Andrzej Sikora[1], Ewa Niewiadomska-Szynkiewicz[1,2]
[1]Politechnika Warszawska,
Instytut Automatyki i Informatyki Stosowanej
ul. Nowowiejska 15/19, 00-665 Warszawa
[2]Naukowa i Akademicka Sieć Komputerowa (NASK)
ul. Wąwozowa 18, 02-796 Warszawa,
email: *asikora@elka.pw.edu.pl*, *e-n-s@ia.pw.edu.pl*

ROZPROSZONA I RÓWNOLEGŁA SYMULACJA DYSKRETNA, METODY REALIZACJI I SYSTEM OPROGRAMOWANIA

Streszczenie – w niniejszym artykule koncentrujemy się na eksperymencie komputerowym wykonywanym przez równoległe lub rozproszone systemy komputerowe czyli komputery wieloprocesorowe lub sieci komputerowe złożone z wielu, często różnych, maszyn. Omawiamy rozwiązania stosowane do budowy symulatorów równoległych i rozproszonych. Szczególna uwaga jest poświęcona bibliotece ASim/Java wspomagającej tworzenie symulatorów systemów zdarzeń dyskretnych. Prezentowane są przykłady wykorzystania ASim/Java do symulacji wybranych systemów fizycznych.

1 Wstęp

Modelowanie matematyczne i symulacja komputerowa są od lat uznawane za skuteczne techniki wspomagające badania i zarządzanie systemami [1, 9]. Obecnie czas tworzenia modelu symulacyjnego bardzo się skrócił. Stało się to możliwe dzięki stosowaniu standardowych bibliotek modeli oraz przyjaznych użytkownikowi aplikacji ułatwiających ich projektowanie i wykonanie. Mimo tego realizacja eksperymentu symulacyjnego może być bardzo czasochłonna. Dzieje się tak, ponieważ cechą wspólną systemów dynamicznych jest ich złożoność.

Naturalnym sposobem przyspieszenia, a często także umożliwienia przeprowadzenia symulacji komputerowej wydaje się zwiększenie mocy obliczeniowej maszyn. Istotne jest odpowiednie wykorzystanie komputerów równoległych oraz sieci komputerowych. Realizowana za ich pomocą symulacja rozproszona odzwierciedla rzeczywistą strukturę systemu, często składającego się z wielu równolegle pracujących

podsystemów, oraz efektywnie wykorzystuje zasoby współczesnych maszyn [8, 16].

Od końca lat siedemdziesiątych XX wieku prowadzone są intensywne badania poświęcone obliczeniom realizowanym w sposób synchroniczny oraz asynchroniczny, w środowisku równoległym i rozproszonym, na różnych platformach sprzętowych. Opracowywano nowe protokoły komunikacji oraz odpowiednie narzędzia programistyczne [2, 8]. Stosując analizę formalną badana jest zbieżność algorytmów rozproszonych [3]. Jednocześnie w dziedzinie sprzętu komputerowego nastąpił gwałtowny postęp, którego efektem są maszyny równoległe o bardzo szybkich procesorach i szybkiej transmisji międzyprocesorowej [8, 9], stwarzające możliwości efektywnej realizacji obliczeń. Techniki równoległej i rozproszonej symulacji mają obecnie wiele praktycznych zastosowań i są intensywnie rozwijane. Główne problemy ograniczające ich wykorzystanie to uboga ilość narzędzi i języków służących do automatycznego tworzenia rozproszonych aplikacji, oraz problemy związane z realizacją i testowaniem programów równoległych i rozproszonych [8, 16].

Pierwsza część artykułu poświęcona jest zagadnieniom związanym z realizacją symulatorów równoległych i rozproszonych. Szczególna uwaga jest zwrócona na protokoły synchronizacji obliczeń oraz metodom efektywnego zarządzania pamięcią komputera. Druga część dotyczy biblioteki ASim/Java (*Asynchronous Simulations/Java*), wspomagającej tworzenie rozproszonych symulatorów zdarzeniowych. W ostatniej części prezentowane są przykłady wykorzystania ASim/Java do symulacji wybranych systemów fizycznych. Rozważane są: prosta linia produkcyjna oraz sieć komputerowa.

2 Rozproszone symulatory zdarzeń dyskretnych

W równoległej i rozproszonej symulacji dyskretnej (RSD) zakłada się, że poszczególne procesy logiczne *PL* symulujące procesy fizyczne *PF* są wykonywane na różnych maszynach lub procesorach i porozumiewają się między sobą poprzez wysyłanie komunikatów z danymi. Odpowiada to sytuacji rzeczywistej, w której procesy fizyczne komunikują się między sobą. Poszczególne *PL* wykonują swoje obliczenia niezależnie tak długo, jak długo niezależnie działają symulowane przez nie procesy fizyczne *PF*. Następnie *PL* są wstrzymywane w celu odebrania komunikatów z danymi od innych, powiązanych z nimi procesów logicznych, symulując w ten sposób interakcje zachodzące między fizycznymi podsystemami. Na podstawie sygnałów wejściowych generują sygnały wyjściowe i przesyłają wyniki do powiązanych procesów. Nie ma znaczenia na jakiej platformie jest wykonywany eksperyment symulacyjny, czy jest to sieć komputerów,

czy komputer wieloprocesorowy – koncepcja jest taka sama. Implementacja, w przypadku różnych platform może być oczywiście różna. Jest to zazwyczaj związane z koniecznością stosowania różnych narzędzi programowych zrównoleglenia obliczeń. Efektywność symulacji będzie również zależała od realizacji sprzętowej.

Realizacja symulacji w wersji równoległej lub rozproszonej wymaga rozwiązania trzech podstawowych problemów: dekompozycji zadania i przydziału zadań do procesorów, dostarczenia mechanizmów synchronizacji obliczeń oraz mechanizmów efektywnego zarządzania pamięcią komputera.

2.1 Synchronizacja obliczeń – pojęcia i definicje

Rozważmy symulator systemu realizowany przez P procesów logicznych, PL_p, p=1,...,P. Podobnie jak w symulacji sekwencyjnej mamy dwie skale czasowe:

czas symulacji t, tzn. czas wykonywanego w komputerze eksperymentu (czas w komputerze),

czas symulowany T, tj. czas biegu procesów w rzeczywistym systemie.

Wymiana danych między poszczególnymi procesami logicznymi jest realizowana poprzez wysyłanie w chwilach t komunikatów z odpowiadającymi im znacznikami symulowanego czasu T. W przypadku symulacji komputerowej, której zadaniem jest naśladowanie procesów przebiegających w świecie rzeczywistym możliwości zrównoleglenia obliczeń są ograniczone przez związki przyczynowo-skutkowe występujące w fizycznym systemie. Jeżeli w rzeczywistości zdarzenie *e1* poprzedza zdarzenie *e2*, to podczas symulacji obliczenia związane z *e1* muszą zostać wykonane przed obliczeniami związanymi z wystąpieniem zdarzenia *e2*. Realizacja obliczeń w kolejności niezgodnej z wystąpieniem zdarzeń powoduje wystąpienie błędu przyczynowo-skutkowego. R.M. Fujimoto w pracy [6] definiuje warunki gwarantujące spełnienie ograniczeń przyczynowo-skutkowych.

Definicja 1

Symulacja realizowana przez wiele komunikujących się wzajemnie procesów obliczeniowych spełnia ograniczenia przyczynowo-skutkowe wtedy i tylko wtedy, gdy kolejne zdarzenia przetwarzane przez każdy proces mają niemalejące znaczniki czasowe.

Uwzględnienie ograniczeń przyczynowo-skutkowych wymaga wprowadzenia dodatkowych mechanizmów gwarantujących poprawny przebieg symulacji. Zanim przejdziemy do omówienia poszczególnych rozwiązań zdefiniujemy wyko-rzystywane przez nie wielkości i pojęcia.

Realizując symulator zdarzeń dyskretnych tworzymy jego listę zdarzeń wewnętrznych *IEL* (*Internal Event List*) oraz bufory

przechowujące listy komunikatów otrzymanych od innych procesów *IQ* (*Input Queue*) – lista zdarzeń zewnętrznych – i wysyłanych do odbiorców *OQ* (*Output Queue*). Każdemu zdarzeniu oraz wszystkim komunikatom przypisujemy znacznik czasowy *T* odpowiadający chwili symulowanego czasu, w którym dane zdarzenie lub komunikat zachodzi. Przyjmujemy, że na wszystkich listach obowiązuje uszeregowanie zdarzeń zgodnie z ich znacznikami czasowymi. Każdy *PL* może mieć wiele buforów wejściowych i wyjściowych. Oznaczmy przez EIT_p (*Earliest Input Time*) najmniejszy znacznik czasowy z listy aktualnych komunikatów wejściowych. Analogicznie komunikaty z wynikami symulacji umieszczane są w buforach wyjściowych. Oznaczmy przez EOT_p (*Earliest Output Time*), najmniejszy znacznik czasowy z listy aktualnych komunikatów wyjściowych. W przypadku realizacji równoległej lub rozproszonej możemy mieć jedną, globalną listę zdarzeń wewnętrznych lub wiele lokalnych list IEL_p, p=1,...,P związanych z poszczególnymi procesami. Podobnie jest z buforami wejściowymi i wyjściowymi. Możemy mówić o zegarze globalnym *GVT* (*Global Virtual Time*) oraz zegarach lokalnych *LVT* (*Local Virtual Time*).

Definicja 2

W chwili symulacji t zegar lokalny LVT_p procesu PL_p, przyjmuje wartość znacznika czasowego aktualnie realizowanego przez ten proces zdarzenia wewnętrznego lub zewnętrznego, tj.

$$LVT_p(t) = \min\{\min_{e \in IEL_p(t)} T_e, EIT_p(t), EOT_p(t)\}.$$

Definicja 3

W chwili symulacji t zegar globalny GVT przyjmuje najmniejszą wartość z stanów lokalnych zegarów oraz znaczników komunikatów aktualnie krążących w sieci, $GVT(t) = \min\{\min_{p=1...P} LVT_p, \min_{m \in Q(t)} T_m\}$*, gdzie Q(t) jest globalną listą komunikatów aktualnie krążących w sieci.*

Na zakończenie zdefiniujemy jeszcze dwa okna czasowe wyznaczane lokalnie dla każdego *PL* i związane z najbliższą przyszłością i przeszłością. Są one wykorzystywane przez algorytmy synchronizacji obliczeń. Pierwsze, [*T*,*T*+*LA*], zdefiniowane w pracy [6], określa przedział czasu, w którym lokalnie możemy przewidzieć przyszłość.

Definicja 4

Niech T oznacza aktualną chwilę symulowanego czasu. Przedział czasu [T,T+LA], w którym jest zagwarantowane, że dany proces logiczny nie będzie przesyłał żadnych komunikatów do innych procesów, nazywamy oknem ,,lookahead".

W pracy [4] definiowany jest analogicznie przedział czasu wstecz [*T*-*LB*,*T*].

Definicja 5

Przedział czasu [*T-LB*,*T*], *w którym gwarantuje się, że przetworzenie zdarzeń z chwil wcześniejszych od aktualnego czasu symulowanego T, ale zawartych w tym oknie nie zmieni już wysłanych komunikatów, nazywamy oknem ,,lookback".*

2.2 Protokoły synchronizacji

Rozważamy dwa rodzaje symulacji rozproszonej: synchroniczną i asynchroniczną [9, 12, 16]. Symulacja synchroniczna to prosta adaptacja symulacji sekwencyjnej w równoległym lub rozproszonym środowisku sprzętowym. Cechą charakterystyczną jest globalna synchronizacja procesów obliczeniowych za pomocą globalnego zegara *GVT*. Synchronizacja globalna zabezpiecza przed wystąpieniem błędów przyczynowo-skutkowych oraz przed zakleszczeniem programów. O kolejności uruchomienia procesów logicznych decyduje kolejność wystąpienia symulowanych przez nie zjawisk. Możliwości zrównoleglenia są ograniczone.

W podejściu asynchronicznym następuje odejście od globalnej synchronizacji procesów. Dopuszcza się sytuacje, w których kolejność rozpoczęcia obliczeń przez symulatory jest inna niż kolejność wystąpienia zjawisk w fizycznym systemie. Tak więc, w danej chwili symulacji każdy z rozproszonych procesów może znajdować się w innym stanie, odpowiadającym innej chwili symulowanego czasu. Realizacja symulatora jest następująca:

- symulator składa się z wielu procesów logicznych *PL* równolegle, synchronicznie lub asynchro-nicznie, przetwarzających zdarzenia,
- każdy *PL* ma dostęp tylko do lokalnego wektora stanu, posiada lokalne, uporządkowane listy zdarzeń wewnętrznych i komunikatów wejścio-wych i wyjściowych,
- każdy *PL* ma lokalny zegar LVT_p; krok taktowania każdego z zegarów lokalnych może być różny,
- podczas eksperymentów wykorzystuje się specjalne protokoły synchronizacji lokalnej.

W ostatnich latach opracowano wiele protokołów gwarantujących poprawny przebieg symulacji. Wyróżnia się dwa podejścia: konserwatywne i optymistyczne.

Techniki konserwatywne

Idea technik konserwatywnych polega na unikaniu sytuacji wystąpienia błędu przyczynowo-skutkowego. Wspólną cechą metod należących do tej grupy jest przyjęcie założenia, że na każdym etapie obliczeń wykonywane są tylko procesy realizujące zdarzenia bezpieczne. Różnią się one sposobem w jaki te zdarzenia są

identyfikowane oraz mechanizmami unikania zakleszczeń programów. Chandy, Misra i Bryant są autorami jednego z pierwszych protokołów synchronizacji oznaczanego przez CMB [8]. Zakłada on, że w danej chwili czasowej żaden proces obliczeniowy nie zostanie uruchomiony, dopóki nie będzie gwarancji, że nie otrzyma od procesów przekazujących do niego komunikaty, danych z wcześniejszej chwili. Każdy *PL* symuluje działanie *PF* do chwili symulowanego czasu *T*, tylko wtedy, gdy zna on stan początkowy i wszystkie komunikaty, które *PF* otrzyma do chwili *T*. Zakładamy przy tym, że w przypadku sekwencji komunikatów <...(T_k, j, i, m_k), (T_{k+1}, j, i, m_{k+1}),...> są one wysyłane kolejno, zgodnie z ich uszeregowaniem według znaczników czasowych oraz odbierane w tej samej kolejności. Do zabezpieczenia się przed wystąpieniem zakleszczeń programów (np. w przypadku cyklicznych powiązań procesów) służy mechanizm wykorzystujący komunikaty z pustymi wiadomościami (*null messages*). Wysłanie przez PL_j do PL_i komunikatu (*T, j*, *i*, *null*) oznacza, że każdy następny komunikat (T_k, *j*, *i*, m_k) przesłany przez PL_j do PL_i będzie spełniał warunek $T_k>T$. W pewnych sytuacjach puste komunikaty nie są wystarczającym rozwiązaniem. Następna modyfikacja polega na wykorzystaniu przedziału *lookahead* (definicja 4). Na podstawie stanu procesu x_p, stanu zegara LVT_p i estymaty EIT_p wyznaczany jest najmniejszy znacznik komunikatu na wyjściu systemu, EOT_p.

Alternatywnym rozwiązaniem, do technik wykorzystujących *lookahead* jest mechanizm zakładający zdolność procesów do lokalnej modyfikacji przeszłości. Analizując procesy przebiegające w świecie rzeczywistym okazuje się, iż symulacja zjawisk niezgodnie z ich uszeregowaniem, w pewnych sytuacjach nie wpływa na końcowe rezultaty eksperymentu, zidentyfikowane błędy przyczynowo-skutkowe można wówczas zaniedbać. Mechanizm opisany w pracy [4] zakłada wyznaczenie przedziału *lookback* (definicja 5), w którym obsługa zdarzeń z chwil wcześniejszych od aktualnego stanu lokalnego zegara nie wpłynie na stany pozostałych procesów obliczeniowych.

Inny algorytm z grupy konserwatywnych, nazwijmy go WIN, polega na wyznaczaniu na osi czasu okna (przedziału czasu), w którym można bezpiecznie przetwarzać zdarzenia w sposób równoległy [10]. Zadaniem każdego procesu jest oszacowanie chwili $EOT_p(t)$, w której przewidywane jest wysłanie nowego komunikatu (przy założeniu, że do tego czasu nie pojawią się na jego liście żadne nowe zdarzenia). Następnie, globalnie wybierany jest najmniejszy z czasów wyznaczonych przez wszystkie procesy, *EOT*. W ten sposób definiowane jest okno [*T*, *EOT*) i wszystkie procesy mogą równolegle symulować zdarzenia, których znaczniki czasowe przyjmują wartości z tego przedziału. Zdarzenia z różnych okien są symulowane sekwencyjnie.

Podstawowe mankamenty mechanizmów konserwatywnych to ograniczone zrównoleglenie i zwiększenie obciążenia sieci komputerowej spowodowane przesyłaniem dodatkowych komunikatów.

Techniki optymistyczne

W metodach optymistycznych dopuszcza się możliwość wystąpienia błędu przyczynowo-skutkowego i podaje sposoby na usunięcie skutków spowodowanych wystąpieniem takiego błędu. Klasycznym reprezentantem technik optymistycznych jest mechanizm z zawijaniem czasu TW (*Time Warp*) [7]. Zasada działania jest następująca. Procesy przetwarzają kolejne zdarzenia z list, do momentu wystąpienia błędu przyczynowo-skutkowego, czyli otrzymania przez proces *p*-ty komunikatu (T^*, j, p, m^*) z czasem symulacji $T^*<LVT_p$. Obliczenia wykonane przez PL_p są cofane do chwili T^* i do jego odbiorców wysyłane są tzw. „antykomunikaty" unieważniające wszystkie wiadomości, których znaczniki czasowe były większe od chwili wystąpienia błędu. Jeden antykomunikat może wyzwolić całą lawinę antykomunikatów. Algorytm TW pozwala na lepsze wykorzystanie mocy obliczeniowej systemów równoległych i rozproszonych. Jego wadą, poza zwiększeniem obciążenia sieci spowodowanym przez wysyłane antykomunikaty, jest duże zapotrzebowanie na pamięć systemu komputerowego. Proponowane są różne modyfikacje algorytmu polegające na ograniczeniu optymizmu podczas procesu obliczeniowego i włączeniu pewnych mechanizmów rozważanych w algorytmach konserwatywnych. W podejściu znanym pod nazwą ,,przesuwane okno czasowe'', MTW (*Moving Time Window*) [15], stosuje się mechanizm TW, ale zakłada się ograniczenie równoległości do pewnych okien czasowych [T, $T+\Delta T$). Między oknami następuje synchronizacja. Tak więc cofanie obliczeń występuje tylko w obrębie okna, a o szybkości symulacji decydują najwolniejsze jednostki. W ten sposób zmniejsza się liczbę wysyłanych antykomunikatów i wymagań odnośnie zasobów komputera – w pamięci muszą być przechowywane tylko dane dotyczące zdarzeń z aktualnego okna czasowego. Głównym problemem jest oczywiście wyznaczenie długości okna, tak by osiągnąć największą efektywność obliczeń. Pokazują to wyniki badań zamieszczone w dalszej części artykułu.

Następną propozycja jest algorytm WTW (*Wrapped Time Warp*), w którym zdarzenia są grupowane w pakiety. Obsługa zdarzeń wchodzących w skład danego pakietu jest realizowana za pomocą TW. Poszczególne pakiety są przetwarzane z wykorzystaniem strategii konserwatywnej.

2.3 Zarządzanie pamięcią komputera

Podstawowa realizacja mechanizmu optymi-stycznego TW zakłada nieograniczone zasoby pamięci w systemie. Cofnięcie obliczeń do dowolnej chwili symulowanego czasu wymaga przechowywania w pamięci każdego procesora wektorów stanów oraz komunikatów wejściowych i wyjściowych. Modyfikacje TW pozwalają na pewne zmniejszenie wymagań na pamięć, ale nadal istotnym problemem pozostaje efektywne zarządzanie pamięcią komputera/komputerów. W literaturze proponowane są różne metody. Są to zarówno rozwiązania sprzętowe jak i programistyczne [12, 18]. Jednym z podejść programistycznych jest periodyczne sprawdzanie i zapisywanie stanu. Przyjmijmy, że w każdym procesorze zmienne stanu są sprawdzane i zapamiętywane co okres czasu $\Delta t > 1$. W pewnych przypadkach takie podejście skutkuje zwiększeniem nakładu obliczeń. Ustalenie odpowiedniego czasu repetycji Δt zachowywania stanu jest wynikiem kompromisu między istotnym obciążeniem pamięci komputera a nakładem obliczeń. Im dłuższy czas repetycji, tym mniejsze zapotrzebowanie na pamięć, ale większy nakład obliczeń i odwrotnie.

Podjęto liczne próby dynamicznego wyznaczania chwil sprawdzania stanu. W pracy [17] przedstawiony jest matematyczny model periodycznego zapisywania stanu, który został zastosowany w algorytmie ustalania optymalnych czasów repetycji Δt. Uwzględnia się w nim wpływ długości czasu repetycji na liczbę powtórzeń obliczeń wywołanych wystąpieniem błędów przyczynowo-skutkowych. Innym rozwiązaniem jest wyznaczenie czasu Δt jako kombinacji liniowej aproksymacji czasu repetycji Δt_1 obowiązującego do chwili bieżącej oraz aproksymacji czasu repetycji Δt_2 wyznaczonego dla bieżącego okna, tj. $\Delta t = \alpha_1 \Delta t_1 + \alpha_2 \Delta t_2$, gdzie α_1 i α_2 oznaczają heurystycznie dobrane wagi. Takie podejście pozwala na uwzględnienie na bieżąco zmian zachodzących w procesie obliczeniowym, wywołanych zmianami w środowisku obliczeniowym. Opracowywano również liczne algorytmy heurystyczne wyznaczania Δt. Przykładem może być metoda prezentowana w pracy [19], której celem jest doprowadzenie do zrównoważenia liczby chwil, w których jest zapisywany stan procesu i liczby dodatkowych powtórzeń obliczeń, spowodowanych koniecznością odtworzenia stanu symulacji. Algorytm jest wielokrotnie uruchamiany w trakcie jednego przebiegu eksperymentu. W pracy [20] zaprezentowano wyniki eksperymentów pokazujących wpływ wielkości czasu Δt na efektywność algorytmów optymistycznych i całkowity czas obliczeń. Uwzględniono różne topologie sprzętowe oraz różne warianty protokołu *Time Warp*.

Podejściem alternatywnym do periodycznego sprawdzania stanu procesu jest zapamiętywanie na bieżąco tylko części wektora stanu – tej która uległa zmianie w wyniku realizacji zdarzenia. Jest to tzw. podejście przyrostowe (ang. *state increment*). Zaletą takiego rozwiązania jest zmniejszenie zajętości pamięci i czasu potrzebnego na kopiowanie danych do pamięci, wadą wzrost nakładu obliczeń spowodowanych koniecznością odtworzenia stanu w przypadku powtarzania obliczeń. Omawiany mechanizm warto stosować do układów wysokiego rzędu, gdy w wyniku obsługi zdarzeń modyfikowana jest niewielka liczba zmiennych stanu, czas realizacji zdarzeń jest odpowiednio długi, a obsługa błędów przyczynowo-skutkowych nie wymaga cofania obliczeń o długi okres. Analiza porównawcza obu technik: periodycznej i przyrostowej znajduje się w pracy [21].

Do efektywnego zarządzania pamięcią często wykorzystuje się czas globalny *GVT*. Wszystkie dane z chwil T, $T<GVT$ (tzw. *fossil collection*) nie będą już wykorzystywane i mogą zostać usunięte z pamięci. Konieczne jest oczywiście utrzymanie globalnego procesu zajmującego się usuwaniem przestarzałych danych. W pracy [22] proponowane są różne, scentralizowane i rozproszone algorytmy wyznaczania stanu zegara globalnego *GVT*.

Wspomniane rozwiązania pozwalają na ograniczenie zapotrzebowania na pamięć, ale nie gwarantują poprawnego działania symulatora w sytuacji wyczerpania dostępnych zasobów. Konieczne jest wykorzystanie dodatkowych technik umożliwiających zwalnianie pamięci. Opracowano algorytmy odzyskiwania zasobów poprzez odsyłanie pewnych komunikatów do nadawców. Należą do nich:

- Protokół *Sendback* [7], w którym, gdy zaistnieje taka konieczność, odsyłany jest komunikat wejściowy o największym znaczniku czasowym oraz jego ulepszona wersja opracowana przez A. Gafni [23].
- Algorytm *artificial rollback* zaproponowany w pracy [24], w przypadku braku odpowiednich zasobów, zmusza najbardziej zaawansowany w obliczeniach proces do cofnięcia obliczeń.
- Mechanizm adaptacyjnego zarządzania pamięcią AMM (*Adaptive Memory Management*) [25] łączy ograniczony optymizm w przetwarzaniu zdarzeń i automatyczne dostosowanie wolnego obszaru pamięci, aby zrównoważyć koszty związane z powtarzaniem obliczeń i archiwizowaniem danych. Dostępne zasoby są dzielone na obszar przechowujący wyniki przetworzonych zdarzeń, obszar przeznaczony na przyszłe obiekty oraz obszar niewykorzystanej przestrzeni pamięci. Przebieg eksperymentu symulacyjnego jest determinowany przez przepływ pamięci między tymi obszarami.

3 Biblioteka ASIM/JAVA

Biblioteka ASim/Java (*Asynchronous Simulation/ Java*) [11] wspomaga tworzenie rozproszonych symulatorów zdarzeniowych. Pakiet ten umożliwia realizację symulacji rozproszonej i równoległej, dostarcza gotowe mechanizmy komunikacji, synchronizacji, zapisu wyników symulacji oraz podstawowe elementy symulatora zdarzeniowego. Dodatkowo został on wzbogacony o szereg nowoczesnych rozwiązań poprawiających jego wydajność i funkcjonalność.

Podstawowym przeznaczeniem aktualnej, trzeciej wersji oprogramowania jest symulacja systemów fizycznych o różnym stopniu złożoności, w szczególności symulacja sieci komputerowych. Możliwa jest realizacja symulatorów synchronicznych oraz asynchronicznych, w kilku wersjach różniących się sposobami synchronizacji. Oferowane są następujące protokoły: konserwatywne CMB i WIN, optymistyczny TW, hybrydowe MTW, WTW.

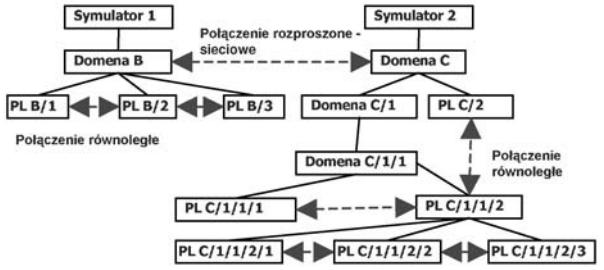
3.1 Architektura pakietu ASim/Java

Pakiet oprogramowania został zrealizowany w języku Java, przy wykorzystaniu schematu XML oraz wybranych technologii wspierających komunikację równoległą i rozproszoną.

Modułowa architektura systemu oraz proste i określone w dokumentacji [14] zasady rozwijania biblioteki ASim/Java pozwalają na rozszerzanie jej funkcjonalności o nowe elementy. Mogą do nich należeć np. kolejne dyscypliny kolejkowania, algorytmy synchronizacji obliczeń, protokoły komunikacji w architekturze rozproszonej oraz moduły obsługi zdarzeń o określonych typach.

Uniwersalność klas obsługi zdarzeń w procesach logicznych pozwala na wykorzystanie ich w innych symulatorach systemów tego samego typu. Tworzone są w ten sposób tzw. moduły narzędziowe dedykowane symulacji systemów związanych z konkretną dziedziną badań. Aktualnie autorzy rozwijają moduł dedykowany symulacji sieci komputerowej (np. klasy do budowy symulatorów protokołu TCP).

Bardzo często duże wymagania na wydajność symulatora, np. w przypadku modelowania złożonych systemów sieciowych, wymuszają wykorzystanie wielowątkowości lub obliczeń rozproszonych w sieci komputerowej [13]. ASim/Java w przezroczysty dla użytkownika sposób pozwala połączyć te dwie idee wykorzystując ogólny model symulatorów sfederowanych [5]. Wybór różnych technologii komunikacyjnych wprowadza również możliwość adaptacji symulatora do zmieniających się warunków obliczeniowych.



Rys. 1. Architektura przykładowego symulatora.

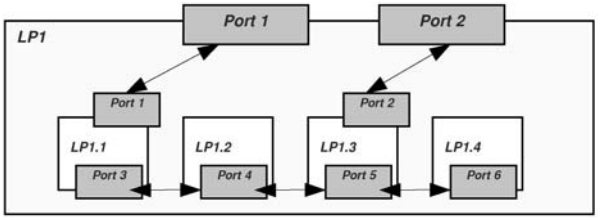
Zaprojektowana struktura danych umożliwia grupowanie procesów logicznych oraz wprowadza hierarchię pomiędzy nimi. Struktura ta oddaje hierarchiczną i/lub rozproszoną architekturę symulowanych systemów. Pozwala na dekompozycję symulowanego modelu w sposób naturalny i odpowiedni do liczby dostępnych procesów (rys. 1).

Podstawowe elementy aplikacji symulatora to:

- *Symulator* – zbiór domen, samodzielna aplikacja posiadająca możliwość współdziałania z innym symulatorem lub symulatorami w sieci [5].
- *Domena* – zbiór procesów logicznych i/lub domen; nie bierze udziału w symulacji i synchronizacji obliczeń.
- *Proces logiczny* – podstawowy element symulatora zgodnego z założeniami symulacji zdarzeniowej.

Duża skalowalność rozwiązań zastosowanych w implementacji ASim/Java umożliwia realizację obliczeń symulacyjnych na platformie Internetu. Wykorzystywane metody komunikacji symulatorów w sieci globalnej to technologia JMS (*Java Messaging Service*) oraz połączenia typu Peer2Peer.

Ciekawa i niewykorzystywana w innych środowiskach do symulacji zdarzeniowej jest możliwość zagnieżdżania procesów logicznych. Funkcjonalność ta została osiągnięta poprzez modyfikację algorytmów synchronizacji. Synchronizacja obliczeń odbywa się w tym przypadku tylko pomiędzy procesami logicznymi znajdującymi się na tym samym poziomie hierarchii symulatora (rys.2). Powyższe rozwiązanie daje m.in. możliwość ograniczenia otoczenia procesu logicznego do określonej grupy elementów symulatora (łatwe do wprowadzenia ograniczenia w komunikacji).



Rys. 2. Zagnieżdżanie procesów logicznych.

3.2 Model symulowanego systemu

Podstawowym zadaniem użytkownika korzystającego z biblioteki ASim/Java jest implementacja w języku Java klas odpowiedzialnych za obsługę zdarzeń w procesach logicznych. Klasy te powiązane są ze zdarzeniami określonego typu i można je użyć wielokrotnie w innych symulatorach (grupujemy je w tzw. moduły narzędziowe).

Następnym etapem jest zapis modelu symulowanego systemu w pliku XML o schemacie zgodnym z językiem ASimL opracowanym przez autorów [14]. Jest to język służący do modelowania systemów złożonych. Struktura języka ASimL ściśle odpowiada prezentowanej strukturze symulatora.

Język ASimL wykorzystywany jest w bibliotece ASim/Java do: definiowania struktury symulowanego systemu oraz dowolnej liczby parametrów opisujących system; definiowania wyszczególnionych parametrów konfiguracyjnych pakietu ASim/Java; zapisu kolejnych stanów symulatora i jego elementów podczas przebiegu symulacji; automatycznego uruchamiania symulatora na podstawie przygotowanego lub wygenerowanego w innej symulacji pliku XML.

Model systemu w tekstowym pliku XML jest jednocześnie plikiem konfiguracyjnym gotowego skompilowanego systemu. Korzystanie z języka ASimL nie wymaga specjalistycznej wiedzy informatycznej ani dedykowanego oprogramowania. Po przygotowaniu pliku, zgodnie z zasadami opisanymi w dokumentacji, symulator jest uruchamiany za pomocą jednego polecenia. W łatwy sposób można zmienić parametry modelu i uruchomić symulator ponownie. Pakiet ASim/Java posiada wbudowany dwukierunkowy interfejs pomiędzy aplikacją a plikiem XML, opartym na schemacie ASimL. Umożliwia to przerwanie wykonywanych obliczeń — stan symulatora jest zapisywany do pliku — i wznowienie ich w innym terminie.

Zestaw elementów języka AsimL to:

- simulator – symulator,
- domain – zbiór procesów logicznych/domen,

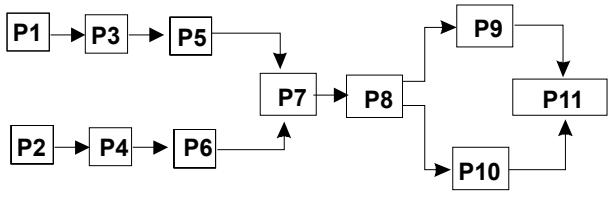
- logicalProcess – proces logiczny,
- link – połączenie pomiędzy dwoma portami,
- history – zbiór stanów procesu logicznego,
- historySnapshot – stan procesu logicznego w określonej chwili symulacji (zdjęcie),
- eventService – moduł obsługi zdarzenia,
- port – port komunikacyjny,
- queueType – kolejka zdarzeń,
- message – komunikat zawierający jedno lub wiele zdarzeń do obsługi,
- eventType – typ komunikatu/zdarzenia,
- parameterType – parametr dowolnego elementu symulatora,
- stateType – stan procesu logicznego,
- QueueingType – dyscyplina kolejkowania zdarzeń,
- CommunicationType – protokół komunikacji,
- EngineType – algorytm synchronizacji.

4 Badania porównawcze algorytmów synchronizacji

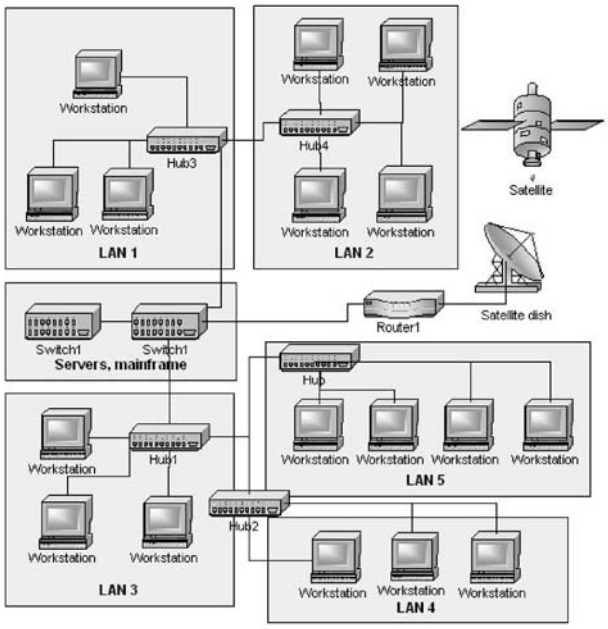
Korzystając z biblioteki ASim/Java przeprowadzono badania porównawcze różnych realizacji symulatorów rozproszonych. Badania wykonano w sieci złożonej z czterech komputerów z procesorami Celeron 433 MHz. Rozważano dwie przykładowe aplikacje: linię montażową złożoną z jedenastu stacji roboczych (rys.3) i sieć komputerową zbudowaną z czterech podsieci (rys. 4). Dla obu przypadków przeprowadzono dwie serie testów (warianty *A* i *B*) uwzględniające różne poziomy złożoności:

Linia montażowa: wariant *A* – każde źródło generuje 2 zdarzenia; *B* – źródła generują po 25 zdarzeń, różne czasy obsługi zdarzeń w obu wariantach.

Sieć komputerowa: wariant *A* – transmitowane są pakiety danych o niewielkich rozmiarach, *B* – znaczne rozmiary pakietów.



Rys. 3. Linia montażowa



Rys. 4. Sieć komputerów.

Zrealizowano kilka wersji rozproszonych symulatorów różniących się protokołami synchronizacji. Rozważano następujące algorytmy: dwa konserwatywne: CMB i WIN, optymistyczny TW i hybrydowy MTW. Celem eksperymentów była ocena szybkości protokołów i zbadanie wpływu wprowadzanych zabezpieczeń gwarantujących poprawność obliczeń na ich efektywność. Tabele 1, 2 i 3 prezentują wyniki testów uzyskane dla linii montażowej. Są to czasy symulacji w sekundach i liczby dodatkowych komunikatów: puste wiadomości w CMB, informacje wykorzystywane do wyznaczenia rozmiaru okien w WIN i MTW oraz antykomunikaty w TW i MTW (Tabele 1 i 2). W Tabeli 3 zamieszczono studium porównawcze różnych wariantów algorytmu MTW, w zależności od założonego rozmiaru okna czasowego, po którym następuje synchronizacja. Zwrócono uwagę na liczbę wysyłanych dodatkowych komunikatów oraz liczbę powtórzeń obliczeń.

W przypadku sieci komputerowej algorytm CMB okazał się bardzo mało wydajny. Ograniczono się do przedstawienia wyników działania dwóch algorytmów (Tabela 4): konserwatywnego WIN i hybrydowego MTW, dla dwóch różnych rozmiarów okien – jednej (MTW-1) i dziesięciu (MTW-10) jednostek symulowanego czasu.

Tabela. 1. Czas symulacji (linia montażowa)

Wariant	*CMB* [s]	*WIN* [s]	*TW* [s]	*MTW* [s]
A	247,83	25,81	34,55	20,93
B	316,86	88,43	52,34	41,25

Tabela. 2. Liczba dodadtkowych komunikatów

Wariant	*CMB*	*WIN*	*TW*	*MTW*
A	843	209	5	36
B	918	627	5	43

Tabela. 3. Różne okna czasowe (linia montażowa)

Rozmiar okna	Czas sym. [s]	Dodatkowe komunikaty	Powtórzenia obliczeń
5	54,87	134	0
10	45,09	71	1
20	41,25	36	1
30	44,87	33	2
40	63,82	34	0
50	59,87	23	1

Tabela. 4. Czas symulacji (sieć komputerowa)

Wariant	*WIN*	*MTW–1*	*MTW–10*
A	18186	18090	15742
B	72072	71097	63996

5 Podsumowanie

Zgodnie z przewidywaniami najlepsze rezultaty uzyskano stosując algorytm hybrydowy MTW. Przeprowadzone badania potwierdzają zamieszczone wcześniej uwagi dotyczące tego mechanizmu, iż jego efektywność silnie zależy od ustalonej długości okna, którego rozmiar powinien być dobierany do aplikacji i wykorzystywanego środowiska sprzętowego.

Najmniej wydajnym sposobem synchronizacji okazał się mechanizm CMB. Na czas obliczeń wpłynęła znaczna liczba wysyłanych komunikatów synchronizacyjnych oraz przestoje w pracy procesorów spowodowane oczekiwaniem na nowe dane. Należy jednak pamiętać o podstawowych zaletach tego rozwiązania, takich jak, prostota implementacji i stosunkowo małe zapotrzebowanie na zasoby komputera.

Wydaje się, iż w przypadku, gdy rozważamy systemy zbudowane z wielu podsystemów, w których często dochodzi do wzajemnych interakcji, nie jest rozsądne stosowanie technik optymistycznych. Może

to skutkować koniecznością częstego powtarzania obliczeń i zużyciem znacznych zasobów pamięci maszyny. Wskazane jest również w tym przypadku wykorzystywanie procesorów o podobnych mocach obliczeniowych. Obiecującym rozwiązaniem, jak pokazują to wyniki testów, jest połączenie podejść optymistycznych i konserwatywnych. Obecnie uwaga naukowców skupia się głównie na opracowywaniu nowych wersji wydajnych algorytmów hybrydowych.

Literatura

[1] J. Banks (ed.), „Handbook of Simulation", John Wiley & Sons, Inc., 1998.

[2] F. Berman, G. Fox, T. Hey (eds.), „Grid Computing: Making the Global Infrastructure a Reality", Wiley, 2003.

[3] D.P. Bertsekas, J.N. Tsitsiklis, „Some Aspects of Parallel and Distributed Iterative Algorithms - A Survey", *Automatica*, Vol.27, No.1, pp.3-21, 1990.

[4] G. Chen, B.K. Szymański, „Lookback: A New Way of Exploiting Parallelism in Discrete Event Simulation", *Proc. 16th Workshop on PDS*, IEEE CS Press, pp. 153-162, 2002.

[5] S. Ferenci, K. Perumalla. R.M. Fujimoto, „An Approach to Federating Parallel Simulators", *ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation* (PADS), Bologna, Italy, 2000.

[6] R.M. Fujimoto, „Parallel Discrete Event Simulation", *Communication of the ACM*, No. 33(10), pp. 30-53, 1990.

[7] D.A Jefferson, „Virtual Time", ACM *Transactions on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404-425, 1985.

[8] A. Karbowski, E. Niewiadomska-Szynkiewicz, (ed.) „Obliczenia równoległe i rozproszone", Oficyna Wydawnicza PW, Warszawa, 2001.

[9] N.A. Kheir (ed.), „Systems Modeling and Computer Simulation", Marcel Dekker, Inc., 1996.

[10] J. Misra, „Distributed Discrete-Event Simulation", *Computing Surveys*, Vol. 18, No.1, 1986.

[11] E. Niewiadomska-Szynkiewicz, A.Sikora, „ASimJava: a Java-based Library for Distributed Simulation'', *Journal of Telecommunication and Information Technology*, No 3, pp.12-17, 2004, NIT, Warszawa, 2004.

[12] D.M Nicol, R. Fujimoto, „Parallel Simulation Today", *Annals of Operations Research*, Vol. 53, pp. 249-285, 1994.

[13] A. Park, R. M. Fujimoto, K. S. Perumalla, „Efficient Synchronization of Large-scale Network Simulations", *Elsevier Computer Networks Journal,* 2004.

[14] A. Sikora, ,,Dokumentacja programistyczna ASim/Java", Raport IAiIS, PW, Warszawa, 2005.

[15] L.M Sokol, D.P. Briscoe, A.P. Wieland, „MTW: a Strategy for Scheduling Discrete Simulation Events for Concurrent Execution", *Proc. of the SCS Multiconference on Distributed Simulation*, pp. 34-42, 1988.

[16] B.P. Zeigler, H. Praehofer, T.G. Kim, „Theory of Modeling and Simulation", Academic Press, 2000.

[17] Y.B. Lin,, et al., „Selecting the Checkpoint Interval in Time Warp Simulation", Proc. of the 7th Workshop on Parallel and

[18] distributed Simulation, pp. 3-10, 1993.

[19] H.M. Soliman, „On the Selection of the State Saving Strategy in Time Warp Parallel Simulation", Transactions of The Society for Computer Simulation, Vol. 16, No 1, pp. 32-36, 1999.

[20] [P.A. Fleischmann, J., Wisley, „A Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators", Proc. of the 9th Workshop on Parallel and Distributed Simulation, pp. 50-58, 1995.

[21] [B.R. Preiss, Loucks, W.M., Macintyre, I.D., „Effects of the Checkpoint Interval on Time and Space in Time Warp", ACM Trans. on Modeling and Computer Simulation, Vol. 4, pp. 223-253, 1994.

[22] A.C Palaniswamy., Wilsey, P.A., „An Analytical Comparison of Periodic Checkpointing and Incremental State Saving", Proc. of the 7th Workshop on Parallel and Distributed Simulation, pp. 127-134, 1993.

[23] A. Ferscha, „Parallel and Distributed Simulation of Discrete Event Systems" in "Parallel and Ditributed Computing Handbook" (ed. A.Y.Zomaya), pp. 1003-1039, McGraw-Hill, New-York, 1995.

[24] A. Gafni, „Rollback Mechanisms for Optimistic Distributed Simulation", Proc. SCS Multiconference on Distributed Simulation, SCS, 1988.

[25] Y.B. Lin, Preiss, B.R., „Optimal Memory Management for Time Warp Parallel Simulation", ACM Transactions on Modeling and Computer Simulation, Vol. 1, No~4, pp.~283-307, 1991.

[26] S.R. Das, R.M. Fujimoto, „Adaptive Memory Management and Optimism Control in Time Warp", Proc. of the ACM Conference on Measurement and Modeling of Computer Systems, 1994.

DISTRIBUTED AND PARALLEL SIMULATION, METHODS AND SOFTWARE SYSTEM

Summary - The paper is concerned with simulation experiments carried out on a network of machines or parallel machines. Distributed simulation is proposed as an alternative to traditional sequential simulation which allows to reduce the execution time of a simulation program. The important issues associated with the implementation of parallel and distributed simulation are discussed. Different simulation techniques, including conservative and optimistic protocols for calculation process synchronization are reviewed. Particular attention is paid to the effectiveness of the proposed mechanisms. The paper describes a software system ASim/Java, a Java-based library that enables to do parallel and distributed simulations. The practical examples are provided to illustrate the operation of the presented software tool.