AUTOMATYKA • 2010 • Tom 14 • Zeszyt 3/2

Radosław Klimek*, Paweł Skrzyński*, Michał Turek*

Weryfikacja aktywności systemów modelowanych na bazie architektury SOA – prowadzona w oparciu o wymagania zadane diagramami przypadków użycia

1. Wprowadzenie

Język UML stał się w tej chwili standardem przemysłowym i jest szeroko używany jako język modelowania ogólnego przeznaczenia. Pomimo że wykorzystywanie UML do specyfikacji oprogramowania oraz systemów informatycznych jest coraz częstsze, to możliwość użycia języka UML nie musi być ograniczona wyłącznie do takich zastosowań. W obecnej wersji UML dostarcza projektantom aż trzynaście typów diagramów, które mogą być wykorzystane na różnych etapach rozwoju systemu – poczynając od fazy analizy do wdrożenia i konserwacji systemu. Jednym z pierwszych etapów prac nad stworzeniem systemu jest określenie wymagań funkcjonalnych systemu. W UML dokonuje się tego głównie za pomocą diagramu przypadków użycia. Artykuł dotyczy zagadnień formalnej weryfikacji przypadków użycia oraz scenariuszy przypadków użycia zapisanych w postaci diagramów czynności z wykorzystaniem wnioskowania dedukcyjnego. Do specyfikacji systemu i żądanych jego własności została wykorzystana logika temporalna, jako najlepiej opisująca kwestie żywotności i bezpieczeństwa systemów [4] natomiast wnioskowanie dedukcyjne odbywa się z wykorzystaniem metody tablic semantycznych [1], która może przebiegać automatycznie i jest ciekawą alternatywą dla podejścia tradycyjnego (np. rezolucja), umożliwia m.in. łatwe wskazanie błędów w specyfikacji logicznej systemu. Zaproponowane zostały, co należy podkreślić, metody pozyskiwania formuł logiki temporalnej bezpośrednio ze scenariuszy przypadków użycia zapisanych w UML-owych diagramach czynności, a zazwyczaj właśnie uzyskiwanie specyfikacji systemu stanowi wąskie gardło takiego podejścia dedukcyjnego. W literaturze spotyka się prace dotyczące weryfikacji diagramów aktywności, ale dotyczą one raczej podejścia opartego na eksploracji stanów, por. [5].

Zaletą prezentowanego podejścia jest możliwość wstępnej weryfikacji systemu na bardzo wczesnym etapie procesu modelowania, co może się przyczynić do wykrycia błędów w specyfikacji funkcjonalnej systemu i pomóc w redukcji kosztów wytwarzania systemu.

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

Istnieją już metodyki i narzędzia CASE (przykładowo Rhapsody firmy IBM), które wspierają procesy wizualnej symulacji zachowania modelowanego w języku UML systemu i na ich podstawie umożliwiają wygenerowanie wykonywalnego kodu realizującego zadaną funkcjonalność. Wymaga to jednak precyzyjnego zdefiniowania architektury systemu przy pomocy dużej liczby diagramów UML. Jest to proces wybiegający poza fazę analizy wymagań funkcjonalnych i autorzy nie znają narzędzia CASE dostarczającego wsparcie dla analityków już na etapie analizy wymagań. Dodatkową zaletą prezentowanego podejścia jest to, że weryfikacja jest przeprowadzana w sposób formalny, wobec czego stanowi to wartość dodaną wobec samej symulacji modelu, która nie musi prowadzić do wykrycia wszystkich błędów. Zastosowanie prezentowanego formalizmu ma następujące zalety:

- redukcja kosztów wytwarzania oprogramowania,
- automatyzacja procesu wnioskowania,
- dobra skalowalność metody,
- ekstrakcja formuł logiki temporalnej bezpośrednio ze scenariuszy przypadków użycia zapisanych w postaci diagramów czynności,
- możliwość wykrywania anomalii w wymaganiach funkcjonalnych,
- możliwość sprawdzenia osiągalności wszystkich aktywności w modelu.

W procesie adaptacji MDA (*Model Driven Architecture*) duży nacisk powinien zostać położony na możliwość automatycznych przejść pomiędzy kolejnymi krokami. Prezentowana metodologia pozwala na automatyczne wygenerowanie formuł odpowiadających scenariuszowi przypadku użycia zapisanego w formie diagramu czynności, co może stanowić dobry punkt wyjścia do dalszych etapów modelowania.

2. Diagramy przypadków użycia i czynności

Mechanizm przypadku użycia (*use-case*) jest jednym z narzędzi, jakie mogą zostać użyte w procesie tworzenia oprogramowania. Jego definicja brzmi „przypadek użycia jest sekwencją transakcji w systemie, której zadaniem jest zaoferowanie mierzalnej wartości klientowi systemu" [2]. Zestaw przypadków użycia definiuje powinności systemu w stosunku do użytkownika, jako zestaw transakcji rozumie się pewne operacje, których wykonanie implikuje pewne skutki, które to są właśnie wartością owego przypadku. Dla przykładu prostym przypadkiem użycia systemu obsługującego sklep jest sprzedaż towaru, gdzie aktorami są sprzedawca oraz klient.

Istnieją dwa rodzaje przypadków użycia – systemowe oraz biznesowe. Pierwsze dotyczą interakcji użytkownika z systemem i są osadzone na niskim poziomie abstrakcji, drugie odnoszą się do procesów biznesowych i definiują, w jaki sposób proces reaguje na akcję użytkownika – są bardziej ogólne i komponują się z systemowych przypadków użytkownika.

Przypadek użycia składa się z trzech elementów:

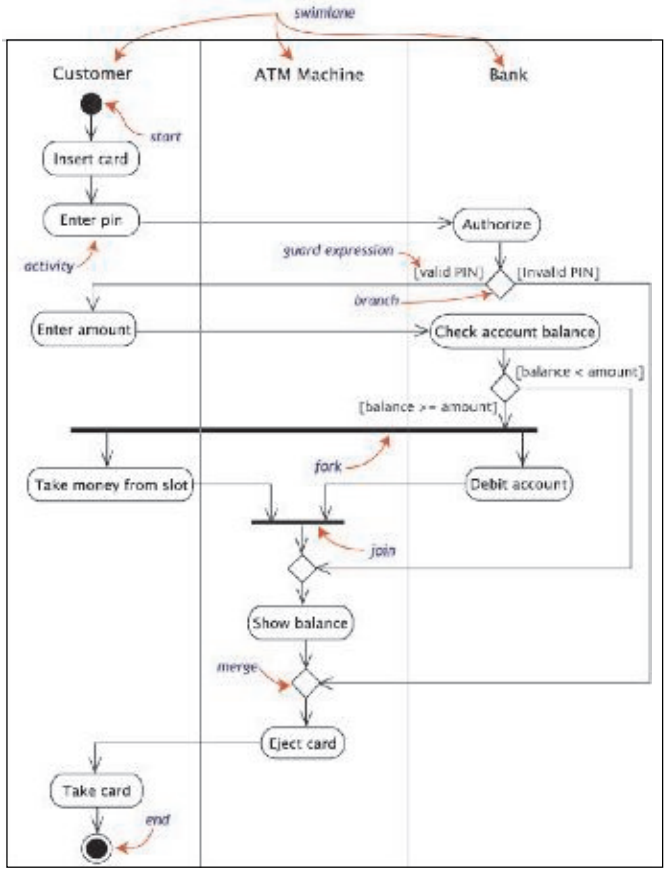
- aktora głównego i ewentualnie aktorów drugoplanowych,
- celu,
- scenariusza.

Elementem kluczowym przypadku użycia jest oczywiście scenariusz. Zawiera on opis sekwencji transakcji, jakie składają się na przypadek użycia, czyli jego logikę. Dodatkowo, poza scenariuszem głównym, mogą istnieć scenariusze dodatkowe (zwane także awaryjny-

mi), które realizowane są, jeżeli zajdzie pewien warunek (przykładem wsparcia scenariuszy dodatkowych ze strony języka programowania jest mechanizm wyjątków). Do opisu scenariusza, jak już powiedziano, często wykorzystuje się diagramy czynności (inaczej aktywności).

Diagramy te służą do modelowania dynamicznych aspektów systemu, zatem ich przeznaczenie jest bardziej ogólne. Przedstawiają one przepływ sterowania od jednej aktywności do innej. Na diagramach aktywności można też zobrazować zmiany zachodzące w obiekcie, gdy przechodzi on z jednej aktywności do drugiej w różnych fazach przepływu sterowania. Mogą być wykorzystane do zobrazowania, wyspecyfikowania, utworzenia lub udokumentowania dynamiki zestawu obiektów. Diagram ten może również zostać wykorzystany do modelowania przepływu sterowania wewnątrz aktywności. Jednak najczęściej są one wykorzystywane przy analizowaniu przypadków użycia, opisie skomplikowanego algorytmu sekwencyjnego i modelowaniu aplikacji z równoległymi procesami. Diagramy aktywności nie dostarczają nam informacji, jak obiekty się zachowują albo jak kolaborują ze sobą.

Rysunek 1 przedstawia scenariusz pobierania pieniędzy z bankomatu zapisany przy pomocy diagramu czynności.



Rys. 1. Przykładowy opis scenariusza przypadku użycia „Pobierz pieniądze z bankomatu” zapisany w formie diagramu czynności

3. Proponowana metodyka modelowania

Jak wspomniano, większość stosowanych obecnie metodyk wytwarzania oprogramowania zakłada rozpoczęcie tego procesu fazą drobiazgowej analizy wymagań, w ramach której określa się funkcjonalność systemu. UML dostarcza graficzną metodę opisu funkcjonalności systemu poprzez diagramy przypadków użycia, które określają funkcjonalność jakiej od systemu wymagają byty zewnętrzne (aktorzy) lecz nie określają na czym ona ma polegać. W dalszej fazie analizy należy określić, na czym te funkcjonalności polegają – odbywa się to poprzez dostarczenie dla każdego przypadku użycia jego scenariusza, który często sporządza się w formie tabelarycznej (opis taki jest de facto standardem przemysłowym w środowisku analityków), określając: aktora głównego, pozostałych aktorów, warunki początkowe, warunki końcowe oraz główny ciąg zdarzeń i alternatywne ciągi zdarzeń (jeśli takowe istnieją). Modelując system za pomocą języka UML, w otoczeniu przypadku użycia można się posłużyć diagramem czynności, który można łatwo stworzyć dla tak skonstruowanego opisu tabelarycznego: poszczególne kroki algorytmu opisującego ciąg zdarzeń traktowane są jako aktywności, do których połączenia używane są linie przepływu, rozwidlenia i złączenia. Należy podkreślić, że semantyka jaka jest dostarczana z diagramami aktywności pozwala na stworzenie diagramu czynności dla dowolnego opisu scenariusza przypadku użycia. Prezentowana w artykule metodyka pozwala natomiast na dokonanie formalnej weryfikacji diagramu czynności. Podejście umożliwi skonstruowanie zmodyfikowanego łańcucha modelowania, składającego się z następujących elementów: budowa diagramów przypadków użycia, określenie scenariuszy przypadków użycia w postaci stabelaryzowanej, konstrukcja diagramów czynności im odpowiadających, formalna weryfikacja powstałych diagramów czynności (aktywności). Proponowana w pracy metodyka analizy wymagań prowadzi przez następujące kroki:

- Konstrukcja diagramu przypadków użycia,
- Definicja scenariuszy przypadków użycia w formie tabelarycznej,
- Konstrukcja diagramów czynności dla scenariuszy,
- Formalna weryfikacja przy pomocy metody tablic semantycznych.

Najważniejszym krokiem wykonywanym przez projektanta jest konstrukcja diagramu aktywności dla scenariusza przypadku użycia.

Określając treść diagramów aktywności należy dokonać adaptacji aktorów z diagramów *use-case* – ustalając, które kroki scenariusza wykonywane są przez których aktorów oraz jakie zdarzenia inicjują kolejne kroki tego scenariusza. Na końcu należy się zastanowić, czy w diagramie nie występują aktywności rozbudowane, które mogą wymagać osobnego diagramu. Łącząc aktywności należy ustalić: po pierwsze kolejność ich przetwarzania, określając przy tym akcje, które mogą zachodzić równocześnie, jakie warunki muszą zaistnieć, żeby aktywność mogła mieć miejsce, gdzie jest konieczne wprowadzenie rozgałęzień oraz złączeń, w końcu, dla każdej aktywności ustalić, czy musi ona się zakończyć, by wykonanie scenariusza mogło być kontynuowane. Sytuacja szczególnie trudna do

wymodelowania to taka, kiedy system powinien obsługiwać jednoczenie wiele instancji tego samego aktora (funkcje systemu realizowane są współbieżnie), a ich obsługa jest wzajemnie od siebie uzależniona.

Dużym wyzwaniem jest stworzenie algorytmu dostarczającego rozwiązanie automatyzujące czynności związane z wygenerowaniem kompletu diagramów aktywności modelowanego systemu. Teoretycznie nie jest tu możliwe stworzenie transformacji uniwersalnej – czyli generującej diagramy aktywności zawierające odpowiedniki zestawu dowolnych, często współbieżnie realizowanych przypadków użycia. Przykładowo – decyzje o realizacji akcji alternatywnych projektowanych w ramach poszczególnych przypadków użycia mogą być uzależnione od realizacji innych akcji, prowadzonych na rzecz innej instancji aktora w tym samym czasie. Na obecnym etapie rozważań założone zostanie istnienie hermetyzacji szeregów aktywności związanych z obsługą różnych przypadków użycia. Jest to często stosowanym zabiegiem upraszczającym proces modelowania aktywności systemu. Przy takim założeniu można zaproponować algorytm przeprowadzający postać stabelaryzowaną specyfikacji przypadków użycia do diagramów aktywności. Jego implementacja umożliwi dostarczenie materiału do badań nad mechanizmami formalnej weryfikacji wymagań.

Na potrzeby przedstawionych rozwiązań zaproponowany zostanie zatem algorytm, służący do automatycznego tworzenia diagramów aktywności poddawanych następnie procesowi formalnej weryfikacji opisanej w rozdziale czwartym. Będzie on zakładał trzyetapowe postępowanie. Pierwszy etap to generowanie list operacji elementarnych wykonywanych przez system w związku z jego aktywnością wokół obsługi określonego przypadku użycia – we wszystkich możliwych ich wariantach alternatywnych. Drugi etap to generowanie zbioru zdarzeń (*events*) oraz ograniczeń stanu systemu (*constraints*), które mogą wpłynąć na aktywację wytypowanych akcji alternatywnych. Na ostatnim etapie będzie prowadzone łączenie list w diagramach aktywności. Proces łączenia będzie bazował głównie na informacji o relacjach pomiędzy przypadkami użycia i ich alternatywnymi wariantami realizacji w systemie. Materiałem źródłowym będą tu opisy przypadków użycia w postaci stabelaryzowanej oraz powiązania typu <<*extends*>> i <<*includes*>> pomiędzy przypadkami użycia na diagramach *use-case*.

Sam algorytm może wyglądać następująco:

1. Wygeneruj globalną listę aktywności (1) powiązaną z przypadkami użycia z diagramów *use-case* i każdy wpis z tej listy dopasuj do aktora.
2. Dla pozycji każdej listy (1) wygeneruj listę operacji elementarnych (2), związanych z obsługą akcji głównej dla przypadku użycia. Wyprowadź listy (2) na diagramy aktywności, głowy list łącząc z symbolami początku (*initial*), a końce – końca (*final*).
3. Dla każdej pozycji listy (1) wygeneruj zbiór list odpowiadających ewentualnym sekwencjom aktywności obsługującym akcje alternatywne (3), przewidziane w opisie tabelarycznym przypadków użycia.
4. Wygeneruj zbiór zdarzeń (4) oraz uwarunkowań w systemie (5), mających wpływ na wybór każdej z akcji alternatywnych (3) w czasie realizacji aktywności związanej z danym przypadkiem użycia.

5. Stwórz na bazie diagramów *use-case* listę powiązań pomiędzy przypadkami użycia (6) (powiązania są klasyfikowane w kategorii <<*includes*>> lub <<*extends*>>).
6. Na bazie listy (6) dokonaj dołączenia tworzonych na diagramach aktywności wszystkich wskazywanych przez nią list (2) do odpowiednich innych list aktywności (2). W przypadku wariantu powiązania <<*includes*>> przy łączeniu zastosuj symbol rozwidlenia (*fork*), ustalając lokalizację symbolu złączenia (*join*) na podstawie opisu tabelarycznego odpowiedniego przypadku użycia powiązanego z dołączaną listą (2). W przypadku wariantu powiązania <<*extends*>> przy łączeniu zastosuj symbol rozgałęzienia (*decision*), ustalając późniejsze połączenie dwóch przepływów aktywności (*flow*) na podstawie opisu tabelarycznego odpowiedniego przypadku użycia powiązanego z dołączaną listą (2).
7. Na bazie treści list (4) *oraz* (5) dokonaj dołączenia wszystkich list (3) do odpowiednich list (2) w pozycjach wskazanych przez opis tabelaryczny przypadków użycia. Przy łączeniu zastosuj symbol rozgałęzienia (*decision*). Następnie dla każdej listy (2) ustal lokalizację symbolu złączenia (*join*) – także na podstawie opisu tabelarycznego. Jeśli w opisie nie przewiduje się złączania – zakończ listę dołączaną (3) symbolem końca (*final*). Jeśli punkt dołączenia określono – połącz koniec listy dołączanej (3) z odpowiednią listą (2) za pomocą symbolu przepływu aktywności (*flow*).

Zaproponowany algorytm umożliwi także modelowanie przetwarzania współbieżnego na diagramach aktywności – jeżeli przy projektowaniu modelu wymagań *use-case* takie przetwarzanie dopuszczono. Poprawnie skonstruowany diagram aktywności powinien:

– uwzględniać wszystkie sytuacje, które mogą zaistnieć w trakcie realizacji aktywności systemu (przepływ aktywności nie może „utknąć”);
– posiadać zbalansowane rozgałęzienia i złączenia – liczba rozwidleń z gałęzi powinna być równa liczbie przepływów wchodzących do korespondującego złączenia;
– jego struktura musi uwzględnić zarówno relacje pomiędzy przypadkami użycia naniesione na diagramy *use-case* (<<*extends*>> <<*includes*>>), jak i akcje alternatywne dla realizacji przypadków użycia – opisane w postaci tabelarycznej;
– musi jasno określać punkty wejścia i wyjścia dla aktywności, każdorazowo jasno wyrażając kierunek, w jakim przepływ ma być kontynuowany.

Formalizm umożliwiający automatyczne przeprowadzenie weryfikacji diagramów byłby zatem bardzo pożądany. W obecnie stosowanych metodologiach diagramu weryfikacja prowadzona jest manualnie przez analityka. Może być trudna i czasochłonna. Nie daje także żadnej gwarancji uzyskania poprawnego wyniku.

4. Weryfikowanie funkcjonalności modelu

Modele uzyskane w wyniku zastosowania metodyki opisanej powyżej mogą zostać z powodzeniem zweryfikowane w sposób formalny. Metodyczne podejście do modelowa-

nia diagramów przypadków użycia oraz diagramów aktywności pozwala na zastosowanie wnioskowania dedukcyjnego, jako najbardziej przyjaznego analitykowi, przy czym proponuje się wnioskowanie metodą tablic semantycznych [1], stanowiącego ciekawą alternatywę dla podejścia tradycyjnego, związanego z metodą rezolucji. Ponadto, wykorzystana będzie logika temporalna [4], posiadająca ugruntowaną pozycję i z powodzeniem pozwalająca specyfikować następstwa i kolejności zdarzeń związane z własnościami żywotności i bezpieczeństwa systemów informatycznych.

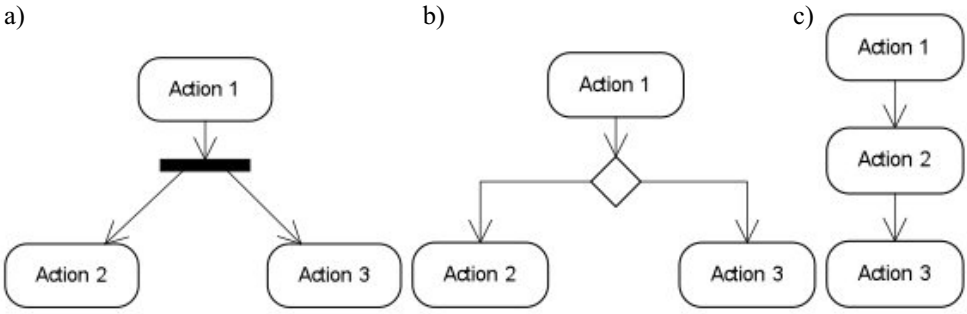
Sama proces wnioskowania będzie zakładał następujące kroki procedury wynikające z przyjętej metodologii postępowania. Modelowanie podstawowej funkcjonalności, znajdujące swoje odzwierciedlenie we wszystkich uzyskanych w ten sposób diagramach przypadków użycia. Uzyskane przypadki użycia UC_1, ..., UC_n tworzą całą przestrzeń funkcjonalną systemu. Każdy przypadek jest udokumentowany poprzez odpowiedni scenariusz obrazujący sekwencje czynności niezbędnych do realizacji takiego przypadku. Scenariusz ten jest podstawą do utworzenia diagramu czynności, zgodnie z metodologią opisaną w pracy. Dla każdego przypadku użycia tworzony jest przynajmniej jeden diagram aktywności, co powoduje, że uzyskujemy ogólnie A_1, ..., A_m diagramów aktywności. Dla każdego diagramu aktywności tworzona jest jego specyfikacja wyrażona w logice temporalnej. Specyfikację logiczną P_i każdego diagramu aktywności tworzy koniunkcja formuł logiki temporalnej uzyskanych z danego diagramu aktywności $P_i = p_{i,1} \wedge ... \wedge p_{i,k}$. Sposób generowania formuł z dowolnego diagramu aktywności zostanie omówiony w dalszej części. Należy zwrócić uwagę, że po wytworzeniu specyfikacji logicznej dla każdego diagramu aktywności uzyskujemy specyfikacje logiczną S całego systemu, systemu którego funkcjonalność została wcześniej wyrażona poszczególnymi przypadkami użycia UC_1, ..., UC_n. Specyfikację systemu stanowi koniunkcja wszystkich dostępnych formuł, a więc

$$S = \wedge_{i=1..m} P_i = \wedge_{i=1..m,\, j=1..k} p_{i,k} \tag{1}$$

Na podstawie tak uzyskanej specyfikacji możliwe będzie formalne weryfikowanie żądanych własności systemu. Sposób wnioskowania dedukcyjnego metodą tablic semantycznych zostanie przedstawiony w dalszej części.

Kluczowe znaczenie ma tutaj sygnalizowany wcześniej proces generowania formuł logiki temporalnej z poszczególnych diagramów aktywności. Ograniczmy się tutaj do tzw. najmniejszej logii temporalnej [3], operujących m.in. dwoma podstawowymi operatorami temporalnymi. Diagramy te stanowią jednak dobrą podstawę do ekstrakcji formuł logiki temporalnej. Wynika to miedzy innymi ze specyfiki samych diagramów, obrazujących w istocie przepływy sterowania pomiędzy poszczególnymi częściami systemu, opisanymi poprzez tory diagramu. Przepływy sterowania umożliwiają m.in. wybór aktywności przeznaczonych do wykonania po spełnieniu odpowiedniego warunku, albo też przykładowo równoczesną realizację aktywności poprzedzoną rozgałęzieniem sterowania. Wydaje się, że wszystkie przypadki przepływu sterowania na diagramach aktywności można wyodrębnić i zgrupować w pewnego rodzaju wzorce na podobieństwo podobnych wzorców

istniejących np. w procesach biznesowych. W przypadku diagramów aktywności wzorce takie wydają się jednak prostsze. W pracy zostaną rozważone trzy podstawowe przypadki, tak jak to przedstawiono na rysunku 2.



Rys. 2. Wzorce przepływów sterowania na diagramach aktywności:
a) rozgałęzienie; b) rozwidlenie; c) sekwencja

Oprócz tych przypadków można rozważać jeszcze inne, np. związane z pętla przepływu sterowania. Zostaną one jednak tutaj pominięte, tak jak inne jeszcze wątki charakterystyczne dla diagramów aktywności, np. pływające obiekty czy stany, które mogą być umieszczane na diagramach czynności. Wszystkie te wątki stanowić będą przedmiot osobnych, dalszych badań. Obecnie rozważane będą tylko przypadki przedstawione na rysunku 2. Pierwszy przypadek *sekwencji* dwóch aktywności zostanie opisany formułą:

$$Action1 \Rightarrow \Diamond Action2 \tag{2}$$

Z kolei *rozgałęzienie* wymaga wykluczenia realizacji niektórych aktywności:

$$Action1 \Rightarrow (\Diamond Action2 \wedge \sim\Diamond Action3) \vee (\sim\Diamond Action2 \wedge \Diamond Action3) \tag{3}$$

$$\Box\sim(Action2 \wedge Action3) \tag{4}$$

Wreszcie *rozwidlenie* sterowania i potencjalnie równoczesna realizacja aktywności:

$$Action1 \Rightarrow (\Diamond Action2 \wedge \Diamond Action3) \tag{5}$$

$$\Diamond(Action2 \wedge Action3) \tag{6}$$

Przedstawione wzorce elementarne stanowią także podstawę do budowy bardziej złożonych przypadków.

Po przedstawieniu sposobów pozyskiwania formuł logiki temporalnej, składających się na całą specyfikację systemu, zgodnie z formułą (1), należy jeszcze przedstawić sam sposób wnioskowania o własnościach systemu dedukcyjną metodą tablic semantycznych. Metoda zakłada dekompozycję pewnej formuły początkowej na formuły coraz prostsze, aż

do uzyskania formuł elementarnych w liściach drzewa wnioskowania. W ten sposób budowane jest drzewo wnioskowania, przy czym w jego korzeniu jest umieszczana główna formuła poddawana dekompozycji. W rozważanym tutaj procesie weryfikacji systemu informatycznego, zostanie więc utworzona formuła

$$S \Rightarrow Q \tag{7}$$

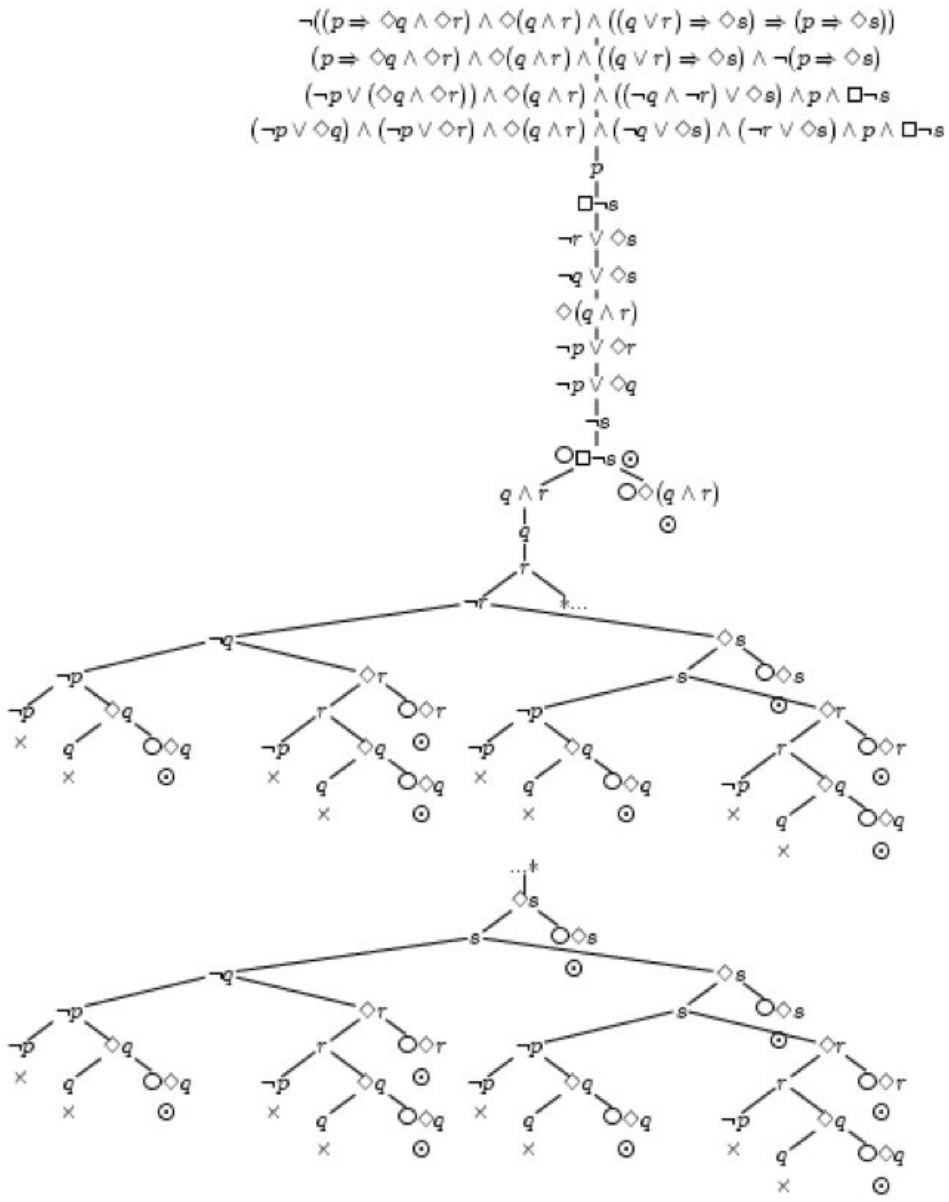
gdzie S stanowi specyfikację systemu zgodnie z formułą (1), a więc koniunkcję wszystkich formuł uzyskanych w trakcie procesu generowania formuł logiki temporalnej, natomiast Q to badana własność systemu. Należy jednak zwrócić uwagę, że w korzeniu drzewa wnioskowania umieszczane jest zaprzeczenie formuły (7), tzn. $\sim(S \Rightarrow Q)$. Takie postępowanie wraz z dekompozycją według dobrze zdefiniowanych reguł [1], prowadzi do domknięcia drzewa. Drzewo uznajemy za domknięte, jeżeli wszystkie jego gałęzie są domknięte. Dana gałąź jest domknięta, jeżeli w swojej gałęzi zawiera sprzeczności w odniesieniu do zdań elementarnych uzyskanych po procesie dekompozycji. Domknięcie drzewa świadczy o tym, że formuła umieszczona w korzeniu jest sprzeczna, tj. nie zawiera żadnego spełniającego wartościowania, co w konsekwencji prowadzi do stwierdzenia, że formuła początkowa, tj. przed umieszczeniem w korzeniu drzewa, w rozważanym tutaj przypadku formuła (7), jest zawsze prawdziwa. Takie postępowanie umożliwia przebadanie wszystkich własności systemu.

Ilustracja działania metody zostanie pokazana na przykładzie diagramu aktywności przedstawionego na rysunku 1. Rozważania zostaną jednak ograniczone do fragmentu tego diagramu, pominiemy ponadto jedno z rozgałęzień, które w tym fragmencie nie wnosi nowych elementów – wszystko to wynika z faktu, że rozważanie całej specyfikacji logicznej diagramu prowadziłoby do zbudowania formuły logicznej bardzo dużych rozmiarów, znacznie przekraczających rozmiary tej pracy. Należy jednak podkreślić, że tak długa formuła w niczym nie utrudnia samego procesu wnioskowania dedukcyjnego metodą tablic semantycznych, gdyż w każdym kroku procedury dekompozycyjnej uzyskiwane są formuły coraz prostsze i o mniejszej złożoności. Należy także zauważyć, że sam proces dekompozycji i wnioskowania metodą tablic semantycznych może zostać całkowicie zautomatyzowany. Dokonajmy odwzorowania wybranych aktywności diagramu do następujących zdań elementarnych:

p – *Check account balance*
q – *Take money from slot*
r – *Debit account*
s – *Show balance*

Dalej już dla uproszczenia będziemy się posługiwać tymi zdaniami. W drzewie wnioskowania przedstawionym na rysunku 3, w jego korzeniu, została umieszczona negacja formuły wyrażającej własności systemu oraz badaną własność. Drzewo zostało ukończone i jest zamknięte. Wszystkie gałęzie zawierają sprzeczności (x), niektóre z gałęzi nie

wymagają dalszego rozwijania, gdyż nie wprowadzają nowych zdań elementarnych. Tak więc formuła własnościowa jest prawdziwa, w tym konkretnym przypadku oznacza to, że aktywność początkowa rozważanego fragmentu diagramu aktywności (*Check account balance*) zawsze musi skutkować następującą po niej inną aktywnością (*Show balance*).



Rys. 3. Drzewo wnioskowania dla rozważanego przypadku

5. Podsumowanie

Artykuł przedstawia nowe podejście do formalnej weryfikacji scenariuszy przypadków użycia zapisywanych w formie diagramów czynności za pomocą podejścia dedukcyjnego metodą tablic semantycznych i logiki temporalnej. Zostały w nim zaproponowane metody szybkiego generowania diagramów czynności oraz metoda ekstrakcji formuł logiki temporalnej odpowiadających takim diagramom. Opisano następnie technikę przetwarzania otrzymanych formuł za pomocą metody tablic semantycznych. Załączono także przykład obrazujący działanie opisanej metody. Do najważniejszych jej zalet należą:

- możliwość weryfikacji scenariuszy przypadków użycia, a więc funkcjonalności systemu, na etapie jego modelowania,
- redukcja kosztów wytwarzania oprogramowania poprzez wcześniejszą detekcję błędów,
- możliwość automatycznego generowania formuł logiki temporalnej bezpośrednio ze scenariusza przypadku użycia.

Dalsze badania autorów są dwutorowe i na poziomie koncepcyjnym obejmują:

- zaproponowanie bardziej rozwiniętego odwzorowania związków pomiędzy przypadkami użycia (extends, includes, generalizacja) do formuł logicznych,
- prace w kierunku rozbudowy opisanych algorytmów w celu umożliwienia interpretacji bardziej złożonych przypadków przepływów sterowania na diagramie aktywności,
- prace w kierunku wytworzenia języków i opisywanych tymi językami transformacji, umożliwiających interaktywne modyfikowanie istniejących już przepływów sterowania (nie tylko na poziomie reprezentacji graficznej w modelu, ale także bezpośrednio pod postacią formuł).

Literatura

[1] D'Agostino M., Gabbay D.M., Hahnle R., Posegga J. (eds), *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.
[2] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.
[3] van Benthem J., *Temporal Logic.* Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4, 241–350, Clarendon Press 1993–95.
[4] Emerson E.A., *Temporal and Modal Logic*. Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics. MIT Press, Elsevier, 1990, 995–1072.
[5] Eshuis R., Wieringa R., *Tool Support for Verifying UML Activity Diagrams*. IEEE Transactions on Software Engineering, vol. 30, No. 7, 437–447.