaniem

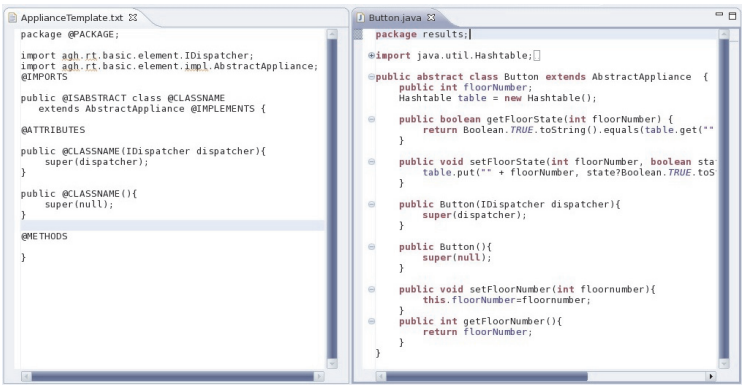
standardowego edytora UML wspierającego standard XMI. Koniecznym warunkiem uruchomienia modelu jest posiadanie maszyny wirtualnej Javy RT. Pakiet RAT zapewnia wsparcie dla modelowania w obrębie dwóch warstw MDA: PIM i PSM. Rolę PIM spełnia model systemu reaktywnego RT zapisany w języku UML, który następnie podlega automatycznej transformacji do kodu źródłowego w Javie. Powstała w ten sposób uruchamialna aplikacja odpowiada warstwie PSM w podejściu MDA.

Biblioteka RAT API

Biblioteka RAT API dostarcza zbioru ogólnych pojęć przydatnych podczas modelowania urządzeń reaktywnych. Podstawowym pojęciem jest *appliance*, określające moduł programowy odpowiedzialny za sterowanie dowolnym urządzeniem. Urządzenia mogą otrzymywać i wysyłać między sobą wiadomości. Ich działanie jest modelowane za pomocą maszyny stanów, która definiuje zachowanie danego urządzenia. Warunki czasowe definiowane w modelowanym systemie związane są z czasem wykonania przejść (ciągu przejść) w obrębie takiej maszyny stanów.

Generator kodu

Generator kodu dostępny w pakiecie RAT pełni rolę kompilatora modelu. Wejściem generatora kodu jest model UML zapisany w postaci XMI, wyjściem zbiór klas języka Java. Otrzymany w ten sposób kod aplikacji można skompilować i uruchomić na platformie Real Time Java. Generator działa w trybie wsadowym i przekształca poszczególne elementy modelu UML w oparciu o stworzony w tym celu zbiór szablonów (rys. 1).

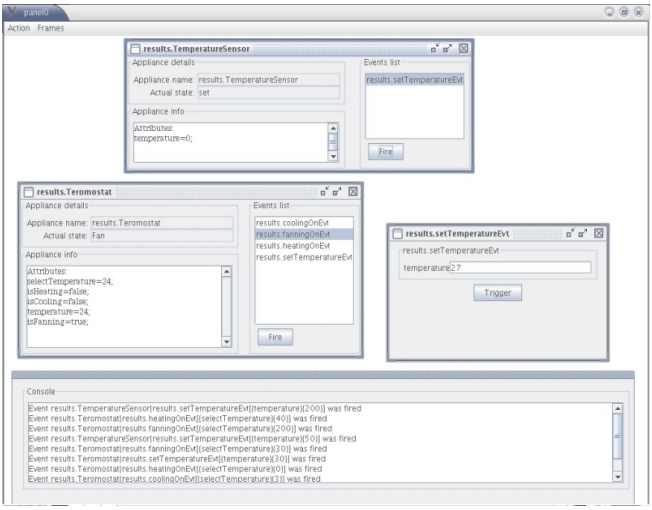


Rys. 1. Szablon Appliance i wygenerowana na jego podstawie klasa Button

Konsola zarządzająca

Możliwość symulacji modelu w ramach biblioteki RAT zapewniana jest przez konsolę zarządzającą (rys. 2). Konsola łączy się poprzez protokół TCP/IP ze skompilowanym i uruchomionym modelem, pobiera nazwy obiektów składających się na działający system,

a następnie umożliwia aktywne śledzenie przebiegu sterowania. W szczególności możliwy jest podgląd stanu obiektów, śledzenie przepływu komunikatów a także wysyłanie komunikatów do określonych obiektów systemu.



Rys. 2. Konsola zarządzająca systemem sterowania termostatem

4. Podsumowanie

Modelowanie systemów czasu rzeczywistego, początkowo przeprowadzane było za pomocą różnego rodzaju języków prototypowania, np. RT-ASLAN, ASTRAL [1, 2] oraz notacji graficznych: Warda-Mellora [25], Hatleya–Pirbhaiego [3] i LACATRE [8, 27]. Obecnie modelowanie takich systemów zaczyna być domeną metod obiektowych opartych na języku UML, w których prym wiodą rozwiązania implementujące podejście MDA. Pakiet RAT jest próbą stworzenia systemu umożliwiającego modelowanie, symulacje, a także generacje systemów reaktywnych czasu rzeczywistego działających w środowisku Real Time Java. Nie jest to jeszcze rozwiązanie kompletne. Na dalszym etapie prac przewidywane jest uzupełnienie tego pakietu o narzędzia umożliwiające formalną analizę powstałego modelu, w szczególności jego własności czasowych.

Literatura

[1] Auernheimer B., Kemmerer R.A., RT-ASLAN, *A specification language for real-time system*. IEEE Trans. Softw. Eng. (1986), 12.
[2] Coen-Porisini A., Ghezzi C., Kemmerer R.A., *Specification of realtime systems using astral*. IEEE Transactions on Software Engineering (1997), TRCS96-30.
[3] Hatley D.J. IP, *Strategies for real-time system specification*. Dorset House, 1987.

[4] Dibble P.C., *Real-time Java platform programming*. Prentice Hall PTR, 2002.
[5] Fencott C., *Formal methods for concurrency*. International Thomson Computer Press, 1995.
[6] Gokhale A.S., Schmidt D.C., Lu T., Natarajan B., Wang N., *Cosmic an MDA generative tool for distributed real-time and embedded applications*. In Middleware workshops, 2003.
[7] Harel D., Pnuelli A., *On the development of reactive systems*. In Logic and models of concurrent systems, 1985.
[8] Schwarz J.J., Skubich J.J., Aubry R., *Lacatre: the basis for a real time software engineering workshop*. In Pearl 91 workshop ueber realzeitsysteme 12. fachtagung des pearl-vereins e.v boppard 28./29. november 1991 proceedings, 1991.
[9] Baeten J.C.M., Middleburg C.A., *Process algebra with timing*. Springer, 2002.
[10] Kleppe A., Warmer J., Bast W., *MDA Explained, The model driven architecture: practice and promise*. Addison-Wesley, 2003.
[11] Klimek R., *Wprowadzenie do logiki temporalnej*. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, 1999.
[12] Kostrzewa M., Kułakowski K., *A practical approach to the modelling visualising and executing of reactive systems*. In Mixed design of integrated circuits and systems, 2006.
[13] Kułakowski K., Kostrzewa M., *Wspomaganie tworzenia oprogramowania systemów reaktywnych w real time java*. In Systemy czasu rzeczywsitego, 2006.
[14] Larsen K.G., Pettersson P., Yi W., *Uppaal in a nutshell*. Int. Journal on Software Tools for Technology Transfer () **1**, 134–152.
[15] Hennesy M.C.B., Milner R., *Algebraic laws for nondeterminism and concurrency*. Journal of the ACM (1985).
[16] Szpyrka M., *Analysis of rtcp-nets with reachability graphs*. Fundam. Inform (2006) **74**, 375–390.
[17] Szpyrka M., *Sieci petriego w modelowaniu i analizie systemów współbieżnych*. WNT, 2008.
[18] Meunier J.N., Lippert F., Jadhav R., Harding N., *MDA and real-time java: the hidoors project*. Technical Report – University of Kent at Canterbury Computing Laboratory (2004) **17**, 89–95.
[19] Oliver I., *Applying UML and MDA to real systems design*. In Date, 2005.
[20] Object Management Group, Common warehouse metamodel, 2003.
[21] Object Management Group, XML metadata interchange XMI, v2.0., 2005.
[22] Object Management Group, Meta Object Facility MOF 2.0 core final adopted specification, 2004.
[23] Object Management Group, UML profile for modeling and analysis of real-time and embedded systems, 2007.
[24] Object Management Group, UML profile for schedulability performance and time., 2007.
[25] Ward P.T., Mellor S.J., *Structured development for real-time systems*. Yourdon London, 1985.
[26] Sunye G., Guennec A.L., Jezequel J., *Using UML action semantics for model execution and transformation*. Inf. Syst. (2002) **27**, 445–457.
[27] Szmuc T., *Modele i metody inżynierii oprogramowania systemów czasu rzeczywistego*. Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH, 2001.
[28] Nickel U., Niere J., Zuendorf A., *The fujaba environment*. In Icse, 2000.