mgr inż. Piotr Trojanek
prof. nzw. dr hab. Cezary Zieliński
Instytut Automatyki i Informatyki Stosowanej
Politechnika Warszawska

SPECYFIKACJA ZŁOŻONYCH SYSTEMÓW ROBOTOWYCH

Przez złożone systemy robotowe rozumiane są takie systemy, które składają się z wielu robotów, bądź też kiedy roboty wchodzące w ich skład wyposażone są w znaczną liczbę czujników lub efektorów. Ich cechą wspólną jest występowanie skomplikowanych interakcji między elementami systemu. W takim przypadku znaczenia nabierają systematyczne metody oraz narzędzia, które wspierają wytwarzanie sterowników tej klasy systemów. Odgrywają one istotną rolę na wszystkich etapach pracy, począwszy od specyfikacji, przez implementację, aż do testowania i sporządzenia dokumentacji.

SPECIFICATION OF COMPLEX ROBOT SYSTEMS

Complex robot systems are those composed of many robots or those, where many sensor and effectors devices are used. Both of them can be characterized with complicated interactions between elements of the system. In this case it is important to use systematized methods and tools, which aids construction of the controllers. They play important role in all the stages of the development, from the specification up to testing and documentation.

1. WPROWADZENIE

W ostatnim czasie coraz większą popularność zyskują metody inżynierii oprogramowania związane z wytwarzaniem opartym na modelach (ang. *model-driven engineering*) [4, 5]. Pozwalają one proces tworzenia oprogramowania potraktować jako zbudowanie modelu (specyfikacji), a następnie jego zautomatyzowane przetwarzanie w celu wytworzenia artefaktów, takich jak szkielet kodów źródłowych czy dokumentacja. W tym podejściu wysiłek skoncentrowany jest nie na wytworzeniu pojedynczej aplikacji, ale na opracowaniu metod specyfikacji modelu systemu oraz jego transformacji do niezbędnych artefaktów. Zarówno metody, jak i transformacje, które powstają, nie są związane z konkretną aplikacją, ale pozostają uniwersalne dla całej dziedziny problemu. Jako takie mogą być wielokrotnie stosowanie do rozwiązywania zadań w obrębie tej samej klasy.

Kluczowe dla całego procesu jest zatem określenie zestawu pojęć dziedziny (słownika) oraz zasad ich wzajemnego łączenia (gramatyki) tak, aby w wyniku otrzymać dedykowany język służący do specyfikacji złożonych systemów robotowych. Jego zaletami są przede wszystkim większa zdolność ekspresji oraz precyzja formułowanych wyrażeń. Dopiero wtedy można przystąpić do opracowania reguł transformacji modelu systemu na artefakty niezbędne w procesie wytwarzania, dokumentowania oraz testowania sterowników robotowych.

W dalszej części przedstawiony został sposób konstruowania zbioru pojęć dedykowanych do specyfikacji systemów obrotowych oraz ich wzajemnych relacji, a następnie nadawania im znaczenia (semantyki).

2. DEFINICJA POJĘĆ DZIEDZINY

Język używany do specyfikacji modeli nazywany jest *metamodelem*, stąd język używany do definiowania języka jest *meta-metamodelem*. W celu uniknięcia rekursji meta-metamodele zazwyczaj używane są do zdefiniowania samych siebie. Formalne definiowanie modeli wymaga, aby zarówno metamodel jak i meta-metamodel były także zdefiniowane formalnie. Przykładowe meta-metamodele to diagramy encji [2], czy składnia EBNF (*Extended Backus-Naur Form*) [7].

Środowiska wytwarzania opartego na modelach dostarczają tak że swoje specyficzne meta-metamodele, jak Ecore dla Eclipse Modeling Framework [6], GOPRR (*Graph, Object, Property, Role and Relationship*) dla MetaEdit+ [4].

Wybór meta-metamodelu uzależniony jest przede wszystkim od dostępności narzędzi wspierających stworzenie kompletnego środowiska modelowania. MetaEdit+ było jednym z pierwszych dostępnych komercyjnie narzędzi związanych z omawianym podejściem [4]. Pakiet *Eclipse Modeling Framework* [6] jest konkurencyjnym, otwartym środowiskiem. Dostarcza ono zaawansowanego wsparcia dla języków modelowania zarówno o graficznej, jak i tekstowej notacji, jak również narzędzi transformacji w artefakty tekstowe. Poszczególne składniki środowiska tworzone są w oparciu o standardy OMG (*Object Management Group*).

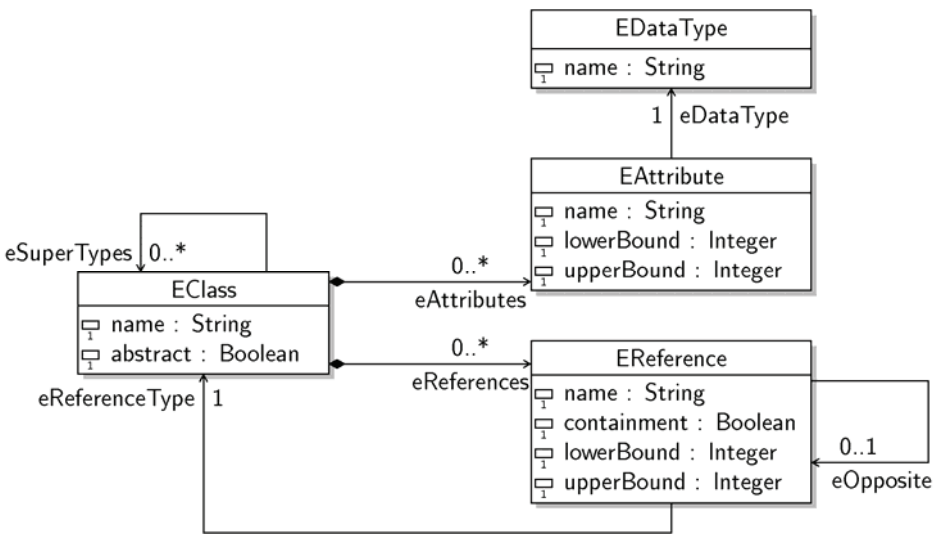
2.1. Wprowadzenie do meta-modelu Ecore

Podzbiór meta-metamodelu Ecore został przedstawiony przy użyciu diagramu (rys. 1) wzorowanego na diagramach klas UML [11]. Klasa reprezentowana jest przy użyciu prostokąta podzielonego na dwie części – górnej zawierającej nazwę klasy oraz dolnej z jej atrybutami i typami danych. Symbole używane dla atrybutów z różnymi wartościami dolnego i górnego ograniczenia ilościowego zostały zestawione w tabeli.

Rodzaj argumentu	Ograniczenie dolne	Ograniczenie górne	Symbol
opcjonalny 0		1	▫
wymagany 1		1	▫ (1)

Zdefiniowane są trzy rodzaje związków, oznaczane różnymi symbolami strzałek. *Dziedziczenie* używane jest w celu zebrania właściwości (atrybutów i relacji) innych klas, tzw. bazowych. Pojęcia, które nie reprezentują obiektów dziedziny, ale są pożyteczne dla modelowania właściwości innych obiektów, przedstawiane są jako klasy *abstrakcyjne*, z nazwami pisanymi kursywą. *Zawieranie* używane jest do oznaczenia relacji kompozycji, zaś *referencja* obrazuje połączenia między pojęciami. Nazwy relacji oraz ich dolne i górne ograniczenia ilościowe umieszczane są przy końcach strzałek.

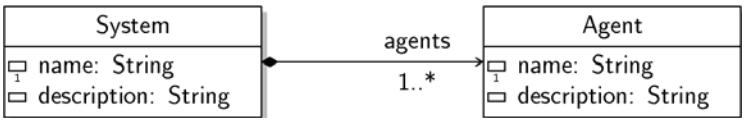
Rodzaj relacji	Oznaczenie
dziedziczenie	——▷
zawieranie	◆——→
referencja	——→



Rys. 1. Diagram Ecore dla używanego podzbioru meta-metamodelu

2.2. Meta-model systemu robotowego

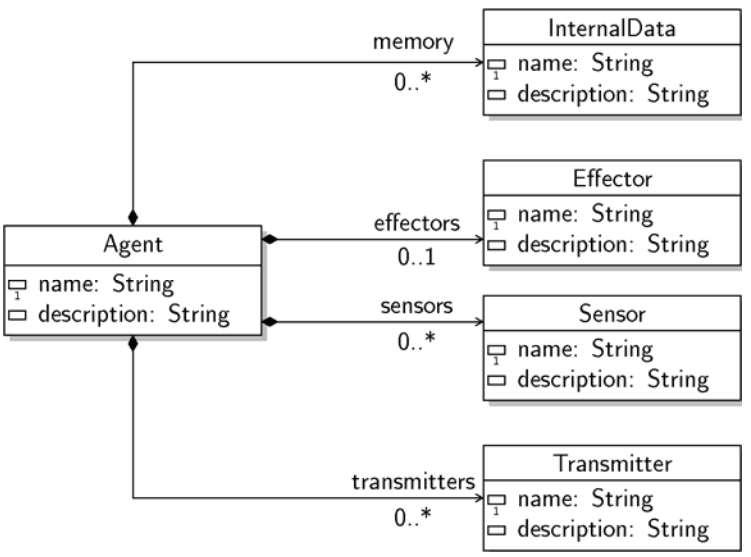
Wyjściowym pojęciem w systemie robotowym jest sam System (rys. 2). Atrybut name jest używany do identyfikacji instancji, zaś description jest dodany jedynie w celach dokumentacyjnych. Te ogólne atrybuty będą wspólne dla wielu pojęć dominujących i mogą zostać wykorzystane do tworzenia nazw oraz komentarzy w generowanych szkieletach kodów źródłowych oraz dokumentacji. System składa się z niezerowej liczby agentów, która zazwyczaj odpowiada liczbie robotów.



Rys. 2. Relacja pomiędzy pojęciem systemu a składającymi się na niego agentami

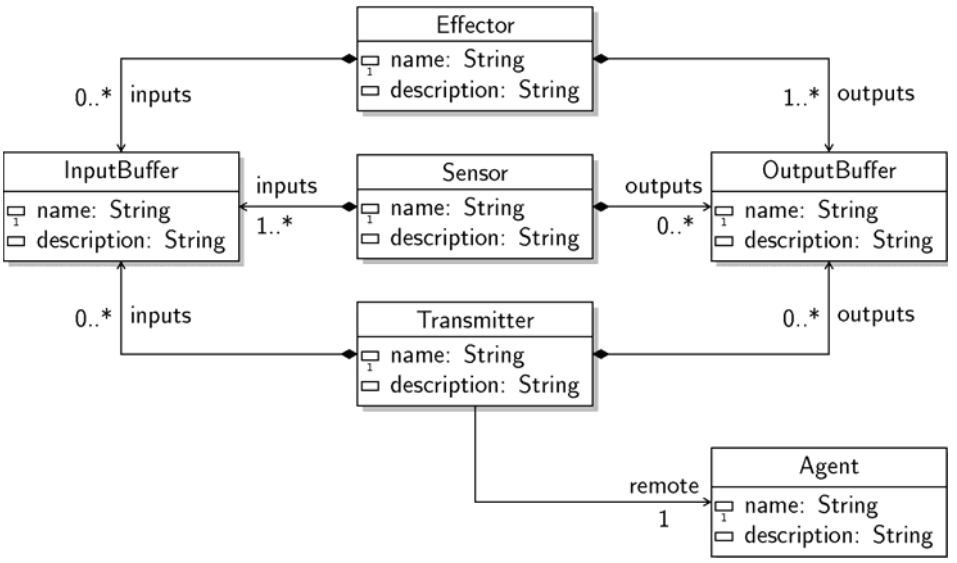
2.2.1. Strukturalna dekompozycja agenta

Agent wykorzystuje sensory oraz efektory służące odpowiednio do postrzegania i wpływania na otoczenie, jak również transmitery, służące do komunikacji z innymi agentami (rys. 3). Zakłada się przy tym, że agent, który zazwyczaj jest utożsamiany z pojedynczym robotem, steruje nie więcej niż jednym efektorem. Jednocześnie nie ma ograniczeń na liczbę sensorów oraz transmiterów. Pamięć jest związana z wewnętrzną zdolnością agenta do przetwarzania danych. Jest to wymagany składnik, bez którego agent nie może istnieć, ponieważ wtedy nie byłby w stanie przetwarzać żadnej informacji.



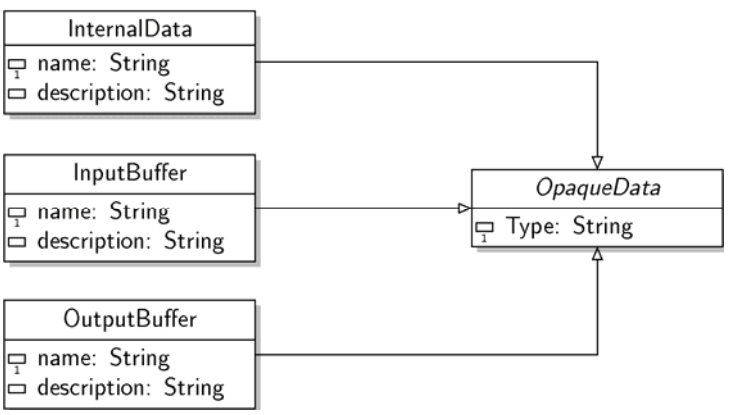
Rys. 3. Relacja łącząca pojęcie agenta i jego podsystemy

Sensory, efektory i transmitery różnią się przeznaczeniem, jednak z punktu widzenia projektanta systemu wszystkie one posiadają zdolność odbierania danych wejściowych oraz publikowania danych wyjściowych. Składają się one zatem z buforów służących do odbierania i wysyłania danych, jak zostało to zamodelowane przez pojęcia odpowiednio InputBuffer i OutputBuffer, lecz różnią się dolnym ograniczeniem liczebności relacji inputs i outputs (rys. 4). Sensory wymagają przynajmniej jednego bufora wejściowego, zaś efektory przynajmniej jednego bufora wyjściowego, co odzwierciedla ich zasadniczy kierunek przepływu danych. W zależności od właściwości sprzętu czujniki mogą tak że przyjmować zlecenia konfiguracyjne, a efektory udostępniać proprioreceptywne dane o ich aktualnym stanie. Z transmiterami nie są związane ograniczenia liczby buforów, ale wymagają one określenia zdalnego agenta (relacja remote), z którym są związane. Zakłada się, że relacja ta nie wskazuje na właściciela transmitera, jak również, że dla każdego transmitera istnieje przynajmniej jeden bufor wejściowy lub wyjściowy.



Rys. 4. Bufory wejściowe i wyjściowe używane przez podsystemy składowe agenta

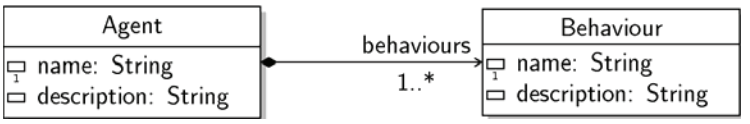
Typ danych zwi ązanych z buforam i wej ściowymi i wyj ściowymi, jak równie ż z danym i przechowywanymi w pam ięci wewn ętrznej, nie m a wp ływu na ogóln ą architektur ę, mi mo t o powinien zosta ć zdefiniowany z dwóch powodów. Po pierwsze, aby przypisa ć szczegó łowe znaczenie wspom nianym poj ęciom. Po drugie, aby zwi ązać generowany szkielet kodu z typam i danych zdefiniowanymi w języku programowania używanym do implementacji. Połączenie między specyfikacją a j ęzykiem program owania ogólnego przeznaczenia jest realizowane przez poj ęcie OpaqueData, które łączy instancje klas buforów i danych przechowywanych w pam ięci wewnętrznej z nazwami typów danych zgodnie z atrybutem Type (rys. 5).



Rys. 5. Połączenie ze zdefiniowanym typem danych

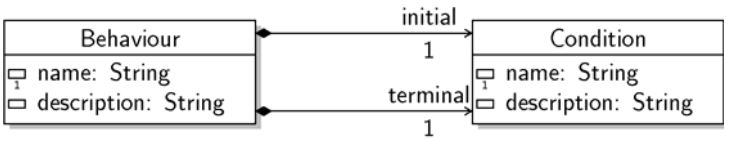
2.2.2. Dekompozycja działania agenta

Aspekt wykonawczy system u wielorobotowego wym aga dekom pozycji jego dzia łania na akcje przypisane poszczególnym agentom . W wi ększości zada ń pojedynczy agent m usi wykonywa ć więcej ni ż jedn ą aktywno ść w czasie swojego życia. Ka żda niezale żna aktywno ść agenta jest nazywana zachowaniem i razem tworzą one repertuar jego zdolno ści (rys. 6). Dolne ograniczenie relacji behaviours wynosi 1, poniewa ż każdy użyteczny agent powinien by ć w stanie wykonywa ć przynajmniej jedną czynność.



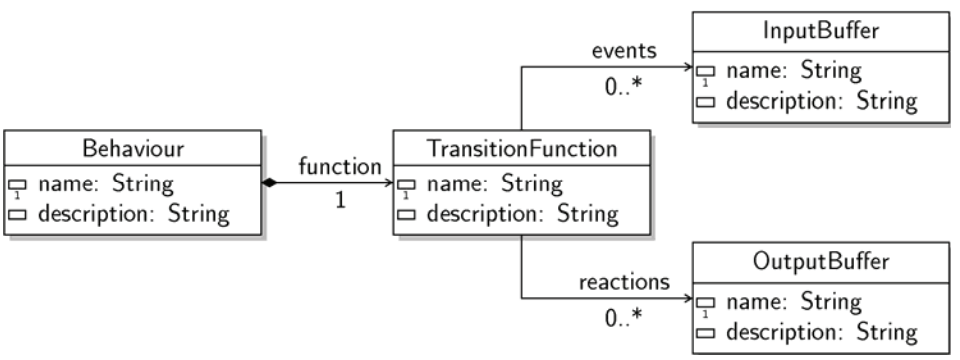
Rys. 6. Powiązania między agentem a jego zachowaniami

Istnieje wiele sposobów określania szczegółów pojedynczego zachowania, jak również reguł rządzących przechodzeniem między nimi [1]. Zazwyczaj określa się zarówno warunek początkowy jak i końcowy, które odpowiednio definiują kiedy zachowanie powinno się rozpocząć oraz zakończyć (rys. 7). Każdy taki warunek reprezentuje predykat (funkcję Boolowską), którego argumentem jest aktualny stan agenta (tj. zawartość pamięci wewnętrznej oraz stan buforów wejściowych i wyjściowych). Zakłada się przy tym, że określenie wartości predykatu (ewaluacja) nie wpływa na stan agenta, tj. zawartości jego zmiennych wewnętrznych oraz stanu buforów wyjściowych.



Rys. 7. Warunek początkowy i końcowy zachowania

Istota zachowania definiowana jest przez *funkcję przejścia* [14], która na podstawie aktualnego stanu agenta (tj. zawartości pamięci wewnętrznej oraz stanu buforów wejściowych i wyjściowych) wyznacza wartości do wysłania przez bufory wyjściowe oraz uaktualnia te przechowywane w pamięci wewnętrznej (rys. 8). Z funkcją przejścia związany jest także zbiór zdarzeń zewnętrznych (relacja events), który określa dane wymagane do kolejnego obliczenia jej wartości i w ten sposób realizowania pętli sterowania. Związek ten oznaczony jest powiązaniem z buforami wejściowymi sensorów, efektorów oraz transmiterów, które stanowią dla agenta jedyny sposób postrzegania jego składników jak i otoczenia. Funkcja przejścia umieszcza wynik obliczeń w określonych buforach wyjściowych (relacja reactions) i w ten sposób agent oddziałuje na otoczenie oraz swoje składniki. Zakłada się, że zakres relacji wiążącej bufory z funkcją przejścia nie wychodzi poza pojedynczego agenta.



Rys. 8. Zachowanie jako zbiór funkcji przejścia oraz powiązanie z buforami wejściowymi i wyjściowymi

4. SEMANTYKA POJĘĆ JĘZYKA DZIEDZINY

Kluczowe dla procesu specyfikacji jest przypisanie znaczenia używanym pojęciom, zwłaszcza jeśli chodzi o aspekt wykonawczy systemu wieloobrotowego [13]. Semantykę najlepiej jest zdefiniować przez określenie reguł, które jednoznacznie odwzorują stosowane pojęcia na model o już

określonym sposobie działania, np. język programowania ogólnego zastosowania. Takie rozwiązanie związuje jednak metodę specyfikacji z konkretną platformą (językiem, systemem operacyjnym), których szczegóły często przesłaniają ogólny obraz działania systemu. Na poziomie specyfikacje korzystniej jest zatem posłużyć się modelem, który abstrahuje od szczegółów implementacyjnych, a pozwala skupić uwagę na zagadnieniach przepływu danych oraz sterowania.

4.1. Wprowadzenie do sieci Petriego

Sieci Petriego są powszechnie stosowanym modelem specyfikacji i analizy systemów współbieżnych [9]. Ich zalety, w porównaniu z innymi modelami, to intuicyjna reprezentacja graficzna oraz ugruntowane metody formalnej analizy właściwości opisywanego systemu. Były one już też wykorzystywane do opisu konkretnych systemów i zadań robotów [12].

Sieć Petriego jest definiowana jako trójka *(P, T, A)*, gdzie *P* jest skończonym zbiorem miejsc, *T* jest skończonym zbiorem przejść (P i T są rozłączne, $P \cap Q = \varnothing$), zaś A określa relację przepływu, $A \subseteq (P \times T) \cup (T \times P)$. W notacji graficznej sieć Petriego jest przedstawiana jako skierowany graf dwudzielny z rozłącznymi zbiorami wierzchołków odpowiadającymi P i T oraz łukami zgodnie z relacją przepływu A. Miejsca sieci są reprezentowane graficznie za pomocą okręgów, przejścia zaś − za pomocą prostokątów.

Miejsca reprezentują zazwyczaj stan systemu u im mogą zawierać dowolną niezerową liczbę znaczników. Znakowanie M sieci Petriego zdefiniowane jest jako liczba znaczników w każdym z miejsc, $M : P \to \mathrm{N}_0$. Przejście jest *aktywne*, jeśli w każdym miejscu połączonym z nim za pomocą łuku wejściowego znajduje się przynajmniej jeden znacznik. *Odpalenie* aktywnego przejścia jest operacją niepodzielną, która pochłania jeden znacznik z każdego miejsca połączonego z nim łukiem wejściowym i produkuje znacznik w każdym z miejsc połączonych łukami wyjściowymi.

Przejścia z wielomałukami wejściowym są używane do synchronizacji współbieżnych czynności, zaś wiele łuków wyjściowych modeluje rozpoczęcie niezależnych działań. W celu lepszego modelowania przepływu sterowania przejścia sieci często rozszerza się o *dozory*. Są to wyrażenia o charakterze predykatów, oznaczające warunki, które dodatkowo muszą być spełnione, aby dane przejście było aktywne. W reprezentacji graficznej umieszcza się je w nawiasach kwadratowych obok przejścia.

Jedno z bardziej popularnych rozszerzeń omówionego modelu, *hierarchiczne sieci Petriego* [8], pozwalają modelować system na różnych poziomach szczegółowości. Przy użyciu *podstawienia przejścia*, jednego z podstawowych pojęć rozszerzenia, możliwe jest zwinięcie fragmentu *modułu* sieci do pojedynczego przejścia. Zwinięta część przedstawiana jest na osobnym *module*. Moduły sieci wyższego i niższego poziomu zawierają odpowiednio miejsca *gniazdowe* (*gniazda*) *portowe* (*porty*). Relacja gniazdo-port wiąże porty modułu z miejscami gniazdowymi i podstawianego przejścia.

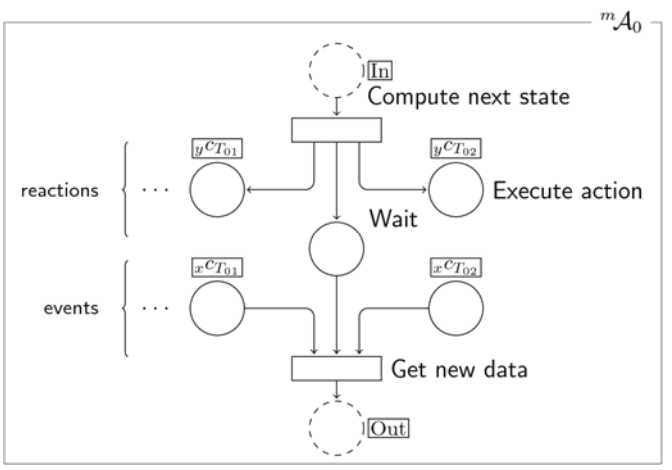
Fuzje są drugim pojęciem hierarchii, które pozwala na wymianę znaczników między różnymi modułami sieci. Miejsca należące do jednej fuzji mogą być narysowane w kilku egzemplarzach, ale pojęciowo reprezentują ten sam obiekt.

Podstawienia przejść razem z fuzjami stanowią dwa główne mechanizmy wspomagające stopniowe projektowanie systemu, odpowiednio przez jego uściślanie i kompozycję.

Podstawienie przejścia reprezentowane jest graficznie za pomocą prostokąta o bokach narysowanych podwójną linią z etykietą nazwy modułu zawierającego podstawianą sieć, zaś porty modułów − jako miejsca narysowane linią przerywaną z etykietami In i Out oznaczającymi odpowiednio port wejściowy i wyjściowy. Miejsca należące do fuzji oznaczane są wspólną etykietą w prostokątnej ramce.

4.2. Sieci opisujące działanie agenta

Działanie agenta oparte jest na funkcjach przejścia (rys. 8), które stanowią podstawowy element warstwy sterowania robota [13]. Sieć, która je reprezentuje, opisuje przechodzenie między czterema kolejnymi etapami: obliczenia kolejnego stanu agenta, wykonanie akcji, oczekiwanie na zmianę stanu systemu oraz pobranie nowych danych niezbędnych do powtórzenia kroku sterowania (rys. 9). Jedno wykonanie tej sekwencji stanowi *akcję elementarną* ${}^{m}A_j$, gdzie j to identyfikator agenta, zaś m wskazuje na funkcję przejścia związaną z konkretnym zachowaniem. Znacznik pojawia się w sieci w miejscu portowym oznaczonym etykietą In. Obliczenie wartości funkcji reprezentowane jest przejściem (etykieta Compute next state), po czym w każdym miejscu sieci związanym z buforem wyjściowym (zgodnie z relacją reactions) pojawia się znacznik. Modeluje to asynchroniczne wysłanie danych przez agenta (etykieta Execute action), który następnie przechodzi do stanu oczekiwania (etykieta Wait). Pozostaje on w nim do czasu pojawienia się danych w buforach wejściowych, co modelowane jest jako obecność znaczników w związanych z nimi miejscach, zgodnie z relacją events. Znaczniki konsumowane są przez odpalenie przejścia (etykieta Get new data), po czym ostatni znacznik opuszcza moduł w miejscu portowym oznaczonym etykietą Out. Bufory wyjściowe jak i wejściowe reprezentowane są fuzjami, dzięki czemu mogą się do nich odwoływać zarówno różne funkcje przejścia, jak i agenty. Etykiety tych miejsc mają postać ${}_{x}c_{T_{01}}$, gdzie indeks lewy dolny oznacza odpowiednio bufor wejściowy (x) lub wyjściowy (y), indeks prawy dolny podsystem agenta (T – transmiter, V – czujnik, E – efektor). Indeks prawy dolny drugiego rzędu identyfikuje konkretny bufor w ramach danego podsystemu [14]. Na rys. 9 zaprezentowano przykładową funkcję przejścia agenta a_0, która oblicza wartości do przesłania przez transmitery dla dwóch koordynowanych agentów a_1 i a_2, przy czym pierwsza cyfra indeksu oznacza właściciela bufora, a druga agenta zdalnego (zgodnie z relacją remote). Samo przesłanie danych między agentami zostało pominięte na rysunku – jest ono modelowane pojedynczym przejściem łączącym odpowiednie bufory wyjściowy i wejściowy.

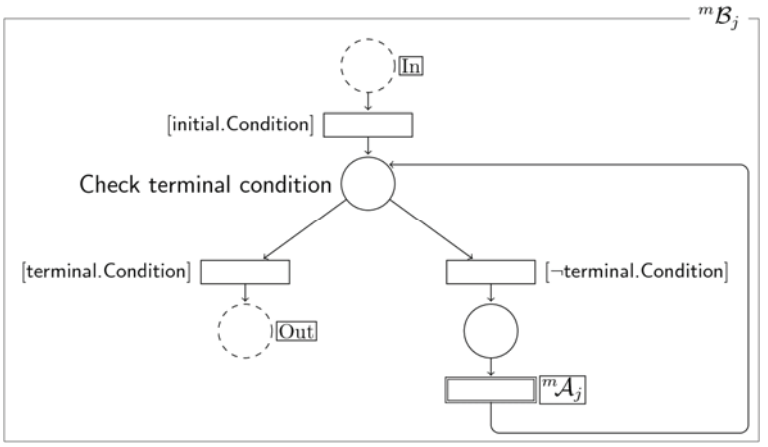


Rys. 9. Moduł sieci Petriego opisujący funkcję przejścia

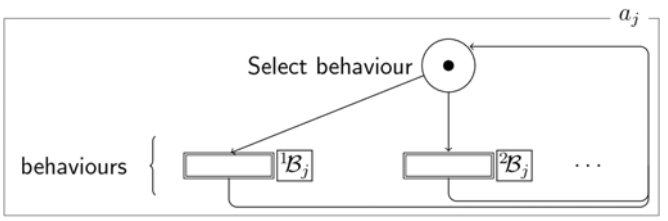
Tak zdefiniowany moduł zawierający sieć Petriego dla pojedynczej funkcji przejścia może służyć za uściślenie podstawienia przejścia w module sieci wyższego poziomu, która opisuje cykliczne wykonywanie akcji elementarnej aż do spełnienia warunku końcowego zachowania ${}^{m}B_j$ (rys. 10). Sieć ta zawiera przejścia z dozorami związanymi odpowiednio z warunkiem początkowym (dozór [initial.Condition]) oraz końcowym (dozór [terminal.Condition] i jego negacja oznaczona symbolem ¬), które wcześniej zostały związane z zachowaniem (rys. 7).

Główny moduł sieci opisującej działanie agenta, a_j, zawiera podstawienia przejść związanych z modułami opisującymi poszczególne zachowania (rys. 11). Znacznik (umieszczony w miejscu oznaczonym etykietą Select behaviour) wskazuje na początkowe znakowanie sieci, tj. stan agenta

w chwili rozpoczęcia działania systemu. Każde z podstawionych przejść oznacza jedno z zachowań agenta, które wcześniej zostały dla niego określone (relacja behaviours) (rys. 6). Działanie agenta opisane tą siecią polega na wyborze jednego spośród zachowań ze spełnionym warunkiem początkowym.



Rys. 10. Moduł sieci Petriego opisujący pojedyncze zachowanie



Rys. 11. Główny moduł sieci opisujący wybór zachowania przez agenta

Przedstawione sieci stanowią strukturę hierarchiczną, gdzie na najniższym poziomie znajduje się pojedyncza akcja elementarna (rys. 9), która jest cyklicznie wywoływana w warstwie zachowania (rys. 10). Wybór spośród wcześniej zdefiniowanych zachowań tego, które w danym stanie systemu powinno być aktywne, dokonywany jest w sieci najwyższego poziomu. Wykonanie zadania rozumiane jest tutaj nie tylko jako przeprowadzenie wcześniej zdefiniowanej sekwencji zachowań [5], ale wybór takiego, które w danym stanie systemu zostało przewidziane przez projektanta przez uważną konstrukcję predykatów związanych z ich warunkami początkowymi.

5. PODSUMOWANIE

W artykule zdefiniowano zbiór pojęć (tj. ontologię) używanych do specyfikacji złożonych systemów robotowych. Razem z relacjami i łączącymi określone pojęcia stanowi on język modelowania specyficzny dla omawianej dziedziny. Został on zdefiniowany formalnie przy użyciu narzędzi wytwarzania opartego na modelach, dzięki czemu możliwe jest wykorzystanie istniejących narzędzi wspomagających generowanie artefaktów takich jak szkielety implementacji, czy dokumentacja.

Znaczenie pojęć i relacji języka zostało zdefiniowane przez schematy obrazujące ich przekształcenie na sieci Petriego, które są modelem o formalnie zdefiniowanej semantyce. W ten sposób zaprezentowana metoda jest niezależna od języka programowania czy systemu operacyjnego, które zostaną wykorzystane do implementacji. Sieci Petriego pozwalają opisywać zarówno przepływ sterowania (z uwzględnieniem współbieżności), jak i danych, oraz modelować komunikację w systemie rozproszonym. Warstwowa struktura układu sterującego została

zamodelowana przy u życiu hierarchii sieci, które jest jedynie rozszerzeniem u łatwiającym modelowanie, zatem nie wp ływa ono na m ożliwość stosowania dost ępnych m etod dla analizy właściwości sieci (np. wykrywania zakleszczeń).

Zaproponowana metoda specyfikacji abstrahuje konkretnej notacji, w której m a być dokonywana specyfikacja. Mo żliwe jest zatem tworzenie jej zarówno w form ie tekstowej jak i przypisanie poszczególnym pojęciom reprezentacji graficznej. Tem at ten nie zosta ł tutaj bardziej szczegó łowo poruszony, poniewa ż dla zastosowanego m eta-metamodelu Ecore istniej ą m etody autom atycznej generacji notacji konkretnej. Zatem z punktu widzenia autorów jest to zadanie m ało interesuj ące [10].

Warto zauwa żyć, że zdefiniowanie warstwy w oparciu o sieci Petriego pozwala w ogólno ści na stosowanie bardziej z łożonych struktur ni ż pojedyncze zachowanie, dla przyk ładu ju ż na poziom ie sieci wyra żane m ogą by ć sekwencje czy nawet zestawy wspó łbieżnie wykonywanych zachowa ń oraz punkty ich synchronizacji.

Praca jest finansowana przez grant badawczy MNiSW N514128733.

BIBLIOGRAFIA

1. R.C. Arkin. *Behavior-based robotics.* The MIT Press, 1998.
2. P. Chen, P. Pin-Shan . *The entity-relationship model: Toward a unified view of data.* ACM Transactions on Database Systems, 1: 9–36, 1976.
3. G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. *Model-integrated development of embedded software.* Proceedings of the IEEE, 91(1):145–164, 2003.
4. S. Kelly, J. P. Tolvanen. *Domain-specific modeling: enabling full code generation.* W iley-IEEE Computer Society Press, April 2008.
5. Gat, E.; Others. *On three-layer architectures.* Artificial Intelligence and Mobile Robots, s. 195–210. AAAI Press, 1998.
6. R. C. Gronback. Eclipse Modeling Project: *A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 2009.
7. International Standard: Inform ation technology. *Syntactic metalanguage – Extended BNF.* ISO/IEC 14977:1996(E), First Edition, December 1996.
8. K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modeling and Validation of Concurrent Systems.* Springer-Verlag New York Inc, 2009.
9. Murata T.: Petri nets: *Properties, analysis and applications.* Proceedings of the IEEE, 77(4):541–580, 1989.
10. Object Managem ent Group. *Human-Usable Textual Notation (HUTN) Specification.* August 2004.
11. Object Managem ent Group. *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.3.* Technical report, May 2010.
12. M. Silva, R. Valette. *Petri nets and flexible manufacturing*. Advances in Petri Nets 1989. Lecture Notes in Computer Science, vol. 424, s. 374–417. Springer 1990.
13. P. Trojanek, C. Zieliński. *Specyfikacja systemów wielorobotowych oparta na sieciach Petriego.* Metody wytwarzania i zastosowania systemów czasu rzeczywistego. Red. L. Trybus i S. Samolej. Wydawnictwa Komunikacji i Łączności, 2010.
14. C. Zieliński. *Systematic approach to the design of robot programming frameworks.* In Proceedings of the 11th IEEE International Conference on Methods and Models in Automation and Robotics (on CD), s. 639–646. Technical University of Szczecin, 2005.