SPECYFIKACJA UZALEŻNIEŃ CZASOWYCH I SYNTEZA OPROGRAMOWANIA SYSTEMÓW STERUJĄCYCH

Krzysztof Sacha, Politechnika Warszawska,
Instytut Automatyki i Informatyki Stosowanej
e-mail: k.sacha@ia.pw.edu.pl (grant 8 S505 010 07)

1 Wprowadzenie

Tradycyjny proces rozwoju oprogramowania systemów sterujących można podzielić na szereg etapów, w których powstają kolejne dokumenty projektowe. Opracowanie i zatwierdzenie wymaganych dokumentów jest warunkiem przejścia do następnego etapu.

Pierwszym etapem pracy jest analiza wymagań użytkownika. Wynikiem analizy jest dokument, nazywany specyfikacją wymagań (lub założeniami na oprogramowanie), który opisuje pożądane zachowanie programów. Specyfikację przedstawia się do akceptacji użytkownikowi, i po zatwierdzeniu przyjmuje za wzorzec dla projektowania i oceny programów. Słabą stroną tej procedury jest niska wiarygodność specyfikacji, wynikająca z dwóch przyczyn: niezdolności użytkownika do dokładnego sprecyzowania wymagań i niejednoznaczności języka naturalnego, prowadzącego do niejednoznaczności specyfikacji. W rezultacie, specyfikacja wymagań nie odzwierciedla dokładnie potrzeb użytkownika, co prowadzi do powstania systemu, który musi być natychmiast modyfikowany.

W etapach projektu i implementacji następuje określenie struktury i opracowanie kodu programów. Obydwa etapy są wykonywane ręcznie, a ich wykonanie wymaga twórczej pracy projektantów i programistów. Weryfikacja poprawności projektu następuje podczas testowania programu. Słabą stroną tej procedury jest fakt, że testowaniu mogą podlegać dopiero gotowe programy. Uniemożliwia to bieżące weryfikowanie kolejnych decyzji projektowych i szybką korekcję błędów. Poprawienie błędów projektu po zakończeniu implementacji jest zazwyczaj bardzo kosztowne, gdyż wymaga powtórzenia części projektu, programowania i testowania. Proces testowanie nie może też udowodnić poprawności programu.

Opracowanie i zainstalowanie programów w docelowym środowisku nie kończy prac projektowych. Programy sterujące żyją wraz z instalacją sterowaną, i podlegają zmianom, odzwierciedlającym zmiany w instalacji i zmiany wymagań użytkownika. W tej fazie, nazywanej konserwacją programu, kolejne zmiany wykonuje się z reguły wprost na tekstach programów, często bez adekwatnej aktualizacja dokumentacji projektowej. Pogarsza to coraz bardziej strukturę oprogramowania, i utrudnia kolejne modyfikacje.

Problemy tradycyjnego procesu rozwoju można podsumować następująco:

- niejednoznaczność specyfikacji wymagań,
- niski stopień automatyzacji podstawowych etapów pracy,
- niska wiarygodność i wysoki koszt testowania programów,
- bardzo wysoki koszt konserwacji.

Proces transformacyjnej implementacji. Specyfikacja wymagań powinna dokładnie i jednoznacznie opisywać pożądane zachowanie programu. Wszystkie dalsze działania zmierzają do implementacji tego zachowania. Specyfikacja opisuje więc w pełni se-

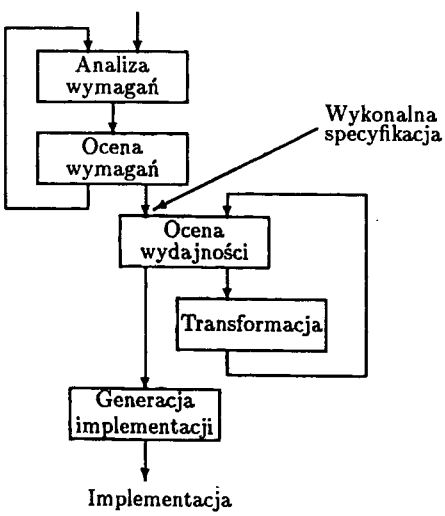
mantykę projektowanego systemu, a wszystkie dalsze zmiany dotyczą jedynie składni tego opisu. Te spostrzeżenia prowadzą do koncepcji transformacyjnej implementacji, która zakłada automatyczne przekształcenie specyfikacji wymagań w implementację programu.

Pierwszym etapem pracy jest formalne zapisanie wymaganego zachowania programu. Formalna specyfikacja wymagań jest wykonalna, i może być traktowana jako prototyp docelowego oprogramowania. Prototyp podlega badaniom i ocenie, i po zatwierdzeniu staje się obowiązującą definicją semantyki projektowanego oprogramowania. Różnica między prototypem, a implementacją polega na braku dopasowania struktury prototypu do struktury środowiska wykonawczego. Wynikiem tego niedopasowania jest nie osiągnięcie wymaganej wydajności, i oparcie wykonania na symulacji rzeczywistego obliczenia.

Dalsze prace stopniowo przekształcają strukturę specyfikacji, i dopasowują ją do ograniczeń środowiska wykonawczego. Proces ten przebiega przez kolejne wykonywanie transformacji, które zmieniają strukturę specyfikacji, lecz nie zmieniają jej zachowania [1]. Jeżeli wszystkie transformacje mają dowody poprawności, to proces testowania implementacji staje się zbędny. W każdym etapie transformacji aktualna specyfikacja oprogramowania może być traktowana jako kolejny prototyp systemu, którego funkcje, wydajność i koszt można badać eksperymentalnie.

Potencjalne zalety procesu transformacyjnej implementacji (rys. 1) dotyczą całego cyklu rozwoju oprogramowania. Jedynym etapem, który musi być wykonany ręcznie jest etap analizy i specyfikacji wymagań, którego produkt (specyfikacja) stanowi podstawowy dokument projektu. W miejsce etapów projektowania i implementacji pojawia się faza transformacji. Podczas konserwacji zmianom podlega również specyfikacja, która po wprowadzeniu zmian jest ponownie transformowana do implementacji.

Wdrożenie procesu transformacyjnej implementacji w praktyce projektowej wymaga opracowania metod i narzędzi wspomagających wykonanie, ocenę i transformację specyfikacji. Obecnie żaden z tych elementów nie jest jeszcze dostępny, a jedynie pewne elementy nowego podejścia występują w komercyjnej metodzie TAGS [2] i w eksperymentalnym systemie Model [3].



Rys. 1: Model transformacyjnej implementacji

Cel i zakres grantu. Wykonane w ramach grantu prace dotyczą oryginalnej metody specyfikacji i projektowania oprogramowania systemów sterujących, o nazwie Transnet. Metoda wpisuje się w proces transformacyjnej implementacji, i obejmuje swym zasięgiem etapy analizy wymagań i projektu wstępnego. Opracowana zgodnie z tą metodą specyfikacja jest wykonalna, i może być weryfikowana zarówno analitycznie, jak doświadczalnie, podczas symulacyjnego wykonania. Dalsze postępowanie zmierza do transformacyjnego przekształcenia specyfikacji w projekt programu.

Całość metody obejmuje następujące elementy [4]:

- metodę opisu funkcjonalnych i czasowych wymagań stawianych oprogramowaniu,
- zestaw technik automatycznej analizy specyfikacji,
- metodę symulacyjnego wykonanie specyfikacji,
- metodę automatycznej transformacji struktury specyfikacji.

Praktyczne wykorzystanie metody wymaga narzędzi programowych, wspomagających tworzenie, analizę, wykonanie i transformację specyfikacji (system CASE). Część tych narzędzi została już opracowana, i znajduje się w fazie testowania. Nie zostały rozpoczęte jedynie prace nad transformatorem struktury, którego implementacja jest planowana dopiero po zakończeniu prac nad opracowaniem pozostałych narzędzi.

2 Model algebraiczny

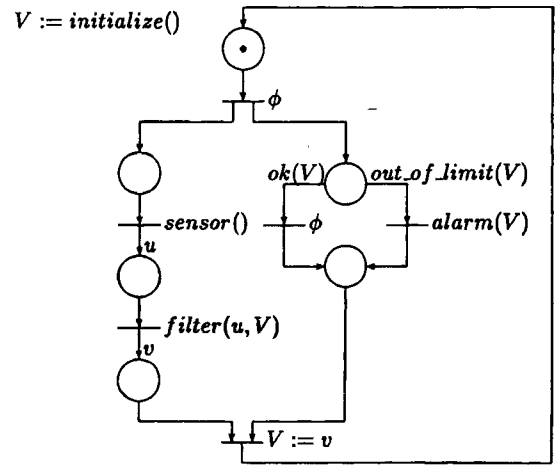
Formalne podstawy metody Transnet łączą elementy dwóch różnych modeli obliczeń komputerowych: procesy współbieżne i sieci Petri. Specyfikacja oprogramowania jest budowana jako zbiór procesów, z których każdy jest opisywany przez rozszerzoną sieć Petri. Poszczególne procesy mogą reprezentować obiekty programowe, takie jak zadania, bufory i zbiory danych, lub obiekty środowiskowe, takie jak urządzenia zewnętrzne, sprzęgi, ludzi, itd. Procesy mogą komunikować się ze sobą, wymieniając dane podczas symetrycznego i synchronicznego spotkania.

Klasyczne sieci Petri opisują przepływ sterowania, bez możliwości przedstawienia przetwarzania danych. Dla rozszerzenia siły wyrazu sieci w tym zakresie, zaproponowano w literaturze szereg rozszerzeń, które doprowadziły do powstania tzw. sieci wysokiego poziomu. Modele tej grupy opisują w istocie maszynę sterowaną przepływem danych, w której kolekcje znaczników przepływają przez sieć, przenosząc kolekcje jednostek danych. Model przyjęty w metodzie Transnet jest znacznie prostszy. Podstawowe rozszerzenia klasycznych sieci Petri związane z reprezentacją przetwarzania obejmują:

- zmienne, przypisane do miejsc sieci,
- funkcje, przypisane do tranzycji,
- warunki logiczne, przypisane do łuków łączących miejsca z tranzycjami.

Rozszerzenia te umożliwiają modelowanie podstawowych struktur przetwarzania, tzn. złożenia, rozgałęzienia warunkowego (alternatywy) i rozejścia równoległego.

Reprezentacja procesu. Specyfikacja oprogramowania składa się z cyklicznych procesów, powtarzanych permanentnie w rytmie określonym przez narzucone uzależnienia czasowe i występujące podczas wykonania specyfikacji zdarzenia. *Proces* definiuje się przez określenie przestrzeni stanów i funkcji przejścia, obliczającej stan następny na podstawie stanu bieżącego. Obliczenie procesu jest potencjalnie nieskończoną sekwencją kroków, w których funkcja przejścia określa następny stan procesu w jego przestrzeni stanów. Obliczony stan zastępuje stan poprzedni dokładnie na granicy między dwoma kolejnymi krokami procesu. W obrębie jednego kroku stan procesu nie ulega zmianom.



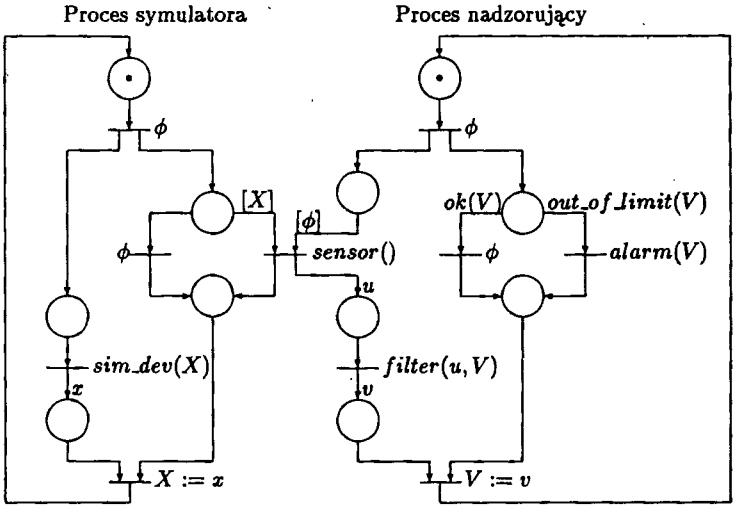
Rys. 2: Proces nadzorujący stan urządzenia

Funkcja przejścia procesu jest definiowana przez rozszerzoną sieć Petri: skierowany graf, zbudowany z dwóch rodzajów wierzchołków, nazywanych *miejscami* i *tranzycjami*, połączących za pomocą *łuków* w taki sposób, że żadne dwa miejsca ani dwie tranzycje nie są połączone bezpośrednio. Miejsca przedstawia się graficznie jako kółka, a tranzycje jako odcinki (rys. 2). W miejscach sieci mogą znajdować się *znaczniki*, rysowane jako kropki. Znaczniki mogą przemieszczać się między miejscami w rezultacie odpalania tranzycji. Tranzycja może odpalić jeżeli jest *wzbudzona*, tzn. jeżeli wszystkie jej miejsca wejściowe zawierają znaczniki. W chwili *odpalenia*, z każdego miejsca wejściowego usuwa się po jednym znaczniku, i w każdym miejscu wyjściowym dodaje się po jednym znaczniku. Wzbudzone tranzycje mogą odpalać w dowolnej kolejności.

Graf sieci reprezentujący pojedynczy proces jest cykliczny, przy czym wszystkie cykle (zamknięte drogi) w grafie przechodzą przez wyróżnione *miejsce terminalne*. Miejsce to nigdy nie zawiera znacznika jednocześnie z jakimkolwiek innym miejscem sieci procesu. Znakowanie sieci, w którym znacznik jest umieszczony w miejscu terminalnym odpowiada stanowi procesu między dwoma kolejnymi krokami wykonania. W początkowym znakowaniu sieci procesu znacznik znajduje się w miejscu terminalnym.

Z miejscami sieci procesu są związane *zmienne*, a z tranzycjami *funkcje*. Odpalenie tranzycji powoduje obliczenie związanej z nią funkcji, i podstawienie wartości na zmienne, przypisane do miejsc wyjściowych tranzycji. Każda zmienna jest związana z jednym miejscem procesu. W konsekwencji, wartość każdej zmiennej może być zmieniona tylko w jednym miejscu procesu. Ta *reguła pojedynczego podstawienia*, zaczerpnięta z teorii języków funkcjonalnych [5], usuwa możliwość powstania efektów ubocznych, wynikających z niedeterministycznej kolejności odpalania tranzycji. Warto zauważyć, że reguła ta niejawnie wyklucza istnienie lokalnych iteracji wewnątrz sieci procesu — jest to zgodne z założeniem, że wszystkie pętle grafu przechodzą przez miejsce terminalne.

Zmienne związane z miejscami nieterminalnymi mają zakres ograniczony do jednego cyklu procesu. Tylko zmienne związane z miejscem terminalnym zachowują określone wartości w następnym cyklu procesu. Wartości tych zmiennych, które będą nazywane dalej



Rys. 3: Współpraca symulatora urządzenia z procesem nadzorującym

zmiennymi terminalnymi, pamiętają historię wykonania procesu. Pozostałe zmienne służą jedynie komunikacji między funkcjami obliczanymi podczas wykonania procesu. Stan procesu jest charakteryzowany przez bieżące wartościowanie zmiennych terminalnych. Wartościowanie to pozostaje niezmienione w ciągu całego cyklu procesu. Początkowe wartości zmiennych terminalnych określa się przed rozpoczęciem wykonania sieci.

Funkcja związana z tranzycją może być:

- funkcją prostą — wbudowaną albo zdefiniowaną przez funkcję języka C, lub
- funkcją złożoną — zdefiniowaną przez oddzielną podsieć, zbudowaną z miejsc, łuków i tranzycji związanych z prostszymi funkcjami.

Drugi warunek określa możliwość hierarchicznej reprezentacji sieci, która umożliwia dekompozycję jednostek projektowych (funkcji) na jednostki prostsze.

Miejsce z wieloma łukami wyjściowymi odpowiada rozgałęzieniu warunkowemu. Jeżeli w takim miejscu znajduje się znacznik, to odpalić może tylko jedna z jego tranzycji wyjściowych. Wybór tej tranzycji zależy od wartości *warunków*, przypisanych do łuków wyjściowych miejsca. Na rys. 2 warunkami łuków są funkcje boolowskie *ok* i *out_of_limit*.

Bieżące położenia znacznika podczas wykonania sieci można interpretować jako stan licznika rozkazów, który wskazuje na aktualnie wykonywaną funkcję. Zgodnie z tą interpretacją, sieć Petri opisująca proces powinna być bezpieczna i zachowawcza.

Komunikacja procesów opiera się na dwukierunkowej, symetrycznej i synchronicznej wymianie danych podczas spotkania. Dwa procesy mogą współpracować ze sobą, wymieniając dane stanowiące argumenty wywołania spotkania. Model nie zawiera żadnych form współpracy poprzez zmienne wspólne ani buforowany przekaz danych.

Spotkanie między dwoma procesami modeluje *tranzycja wymiany*, należąca do obydwu współpracujących procesów. Tranzycja taka ma dwa łuki wejściowe i dwa łuki wyjściowe, przy czym jedna para łuków (wejściowy i wyjściowy łuk tranzycji) należy do jednego procesu, a druga para do drugiego. Związana z tranzycją *funkcja wymiany* zwraca w każdym procesie wartości argumentów wywołania tej funkcji w drugim procesie.

Przykład współpracy procesów pokazuje rys. 3. Proces symulatora urządzenia przekazuje wartość zmiennej X do procesu rejestrującego dane, poprzez tranzycję wymiany o nazwie *sensor*. Żadne użyteczne dane nie są przekazywane w kierunku przeciwnym.

Warto zauważyć, że tranzycja wymiany jest wykonywana przez proces symulatora alternatywnie z inną tranzycją, związaną z funkcją pustą. Tego rodzaju konstrukcja symuluje nieblokujący przekaz danych między procesami: proces symulatora nigdy nie zostanie zatrzymany przez proces rejestrujący. Aby zachować semantykę tej konstrukcji, program symulujący wykonanie sieci w systemie Transnet zawsze próbuje wykonać tranzycję wymiany przed tranzycją alternatywną.

Hierarchiczna struktura modelu. Graficzna reprezentacja sieci Petri jest czytelna i wygodna tylko dla małych sieci, które można przedstawić na jednym rysunku. Stąd, bardzo ważna jest możliwość hierarchizacji modelu, i opisania go w formie wielowarstwowej struktury sieci przedstawiających coraz więcej szczegółów.

W metodzie Transnet diagram wysokiego poziomu przedstawia cały proces, którego tranzycje są związane ze złożonymi funkcjami. Każdy niższy poziom przedstawia rozwinięcie jednej tranzycji i związanej z nią funkcji w podsieć, obrazującą wymagane przetwarzanie. Dekompozycję można kontynuować w dół aż do poziomu prostych funkcji, tzn. funkcji standardowych lub funkcji implementowanych w języku programowania. Semantyka rozwijania tranzycji w podsieci jest zdefiniowana formalnie.

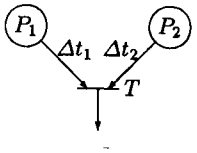
3 Reprezentacja czasu

Rozszerzone sieci Petri, opisane w poprzednim rozdziale, umożliwiają opis przepływu sterowania i przetwarzania danych w specyfikowanym oprogramowaniu. Realistyczne modelowanie zachowania systemów czasu rzeczywistego wymaga również narzędzi do opisywania uzależnień czasowych. W metodzie Transnet wymagania czasowe są wyrażane przez stałe, przypisane do łuków prowadzących z miejsc do tranzycji sieci.

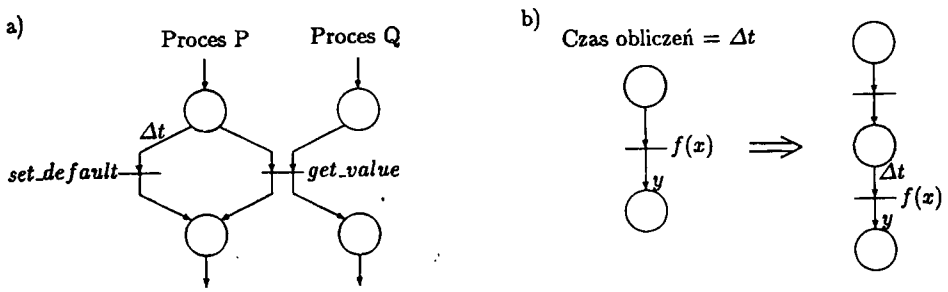
Definicja modelu. Przypisana do łuku stała Δt określa minimalny czas, przez który znacznik musi przebywać w miejscu wejściowym, zanim będzie mógł wzbudzić tranzycję do której ten łuk dochodzi. Łuk sieci bez jawnie podanej wartości czasu ma przypisaną domyślną wartość równą zeru.

Przyjęty model czasu jest jednoznaczny, i umożliwia wyspecyfikowanie przypadków, w których czas odpalenia tranzycji (i obliczenia funkcji) zależy w różny sposób od różnych działań wykonanych wcześniej. Czas odpalenia tranzycji może być odrębnie odniesiony do chwil pojawienia się znaczników w jej miejscach wejściowych (rys. 4).

Zastosowanie tego modelu czasu w metodzie Transnet może być dwojakie. Zastosowanie bezpośrednie polega na określeniu przeterminowania (*timeout*) operacji potecjalnie nieskończonych. Na przykład, proces P, którego część jest pokazana na rys. 5a, będzie oczekiwał na spotkanie z procesem Q nie dłużej niż przez czas Δt. Jeżeli tranzycja wy-



Rys. 4: Stałe czasowe łuków



Rys. 5: Zastosowanie stałych czasowych: przeterminowanie operacji (a); symulacja czasu obliczenia funkcji (b)

miany *get_value* nie odpali w tym czasie, to zostanie wzbudzona tranzycja *set_default*, której odpalenie umożliwi kontynuację wykonania procesu.

Inne zastosowanie tego samego mechanizmu umożliwia modelowanie czasu obliczenia funkcji związanej z tranzycją. Łatwo zauważyć, że czas Δt takiego obliczenia może być modelowany przez konstrukcję sieci pokazaną na rys. 5b.

Wykonanie sieci czasowej. Wprowadzenie stałych czasowych do modelu sieci prowadzi do zmiany pojęcia "stanu" sieci, który nie może być opisywany tylko przez aktualne znakowanie miejsc. Rozważmy łuk prowadzący od miejsca p do tranzycji t, związany ze stałą czasową Δt. Łuk ten będzie nazywany:

- **zamkniętym**, jeżeli żadne znaczniki nie znajdują się w miejscu p,
- **aktywnym**, jeżeli w miejscu p znajduje się znacznik,
- **otwartym**, jeżeli znacznik przebywa w miejscu p co najmniej przez czas Δt.

Stan sieci Petri z przeterminowaniem łuków jest parą $S =< M, E >$, złożoną ze znakowania M i wektora rozkładu czasów E. Wektor E zawiera bieżące wartości czasu, pozostające do upłynięcia przeterminowania wszystkich aktywnych łuków sieci. Rozmiar wektora E nie jest stały, lecz zmienia się wraz ze zmianami znakowania sieci — każdy element wektora opisuje jeden aktywny łuk, i zawiera długość okresu, pozostającego do otwarcia łuku.

Wykonanie sieci Petri z przeterminowaniem łuków polega na odliczaniu odcinków czasu, zapisanych w wektorze E, i odpalaniu tranzycji wzbudzonych w znakowaniu M. W chwili odpalenia tranzycji następuje przeniesienie znaczników i przesunięcie wszystkich czasów otwarcia łuków. Tranzycja t może odpalić w stanie $S =< M, E >$ jeżeli:

- jest wzbudzona w znaczeniu przyjętym w teorii sieci Petri, i
- wszystkie jej łuki wejściowe są otwarte.

Niech $S =< M, E >$ będzie stanem sieci Petri z przeterminowaniami. Obliczenie następnego stanu $S' =< M', E' >$ wymaga rozważenia dwóch przypadków.

1. Przypuśćmy, że żadna tranzycja nie jest wzbudzona w stanie S, i niech Δt będzie najkrótszym, niezerowym czasem zapisanym w wektorze E. Stan następny S' oblicza się ze stanu S następująco:
   a) $M' = M$,
   b) E' zawiera te same elementy co E, i dla każdego elementu e:

$$E'(e) = \begin{cases} E(e) - \Delta t & \text{jeżeli } E(e) \neq 0 \\ 0 & \text{w przypadku przeciwnym} \end{cases}$$

2. Przypuśćmy, że tranzycja t jest wzbudzona w stanie S. Stan następny S' osiągnięty przez odpalenie tranzycji t oblicza się następująco:
   a) M' powstaje z M przez usunięcie po jednym znaczniku z każdego miejsca wejściowego tranzycji t, i umieszczenie po jednym znaczniku w każdym miejscu wyjściowym.
   b) E' powstaje z E przez usunięcie elementów odpowiadających łukom zamkniętym przez usunięcie znaczników, i wprowadzenie stałych czasowych odpowiadających łukom otwartym przez znaczniki umieszczone podczas odpalania tranzycji t.

Właściwości sieci czasowej. Wprowadzone pojęcie stanu $S =< M, E >$ umożliwia zdefiniowanie zmodyfikowanego drzewa osiągalności, analogicznego do drzewa osiągalności klasycznej sieci Petri. Węzły zmodyfikowanego drzewa osiągalności odpowiadają stanom sieci, tzn. parom złożonym ze znakowania M i wektora czasów E. Łuki drzewa odpowiadają odpaleniom tranzycji; przesunięcia czasu (krok 1 wykonania) nie są pokazywane jawnie, i nie powodują rozrastania się drzewa.

Podstawowe właściwości wprowadzonego modelu czasu określają następujące twierdzenia.

- **Twierdzenie 1.** Czasowa sieć Petri z przeterminowaniami łuków jest ograniczona wtedy i tylko wtedy, gdy znakowana sieć Petri jest ograniczona.

- **Twierdzenie 2.** Specyfikacja zapisana w postaci rozszerzonej sieci Petri jest ograniczona wtedy i tylko wtedy, gdy zawarta w jej definicji sieć Petri jest oograniczona.

Ponieważ sieć poprawnie zbudowanej specyfikacji oprogramowania jest bezpieczna, więc twierdzenia te dowodzą, że właściwości specyfikacji można efektywnie badać przez analizę jej drzewa osiągalności.

4 Schemat postępowania

Zastosowanie metody Transnet obejmuje fazę specyfikacji i makietowania wymagań oraz transformacji struktury do postaci projektu wstępnego. Schemat postępowania można opisać w postaci sekwencji pięciu kroków.

1. Określenie struktury problemu. W pierwszym kroku należy przeprowadzić analizę rozwiązywanego problemu, wyodrębnić obiekty występujące konstruowanym modelu rzeczywistości i ustalić komunikaty przekazywane między tymi obiektami. Obiekty są reprezentowane przez współbieżne procesy, których działanie opisuje się za pomocą rozszerzonych sieci Petri. Procesy współpracują ze sobą przez synchroniczną wymianę danych. Rezultatem pierwszego kroku jest wstępna specyfikacja, w której tranzycje są związane ze złożonymi funkcjami. Zarówno funkcje jak typy danych są zdefiniowane wstępnie, w postaci nieformalnego opisu słownego.

2. Rozwinięcie specyfikacji. Typy danych wprowadzone w pierwszym kroku definiuje się jako zbiory wartości zbudowane ze zbiorów elementarnych i operacji zbiorowych: wyliczenia, sumy i produktu kartezjańskiego. Określenie struktury danych umożliwia dekompozycję funkcji, i przedstawienie ich w postaci podsieci, z tranzycjami związanymi z funkcjami prostymi. Równocześnie z rozwijaniem specyfikacji definiuje się charakterystyki czasowe:

- oszacowania czasu obliczania funkcji,
- ograniczenia narzucone na czas wykonanie operacji.

Oszacowania czasów obliczenia funkcji przypisuje się do funkcji na najniższych poziomach hierarchii definicji sieci. Ograniczenia narzucone na czas wykonania operacji przypisuje się do tranzycji na najwyższych poziomach hierarchii sieci oraz do łuków, występujących w konstrukcjach warunkowych operacji potencjalnie nieskończonych.

Rezultatem drugiego kroku jest szczegółowa specyfikacja, złożona z hierarchicznej sieci Petri z dokładnie zdefiniowanymi funkcjami, zmiennymi, typami danych i ograniczeniami czasowymi.

3. Analiza sieci. Podstawową metodą analizy jest badanie drzewa osiągalności sieci. Celem budowy i analizy drzewa jest wykrycie zakleszczeń, i zbadanie osiągalności wskazanych przez użytkownika stanów. Analiza osiągalności może wykazać zarówno możliwość dojścia wykonania do stanu pożądanego, jak i możliwość wystąpienia stanu niepożądanego (niebezpiecznego). Dalszym celem analizy jest kontrola spójności definicji stałych czasowych na różnych poziomach hierarchii sieci. Suma oszacowań czasów wykonania funkcji w podsieciach niższego poziomu nie może przekraczać ograniczenia czasu wykonania tranzycji odpowiadającej tej podsieci na wyższym poziomie.

4. Makietowanie i ocena specyfikacji. Specyfikacja zapisana w postaci rozszerzonej sieci Petri może być wykonana przez symulację wykonania sieci. Rezultatem tego kroku są raporty symulacji, pokazujące ślad wykonania (ciąg wartości zmiennych) wraz z uzyskanymi czasami obliczenia. Podczas symulacji można również ocenić czas wykonania sekwencji działań obejmujących odpalenie wielu tranzycji, należących do różnych procesów.

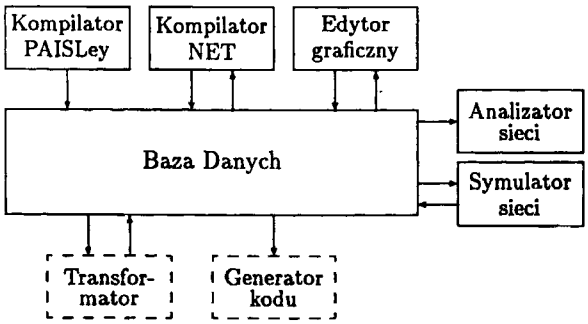
5. Transformacja specyfikacji. Akceptacja zachowania specyfikacji kończy etap analizy i specyfikacji wymagań, i umożliwia przejście do działań, których celem jest otrzymanie projektu programów. Metoda Transnet umożliwia automatyczną transformację sieci, i zbliżenie jej struktury do wymagań środowiska wykonawczego. Zachowanie i właściwości specyfikacji otrzymanej w wyniku transformacji można ponownie badać zarówno analitycznie (cofnięcie do etapu 3), jak i przez makietowanie (cofnięcie do etapu 4).

5 Narzędzia wspomagające

Praktyczne wykorzystanie metody wymaga odpowiednich narzędzi programowych, tworzących system wspomagający procesy analizy wymagań i projektowania oprogramowania (system CASE). Część tych narzędzi została już opracowana, część znajduje się w fazie realizacji. Najważniejszymi narzędziami wspomagającymi są (rys. 6):

- Edytory specyfikacji, umożliwiające wprowdzenie opisu specyfikacji wymagań, i zapisanie jej w wewnętrznym formacie rozszerzonej sieci Petri.
- Analizator specyfikacji, umożliwiający budowę drzewa osiągalności sieci i badanie jej podstawowych właściwości.
- Symulator sieci, pozwalający na wykonanie sieci, i zarejestrowanie wyników w zadanych punktach kontrolnych.
- Transformator, umożliwiający automatyczne wkonanie wskazanych przez użytkownika transformacji sieci.

Wewnętrzną reprezentacją specyfikacji jest zawsze model rozszerzonej sieci Petri. Zewnętrznymi językami specyfikacji mogą być natomiast język sieci Petri lub opracowany w Bell AT&T język funkcjonalny PAISLey [6]. Algorytm translacji programów języka PAISLey do postaci rozszerzonej sieci Petri jest opisany w [7].



Rys. 6: Struktura systemu wspomagającego metodę Transnet

Podstawową metodą analizy specyfikacji jest badanie drzewa osiągalności sieci. Program analizatora buduje drzewo, i automatycznie sprawdza warunek bezpieczeństwa i braku zakleszczeń sieci. Dalsze możliwości analizy obejmują przeglądanie drzewa, i sprawdzenie osiągalności wskazanych stanów oraz obliczenie minimalnych i maksymalnych czasów przejścia po określonej drodze w grafie sieci. Analiza sieci z warunkami łuków wymaga zbudowania odrębnego drzewa dla każdego interesującego układu wartości warunków.

Symulacja wykonania sieci rozpoczyna się w zadanym przez użtkownika stanie początkowym, i przebiega przez kolejne odpalanie wzbudzonych tranzycji i obliczanie związanych z tranzycjami funkcji. Efektem symulacji jest raport, podający wartości zmiennych i czasu obliczenia we wskazanych punktach kontrolnych. Punkty kontrolne można definiować przez podanie częściowego znakowania sieci, wskazanie odpalających tranzycji lub określenie konkretnych wartości czasu.

Szczególną rolę w systemie wspomagającym będzie pełnił transformator struktury, który umożliwi automatyczne przekształcanie rozszerzonej sieci Petri. Opracowanie tego narzędzia wymaga jednak zebrania pewnych doświadczeń z eksploatacją symulatora, i zostało odłożone na później. Przykład transformacji specyfikacji jest podany w [7].

Literatura

[1] P. Zave, The Operational Versus the Conventional Approach to Software Development, *Commun. ACM*, vol. 27, no 2, pp. 104-118, 1984.

[2] G. E. Sievert, T. A. Mizell, Specification-based Software Engineering with TAGS, *Computer*, vol. 18, no 4, pp. 56-65, 1985.

[3] J. S. Tseng, B. Szymanski, Y. Shi, N. S. Prywes, Real-time Software Life Cycle with the Model System, *IEEE Trans. Software Eng.*, vol. SE-12, no 2, pp. 192-197, 1986.

[4] K. Sacha, Real-Time Software Specification and Validation with Transnet, *Real-Time Systems Journal*, vol. 6, pp. 153-172, 1994.

[5] W. B. Ackerman, "Data Flow Languages", *Computer*, vol. 15, pp. 15-25, Feb. 1982.

[6] P.Zave and W. Schell, "Salient Features of an Executable Specification Language and its Environment", *IEEE Trans. Software Eng.*, vol. SE.-12, no 2, pp. 312-325, 1986.

[7] K. Sacha, Transformational Implementation of PAISLey Specifications Using Petri Nets, *Software Eng. Journal*, vol. 7, no 3, pp. 191-204, 1992.