STUDIA INFORMATICA 2013
Volume 34 Number 2B (112)

Łukasz WYCIŚLIK, Paweł WIEJAK
Politechnika Śląska, Instytut Informatyki

IMPLEMENTACJA BIBLIOTEKI MODELOWANIA UKŁADÓW ZDARZEŃ DYSKRETNYCH OPARTEJ NA JĘZYKU CSL, Z WYKORZYSTANIEM PLATFORMY .NET.

Streszczenie. Artykuł przedstawia bibliotekę służącą do modelowania układów zdarzeń dyskretnych według koncepcji wyboru działania, opartą na języku CSL. Cechą charakterystyczną rozwiązania jest zastosowanie platformy .NET oraz języka C#. Duża popularność tych technologii sprawia, że budowa modeli jest możliwa już przy minimalnym nakładzie związanym z zapoznaniem się z API samej biblioteki.

Słowa kluczowe: układy zdarzeń dyskretnych, CSL, modelowanie matematyczne, symulacja

IMPLEMENTATION OF DISCRETE EVENT SIMULATION LIBRARY BASED ON CSL LANGUAGE USING .NET PLATFORM

Summary. The article presents the library used to discrete events systems modeling based on CSL language. The key point of this implementation is involving of .NET platform and C# language. Big popularity of these technologies makes that building of models is possible with minimal effort related to study of library API.

Keywords: discrete event systems, CSL, mathematical modeling, simulation

1. Układy zdarzeń dyskretnych

Modelowaniem nazywamy czynność, która polega na badaniu modeli stanowiących przybliżenie rzeczywistości. Ogólnie rzecz ujmując, proces modelowania traktuje się jako proces odbioru i odpowiedniego przetwarzania informacji dotyczących danego wycinka rzeczywistości [1]. Najważniejszym celem modelowania oraz budowania modeli jest symulacja. Pojęcie to oznacza technikę pozwalającą na imitowanie działania całego systemu lub naśla-

dowania pewnej sytuacji poprzez użycie odpowiednich modeli lub urządzeń, w celu zdobycia informacji lub w celach dydaktycznych.

W zależności od charakteru modelowanego systemu oraz potrzeb, które powinien spełniać skonstruowany model, stosuje się różne podejścia do jego budowy. W przypadku układów ciągłych mogą to być na przykład modele matematyczne, opisujące przebieg pojedynczego eksperymentu, a komputery służyć będą do realizacji obliczeń numerycznych. Istnieją jednak sytuacje, kiedy przeprowadzonej symulacji nie można odtworzyć w świecie materialnym. Dzieje się tak w przypadku, gdy model/zjawisko charakteryzuje się dużymi zmianami w odpowiedziach przy niewielkich zmianach wymuszeń (przykładem może być rzut kostką do gry). Mówi się wtedy o zdarzeniach losowych. Czasami jednak interesujące są obserwacje układów składających się z następujących po sobie zdarzeń losowych. W takich przypadkach symulację pojedynczych eksperymentów sprowadza się do generowania pseudolosowych wyników, zgodnych z zaobserwowanym rozkładem gęstości prawdopodobieństwa. Modele takie nazywa się układami zdarzeń dyskretnych lub systemami masowej obsługi.

Układy zdarzeń dyskretnych to układy, dla których jakakolwiek zmiana stanu układu zachodzi w dyskretnym momencie czasu [2]. Momenty te mogą być losowe bądź nie. Stan układu charakteryzuje zbiór wartości zmiennych opisujących dany układ, a samą zmianę stanu zazwyczaj nazywa się zdarzeniem. Stąd układy takie charakteryzują dwie zasadnicze cechy:

- zdarzenia, czyli zmiany stanu układu,
- działania podejmowane w wyniku tych zdarzeń.

Modelowanie układów zdarzeń dyskretnych znajduje zastosowanie w skalach zarówno mikro, jak i makro. W informatyce stosuję się je szeroko np. do badania protokołów sieciowych różnych warstw oraz przewidywania obciążenia różnych węzłów obliczeniowych (serwerów baz danych, serwerów aplikacyjnych itp.). W skali makro zdarzenia mogą być reprezentowane także przez czynnik ludzki i wtedy można badać np. średni czas oczekiwania klienta na obsługę w urzędach, szpitalach itp.

Istnieje wiele różnych podejść do modelowania układów zdarzeń dyskretnych:

- planowanie zdarzeń,
- koncepcja wyboru działania,
- interakcja procesów.

W artykule zostanie zaprezentowane podejście drugie, czyli koncepcja wyboru działania [2]. Metoda ta wymaga zidentyfikowania w modelowanym systemie wszystkich zdarzeń czasowych (czyli takich, które występują niezależnie od stanu modelu – np. nadejście zgłoszenia) oraz warunkowych (czyli takich, które następują tylko w przypadku osiągnięcia danego stanu przez model – np. rozpoczęcie obsługi zgłoszenia w sytuacji, gdy stanowisko obsługi jest puste, a przed stanowiskiem znajduje się zgłoszenie) i polega na bieżącym sprawdzaniu, czy w danej chwili zachodzi jakieś zdarzenie czasowe bądź warunkowe. Jeśli zacho-

dzi, to realizowana jest symulacja jego obsługi. Istotnym elementem są tutaj specjalne atrybuty, zwane zegarami, które przypisuje się do każdego ze zdarzeń czasowych. Ich wartość wskazuje, czy w danym momencie zdarzenie zachodzi czy nie. Ujemna wartość zegara informuje, że zdarzenie miało już miejsce, zaś dodatnia, że zdarzenie dopiero wystąpi. Gdy wartość zegara dla danego zdarzenia jest równa zeru oraz zegar ten jest włączony, oznacza to, iż zdarzenie właśnie zachodzi. Do modelowania upływu czasu wykorzystywana jest specjalna funkcja, która wyszukuje wśród wszystkich włączonych zegarów taki, który ma najmniejszą nieujemną wartość, a następnie dekrementuje o tą wartość wszystkie zegary, niezależnie od tego, czy są włączone czy nie.

2. Język CSL

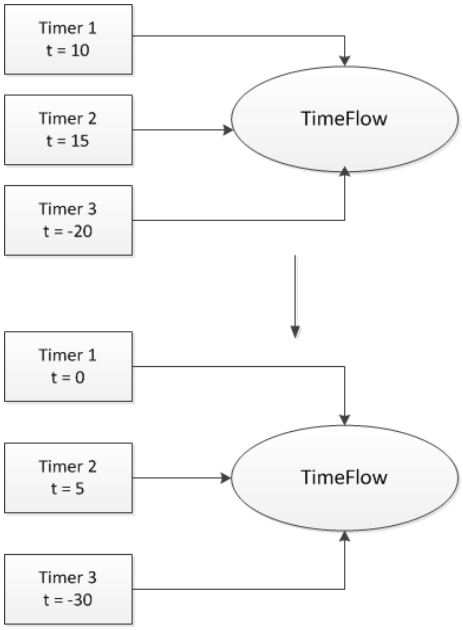
Język CSL jest rozwijany od wielu lat. W latach 70. XX w. popularna była wersja, w której język oparty był na składni języka Fortran (programy w języku CSL były preprocesowane do kodu w Fortanie). W latach 90. powstało autorskie rozwiązanie – CSL++ – wykorzystujące zalety programowania obiektowego, oparte na języku C++ [3]. W XXI w. powstała pierwsza wersja CSL – nazwana CSL# – wykorzystująca zalety środowiska uruchomieniowego .NET Framework 2.0 [4]. Ta wersja CSL, oprócz zastosowania w dydaktyce (podobnie jak CSL++), okazała się użyteczna w modelowaniu działania protokołów w sieciach komputerowych. Opisywana w niniejszym artykule najnowsza wersja CSL wykorzystuje nowe mechanizmy .NET Framework 4.0, np. wyrażenia lambda, czy składnię LINQ to Object.

Aby w pełni wspierać modelowanie układów zdarzeń dyskretnych według koncepcji wyboru działania, potrzebnych jest wiele składowych opisanych w poniższych rozdziałach. Zostały one pogrupowane w zbliżone obszary merytoryczne.

2.1. Modelowanie upływu czasu

Bardzo istotną rolę w symulacji odgrywa sposób obsługi upływu czasu. Każde zgłoszenie w systemie ma atrybut zegara, który może oznaczać czas, jaki pozostał do lub upłynął od wystąpienia zdarzenia. Zegar może też nie być związany z żadnym zdarzeniem, a po prostu odmierzać czas, jaki upłynął od pewnego momentu. Każdy zegar jest rejestrowany w funkcji upływu czasu, która dba o jego symulację i odmierzanie według wcześniej przedstawionego algorytmu.

Na rysunku 1 widać sposób, w jaki funkcja TimeFlow odmierza czas. Znajduje ona zegar o najmniejszej nieujemnej wartości i dekrementuje o nią wartości wszystkich zegarów.



Rys. 1. Funkcja upływu czasu
Fig. 1. Time flow function

2.2. Grupy – zbiory i kolejki

Zgłoszenia obsługiwane przez system trzeba w pewien sposób gromadzić i przetwarzać. Przed stanowiskami obsługi mogą tworzyć się kolejki lub zbiory zgłoszeń oczekujących na obsługę. Biblioteka zapewnia dwie struktury: zbiory oraz kolejki, czyli grupy uporządkowane oraz nieuporządkowane. Dla każdej z grup dostępnych jest wiele klas operacji, które można wykonać, takich jak:

- bezwarunkowe oraz warunkowe działania na grupach,
- badanie stanu grup – proste oraz złożone,
- warunkowy wybór elementów grupy,
- iteracje z wykorzystaniem grup.

2.3. Generatory liczb pseudolosowych o zadanych funkcjach gęstości rozkładu prawdopodobieństwa

Chcąc zamodelować jakiś układ, trzeba zapewnić poprawną losowość, jaka występuje w rzeczywistym środowisku. Mimo że programowo można jedynie generować liczby pseudolosowe, to jest to wystarczające do realizacji wiarygodnych badań modelowych. Służą do tego generatory różnych rozkładów:

- równomierny – o zadanym przedziale,
- wykładniczy – o zadanej wartości średniej,
- normalny – o zadanej wartości średniej oraz odchyleniu standardowym,
- Poissona – o zadanej wartości średniej.

2.4. Statystyki

Wcześniej wymienione funkcjonalności, wraz z bogactwem konstrukcji i standardowych bibliotek platformy .NET, są w zasadzie wystarczające do budowy funkcjonalnie pełnych modeli układów zdarzeń dyskretnych. Jednak celem modelowania najczęściej jest obserwowanie pewnych parametrów systemu (istotne mogą być zarówno ich zmiany w czasie symulacji, jak również wartości końcowe).

Służą do tego histogramy, klasy statystyk ciągłych (gdzie wraz ze zmianą wartości parametru odnotowuje się czas, kiedy to nastąpiło) oraz dyskretnych (kiedy gromadzi się jedynie wartości pewnych parametrów, np. w celu późniejszego obliczenia ich wartości średniej). Ze względu na potrzeby, zaimplementowane zostały różne klasy zapewniające gromadzenie danych, różniące się sposobem interpretacji oraz obliczania różnych statystyk.

3. Implementacja

Podział poszczególnych modułów implementacji został dokonany na podstawie wymienionych w rozdziale drugim odpowiedzialności. Każdy pakiet został zawarty w osobnej przestrzeni nazw, co znacznie ułatwia ich używanie. Należą do nich następujące przestrzenie:

- Groups – zawiera klasy Set oraz Queue, reprezentujące kolejno zbiór oraz kolejkę, implementujące abstrakcyjną klasę bazową PureGroup,
- Statistics – zawiera klasy Hist oraz Histogram, które reprezentują dwie różne implementacje histogramu klasy statystyk ciągłych: StatCTrap oraz StatCRect, różniące się sposobem obliczania statystyk (metodą kwadratów i trapezów) oraz klasę StatD, która reprezentuje statystykę dyskretną,
- Time – klasy Timer oraz Time, służące do modelowania upływu czasu,
- Generators – klasa Random, reprezentująca rozkłady wykładnicze, normalne oraz Poissona.

Implementacja biblioteki w języku C# wzorowana była na implementacji w języku C++. Jednak języki te wiele różni, m.in. w C# brak jest wskaźników na funkcje (wskaźniki te były wykorzystywane w źródłowej implementacji przez funkcję TimeFlow). W implementowanej bibliotece zostało to zastąpione delegatami. W przypadku funkcjonalności związanych z grupami, użytkownik może zaimplementować własną metodę, zgodną z sygnaturą oraz typami delegatów, a następnie przekazać ją do poszczególnych już zaimplementowanych metod klas kolejek oraz zbiorów. Można dzięki temu wykonywać różne złożone operacje filtrowania i porządkowania, co znajduje zastosowanie np. w przypadku potrzeby wprowadzenia regulaminów kolejkowania.

4. Przykładowa implementacja

```
for(…) {
……
    if ((StanObsl1 == -1) && kolejka.Find(ref el,
    CSL.Groups.FindParameters.FIRST))
    {
        kolejka.From(el);
        DlKol--;
        StatDlKol.Add(DlKol, -ZegarStat.t);
        StanObsl1 = (int)el;
        KonObs1.t = GenCzasObsl1.Get();
        KonObs1.SetOn();
    }
    CSL.Time.Time.TimeFlow();
}
```

Powyższy fragment programu stanowi główną pętlę symulacji – warunek zakończenia nie jest tutaj istotny, może być zależny od liczby obsłużonych zgłoszeń bądź bezpośrednio od czasu trwania procesu symulacji. W pętli tej sprawdzanych jest wiele warunków dotyczących zdarzeń czasowych i warunkowych – w przykładzie zamieszono obsługę zdarzenia warunkowego, polegającą na sprawdzeniu, czy dane stanowisko obsługi jest puste (wartość -1) oraz czy jest kolejka przed stanowiskiem. Jeśli tak, pobierany jest pierwszy element tej kolejki. Inkrementowana jest zmienna reprezentująca aktualną długość kolejki, a jej wartość dodawana jest do statystki ciągłej, celem obliczenia średniej długości kolejki po całej symulacji. Stanowisko obsługi zajmowane jest przez pobrany z kolejki element, a następnie generowany jest całkowity czas obsługi tego zgłoszenia na stanowisku i jego wartość przypisywana jest do zegara związanego z obsługą na tym stanowisku. Na końcu zegar jest załączany tak, aby funkcja upływu czasu zidentyfikowała moment jego obsługi. Ponieważ było to już ostatnie zdarzenie na końcu głównej pętli symulacji, wywoływana jest metoda TimeFlow(), ponieważ spowoduje ona symulację upływu czasu. Przy kolejnym obiegu pętli mogą w ten sposób zostać wykonane obsługi innych zdarzeń.

5. Podsumowanie

Biblioteka została zaimplementowania z użyciem platformy .NET. Ten wybór pozwolił na wykorzystanie wielu wbudowanych mechanizmów platformy, takich jak operacje na grupach czy kolejkach. Pozwoliło to na przejrzystą implementację. Pewne obawy rodził fakt, że tak wysoki poziom abstrakcji, jaki oferuje język C#, może przełożyć się negatywnie na wydajność procesu symulacji. Okazało się jednak, że zaimplementowana biblioteka ma minimalnie lepszą wydajność od rozwiązania CSL# i zbliżoną wydajność do swojego pierwowzoru stworzonego w C++.

Należy zaznaczyć, że istnieje już kilka dojrzałych i rozbudowanych pakietów programowych, wspierających modelowanie układów zdarzeń dyskretnych, wśród których nie sposób pominąć np. OMNET++. Jednak podobnie jak nie istnieją uniwersalne języki programowania, tak samo to co w jednym przypadku może być uważane za silną stronę narzędzia (np. uniwersalność), w innych sytuacjach może być balastem (złożoność nauki narzędzia). CSL# plasuje się tu jako proste i generyczne narzędzie, którego nauka nie powinna zająć przeciętnemu programiście więcej niż 60 minut.

Rozwiązanie pozwala w pełni przeprowadzać różne symulacje systemów masowej obsługi, m.in. realizację badania kolejek i obciążenia stanowisk obsługi, przy zdefiniowanej częstotliwości napływu zgłoszeń. Zaimplementowane klasy statystyk pozwalają na wygodne monitorowanie symulowanego systemu oraz prostą prezentację zgromadzonych danych. Biblioteka może być z powodzeniem używana dla prostych i złożonych układów.

BIBLIOGRAFIA

1. Kleiber M.: Modelowanie i Symulacja Komputerowa. Moda czy Naturalny Trend Rozwoju Nauki. Nauka, nr 4, 1999, s. 29÷41.
2. Skowronek M. (red.): Modelowanie Cyfrowe. Zadania. Wydawnictwo Politechniki Śląskiej, Gliwice 2012.
3. Wyciślik M.: Biblioteka CSL++ jako przykład narzędzia do symulacji układów zdarzeń dyskretnych. Materiały konferencyjne z Konferencji Symulacja Procesów Dynamicznych SPD-9, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1996.
4. Augustyn D. R.: CSL# – .NET Computer Simulation Language – środowisko do budowy modeli i symulacji układów zdarzeń dyskretnych. Sieci komputerowe. Nowe technologie, WKŁ, 2007, s. 291÷304.

Wpłynęło do Redakcji 16 stycznia 2013 r.

Abstract

In the article an implementation of library used to discrete events systems modeling based on CSL language was presented. The key point of this implementation is involving of .NET platform and C# language. Big popularity of these technologies makes that building of models is possible with minimal effort related to study of library API. In spite of “lightness” of the library it is still fully functional tool for discrete event systems modeling.

Adresy

Łukasz WYCIŚLIK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, lwycislik@polsl.pl.

Paweł WIEJAK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska.