Symulacja komputerowa – wprowadzenie do DisSimLab

Dariusz Pierzchała

Wprowadzenie

Symulacja komputerowa na przestrzeni ponad 60 lat jej stosowania rozumiana była   
i definiowana na wiele sposobów – w rozdziale przyjmiemy, że jest to ilościowa   
i jakościowa metoda modelowania w języku formalnym oraz odwzorowania w programie komputerowym strukturalnych i behawioralnych cech systemów (rzeczywistych lub projektowanych), umożliwiająca eksperymentowanie z modelem (zamiast z systemem)   
i obserwowanie w nim procesów zachodzących w symulacyjnym czasie. Jest uznaną alternatywą dla eksperymentowania z rzeczywistym systemem lub jego fizycznym prototypem, dla metod analizy matematycznej, a także zastępuje nieformalne sposoby wnioskowania oparte na doświadczeniu i intuicji. W wyniku rozwoju systemów komputerowych i informatycznych przeszła ewolucję w zakresie technologii i algorytmów podobną do przeobrażeń odnotowanych w typowych systemach IT: od sekwencyjnego eksperymentu symulacyjnego realizowanego na jednej maszynie poprzez równoległy, sieciowy aż po rozproszony w wirtualnych zasobach tzw. chmur obliczeniowych   
(ang. *cloud computing*). Pierwsze modele symulacyjne zaliczane do metod „Monte Carlo” stosowano teoretycznie już w latach 30. w zespole Enrico Fermi, chociaż dopiero pod koniec lat 40-ych Stanislaw Ulam, John von Neumann i Nicholas Metropolis oparli to podejście na maszynach cyfrowych laboratorium w Los Alamos. Lata 60. to rozwój modeli sieciowych (Carl Adam Petri), metod analizy dynamiki systemów (Jay W. Forrester)   
a następnie języków i metod symulacji ciągłej i dyskretnej: różniczkowe (Lockheed – Digital Differential Analyzer Simulator DIDAS9), zdarzeniowe (Harry Markowitz, Bernard Hausner – SIMSCRIPT), interakcje procesów (Ole-Johan Dahl i Kristen Nygaard – Algol Simula67, Knuth i McNeley – Algol SOL), selektywnego wykonywania aktywności (John Buxton i John Laski – Fortran CSL) oraz trzy-fazowe (Keith Douglas Tocker – General Simulation Program GSP) [1]. Wspólna cechą niemal wszystkich tych narzędzi było podejście zorientowane na procedury programowe, w których zgodnie z pewnym algorytmem modyfikowane były wartości zmiennych opisujących symulowane zjawisko   
i tym samym system. Wyjątkiem był opracowany w 1967 roku w Oslo język Simula67, który poprzez swoją koncepcję klasy i obiektu odszedł od powszechnie stosowanego wówczas podejścia strukturalnego, stając się niezbywalnie protoplastą obiektowości.   
W ślad za tymi pionierskimi pomysłami poszli twórcy wielu języków ogólnego przeznaczenia, na bazie których powstały biblioteki do oprogramowania modeli symulacyjnych – zarówno stosowanych przez lata ciągłych i dyskretnych, jak też promowanych w ostatnim czasie modeli agentowych. Niezmiennie dynamiczny postęp   
w dziedzinie IT oraz wymienność technologii w okresach kilkuletnich sprawia, że budowanie nowych języków jest racjonalnie nieuzasadnione. Można dziś mówić wyłącznie o sporadycznie tworzonych konstrukcjach lingwistycznych dla prototypowych platform sprzętowych lub zastosowań laboratoryjnych (od ponad 10 lat nie powstały żadne istotne syntaktyki i semantyki dedykowanych stricte symulacji). Co wobec tego proponuje się   
w ich miejsce? Odpowiedzią na wciąż rosnące zapotrzebowanie symulacyjne są pakiety rozszerzające języki ogólnego przeznaczenia (np. C#, Java) o usługi symulacyjne (m.in. dynamiczne wyliczanie wartości zmiennych stanu, sterowanie upływem czasu, generowanie liczb losowych, rozpraszanie procesów w sieci). Dzięki takim praktykom każda ewolucja dotykająca języka implikuje również nowe możliwości w aplikacjach symulacyjnych – eksperymenty symulacyjne prowadzone są już nie tylko na pojedynczych stacjach komputerowych, nie tyko w sieciach lokalnych i gridach, ale również wg modelu przetwarzania danych opartym na usługach dostarczanych z chmur obliczeniowych.

Uzyskanie zwirtualizowanych środowisk symulacyjnych nie jest zadaniem trywialnym, pomimo że istnieje szereg standardów pozwalających na symulację rozproszoną – HLA (ang. *High Level Architecture* – IEEE 1516, STANAG 4603), DIS   
(ang. *Distributed Interactive Simulation* – IEEE P1278, STANAG 4482) oraz TENA   
(ang. *Test and Training Enabling Architecture* – otwarty nieformalny standard   
US DoD). Wciąż niezbędne są modele, algorytmy i techniczne rozwiązania z jednej strony   
w zakresie mechanizmów zarządzania dynamiką zmian stanów w symulowanym świecie,   
a z drugiej mechanizmów integrowania rozproszonych heterogenicznych środowisk symulacyjnych.

Przegląd technik symulacji komputerowej

Sposób badania lub naśladowania systemu z użyciem komputerów poprzez praktyczne eksperymenty z modelem symulacyjnym uzależniony jest od celu eksperymentu, skali oraz rodzaju zadań realizowanych przez system komputerowy. Można mówić o kilku zasadniczych formach eksperymentów symulacyjnych:

* symulacja konstruktywna (ang. *constructive simulation*) – oparta na formalnych modelach zagregowanych obiektów odwzorowanych w programach komputerowych obsługiwanych przez ludzi (decydentów, analityków, konsumentów wyników);
* rzeczywistość wirtualna (ang. *virtual reality*) – odtwarza z wykorzystaniem silników graficznych sceny przestrzenne a odwzorowanie człowieka jest elementem tych obrazów poprzez zastosowanie urządzeń ruchowych, wizualizacyjnych   
  i dźwiękowych;
* rzeczywistość rozszerzona (ang. *augmented reality*) – jest pomostem pomiędzy rzeczywistością a symulacją wirtualną poprzez nakładanie wirtualnych obiektów na rzeczywiste obrazy np. w okularach;
* symulacja rzeczywista (ang. *live simulation*) – to ćwiczenia z wykorzystaniem rzeczywistego sprzętu, trenażerów i urządzeń do sygnalizacji zdarzeń.

W konstrukcji modelu symulacyjnego należy zazwyczaj odwzorować składową statyczną   
i dynamiczną systemu podlegającego symulacji. Struktura statyczna opisuje obiekty wchodzące w skład modelowanego systemu wraz z ich cechami oraz relacjami łączącymi je w związki. Jest ona wystarczająco tylko w przypadku symulacji statycznej, pomijającej zjawisko upływu czasu. W symulacji dynamicznej niezbędne jest zdefiniowanie modeli formalnych i algorytmów programowych wyznaczania kolejnych stanów obiektów   
w modelowanym systemie. Przyjmuje się, że w procesie symulacji wyznaczane będą wartości cech opisujących system w określonych chwilach lub przedziałach czasu symulacyjnego, które to zadawane będą deterministycznie (symulacja deterministyczna) albo losowo/pseudolosowo (symulacja stochastyczna). Każda modelowana cecha powinna być niezależna (nie istnieją dla niej cechy redefiniujące) a zbiór cech powinien być zupełny i mieć kompletny opis z punktu widzenia celu modelowania. Ma to znaczenie na wszystkich etapach tworzenia oprogramowania symulacyjnego: warunkuje formalną poprawność modelu a następnie wpływa na efektywność realizacji programu i adekwatność wyników. Jeżeli do modelowania systemu zastosuje się paradygmat obiektowy, to do każdej istotnej mierzalnej cechy (fizycznej lub abstrakcyjnej) systemu należy przypisać atrybut z wybranej klasy obiektów odwzorowującej fragment modelowanego świata:

* – zbiór symulowanych obiektów klasy *c* identyfikowanych przez *id* o wartości niepowtarzalnej w zbiorze obiektów;
* *CO* – niepusty zbiór klas modelowanych obiektów;
* *Ac* – niepusty zbiór atrybutów określonych dla klasy obiektów ;
* – zbiór dopuszczalnych wartości atrybutu klasy obiektów .

W praktyce zbiory *CO* oraz *Ac* przedstawia się jako zbiory numerów odpowiednio modelowanych klas obiektów i ich atrybutów. W dowolnej chwili *t* czasu symulacyjnego każda składowa stanu systemu, a tym samym wyróżniona cecha systemu, widziana będzie jako czwórka uporządkowana: *<o, a, v, t>*. Zatem stan modelowanego systemu *S*(*t*) będzie zbiorem utworzonym przez wszystkie atrybuty wszystkich obiektów istniejących w chwili symulacyjnej *t*. W symulacji dynamicznej wartości atrybutów *a∈Ac* wyznaczane mogą być jako skutek: przekształcenia, funkcji zmiany stanu, zdefiniowanych reguł lub wprost danych wprowadzonych z zewnątrz (np. od użytkownika). Innym wymiarem dynamiki systemu jest możliwość zmiany struktury obiektów systemu przez ich tworzenie lub usuwanie oraz interakcje między nimi. Kluczowe pojęcie czasu symulacyjnego wprowadza się z uwagi na ograniczone możliwości odwzorowania równoległości zmian stanów na maszynie sekwencyjnej, a także dla umożliwienia realizacji eksperymentu   
w tempie innym niż astronomiczny. Zatem, czas symulacyjny *t*∈*T* jest zmienną (najczęściej rzeczywistą, nieujemną *T⊂R+*∪{*0*}), której zmiany wartości wyrażone w umownych jednostkach mają charakter niemalejący. Przyjmuje się jedną uwspólnioną jednostkę czasu symulacyjnego w całym modelowanym systemie. Jest ona umowna, a jej wybór zależy od wielu czynników, np. dynamiki zmian stanu systemu, ziarnistości modelowania (przyjmuje się np. że w reżimie czasu astronomicznego jednostką jest sekunda). Niech *ti*, *tj*, *tk* będą chwilami (momentami) czasu o zerowej długości. Zbiór *T* chwil, w których zachodzą zmiany stanu może być:

* podzbiorem punktów w pewnym przedziale nieujemnych liczb rzeczywistych takim, że (∀*ti*,*tk*)(∃*tj*)(*ti<tj<tk*) – czyli dla każdych dwóch chwil istnieje moment między nimi – oznacza to ciągły upływ czasu;
* albo podzbiorem przeliczalnym takim, że (∃*ti*,*tk*)(*¬*∃*tj)* (*ti<tj<tk*) – czyli istnieją dwa takie momenty, że nie istnieje żaden moment między nimi – oznacza to dyskretny upływ czasu.

Przyjmuje się, że czas symulacyjny jest lewostronnie skończony: (∀*t0*) (*¬*∃*ti*) (*ti< t0*), zatem przeszłość ograniczona jest do pewnego momentu t0, zwanego chwilą początkową. Jeśli określona jest chwila końcowa t∞ czasu symulacyjnego taka, że: (∀*t∞*) *(¬*∃*ti*)(*ti> t∞*),   
to należy rozumieć, iż wraz z jej osiągnięciem następuje zakończenie eksperymentu,   
co stanowić może jeden z warunków stopu eksperymentu. W symulacji sekwencyjnej (najczęściej jednoprocesowej) obowiązuje liniowy wariant przepływu i uporządkowania czasu: (∀*ti*,*tj*)(*ti<tj∨ti>tj∨ti = tj*). W symulacji rozproszonej możliwe są również inne warianty: rozgałęziony i równoległy [2].

W praktyce stosowane jest wiele metod odwzorowania dynamiki w modelach   
i programach symulacyjnych, dając podstawę do ich klasyfikacji z uwagi na sposób odwzorowania upływu czasu symulacyjnego oraz modelowania zmian stanów:

* ciągłe w czasie – do opisu zjawiska upływu czasu (nie dotyczy to opisu zmian stanu) wykorzystywane są funkcje ciągłe lub quasi-ciągłe (dyskretyzacja ciągłości),   
  np.: metody Eulera, Rungego-Kutty, Adamsa Bashfortha, Adamsa Moultona lub Geara. Specyficzną podklasę stanowią metody analizy dynamiki systemów (ang. *system dynamics*), przybliżające w sposób całościowy (a nie fragmentaryczny) systemy wielopoziomowe oraz wieloelementowe, poprzez specjalnie wprowadzone pojęcia: zasobu (ang. *stock*), przepływu (ang. *flow*) oraz sprzężenia zwrotnego (ang. *feedback loop*);
* dyskretne w czasie – w opisie zjawiska upływu czasu stosuje się funkcje dyskretne, domeną czasu jest skończony zbiór dyskretnych chwil:
  + oparte na stałym kroku – wartość czasu przyrasta ze stałą wartością   
    Δ*t*: *ti+1=ti+*Δ*t*, po czym wykonywane są obliczenia dla nowego stanu;
  + oparte na zdarzeniach – wartość czasu wynika z chwili wystąpienia kolejnego zdarzenia, czyli zmiany wartości wybranych atrybutów stanu, albo zmiany struktury obiektów systemu;
  + oparte na czynnościach i aktywnościach.

W dalszej części uwaga skupiona będzie na metodach dyskretnych. Przyjmiemy, że dla każdego symulowanego obiektu klasy *CO* określone są: wektor atrybutów stanu oraz, zależnie od przyjętego podejścia, opis zdarzeń (ang. *event*) lub aktywności (ang. *activity*)   
i czynności (ang. *action*).

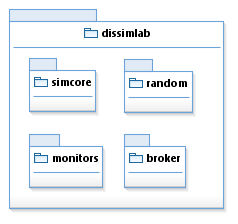
W *pierwszym podejściu – zorientowanym stricte na zdarzenia* – pod pojęciem zdarzenia *e* ze skończonego zbioru zdarzeń *E* rozumiana jest zaplanowana algorytmicznie zmiana stanu obiektu/systemu w określonej chwili czasu symulacyjnego: *e=< t*, *feS>, e*∈*E, t*∈*T*. Funkcja zmiany stanu *feS: T* x *S* → *S* wyznacza stan, w jakim znajdzie się system   
w chwili *t* po zajściu zdarzenia *e*. Mówi się, że zdarzenia opisane różnymi funkcjami zmiany stanu to zdarzenia różnych klas. W symulacji dyskretnej w sensie czasu przyjmuje się następujące uproszczenie modelowe – stan systemu nie ulega zmianie do czasu realizacji kolejnego zdarzenia: *S(t)=Si* dla *t*∈[*ti*, *ti+1*)⊂*T*. W modelu symulacyjnym można również zdefiniować odwzorowanie *S* → *2E* określające dla każdego stanu zestaw zdarzeń, które mogą w tym stanie wystąpić. Zdarzenia muszą zachodzić w nieodwracalnej kolejności od przeszłości, przez teraźniejszość do przyszłości – w tym celu definiuje się kolekcję zdarzeń *LE*⊂*E* (zwaną „kalendarzem zdarzeń”), przechowującą uporządkowane chronologicznie zdarzenia, które oczekują na realizację. W kalendarzu może znajdować się wiele zdarzeń tej samej klasy, jak również wiele zdarzeń może być zaplanowane na jedną chwilę *t*. Wynikające z tego zjawisko jednoczesności zdarzeń wymaga rozwiązań takich, aby kolejność realizacji zdarzeń w symulacji sekwencyjnej nie miała znaczenia dla uzyskanych wyników (np. w problemie kolejności zdarzeń: wybuch, uszkodzenie po zderzeniu okrętu z miną). Przyjmuje się, że ciąg zdarzeń uszeregowanych chronologicznie w czasie symulacyjnym to proces symulacyjny. Zależnie od uwarunkowań może być traktowany jako proces deterministyczny lub stochastyczny (realizacją jest szereg czasowy z dziedziną czasu symulacyjnego). W ogólności może to być wielowymiarowy proces stochastyczny, dla którego poszczególne elementy stanu wektora opisują różne parametry systemu w tej samej chwili symulacyjnej. Bieżący czas symulacyjny równy jest wartości czasu pierwszego w kalendarzu zdarzenia: *t\* = min* (*t: ei=*< *t, feS*>∈*LE, i=*1..2*ExT*). Przed rozpoczęciem symulacji musi być wplanowane co najmniej jedno zdarzenie, zaś podtrzymanie symulacji wymaga umieszczania w kalendarzu nowych zdarzeń. Zdarzenie początkowe nie może być uwarunkowane realizacją innych zdarzeń, chociaż w ogólności pomiędzy zdarzeniami mogą istnieć związki przyczynowo-skutkowe (ang. *causal relation*). Wg [3] zdarzenie *ej* na chwilę *tj* może być utworzone jako skutek poprzedzającego je zdarzenia *ei* (tzw. przyczyny) z chwili *ti*<*tj* wtedy i tylko wtedy, gdy *ej* jest dedukowalne z *ei* za pomocą reguł wynikających z modelu i znanych w chwili symulacyjnej *tj*. Tak rozumiane zależności przyczynowo-skutkowe są bezwarunkowe, gdy zdarzenie *ej* zawsze zajdzie po zaistnieniu zdarzenia *ei* – jest to warunek konieczny i dostateczny dla *ej*, albo warunkowe, gdy zdarzenie *ej* zajdzie po zaistnieniu nie tylko zdarzenia *ei* – warunek konieczny – lecz również dodatkowych warunków dopełniających warunek konieczny.

W *podejściu zorientowanym na aktywności* wprowadzone są pojęcia *czynności   
i aktywności* obiektów symulacyjnych do modelowania składowej behawioralnej systemu. *Czynność obiektu* jest to niepodzielna akcja trwająca przez pewien niezerowy czas, inicjowana i kończąca się zmianą stanu. Biorąc pod uwagę przyjętą definicję zdarzenia, czynność można definiować za pomocą pary zdarzeń *<ei, ej>*, zachodzących w chwilach *ti*,*tj*∈*T*, *ti*<*tj*. Zdarzenie *ei* będzie inicjującym, natomiast *ej* kończącym czynność, której czas trwania równy jest Δ*ti*,*j=tj-ti*. Czas trwania wynika z odległości czasowej między zdarzeniami definiującymi czynność, a biorąc pod uwagę ich zależności przyczynowo-  
-skutkowe zakończenie czynności może wystąpić bezwarunkowo (trwa nieprzerwanie przez zadany czas) lub warunkowo (przerwanie przed planowanym czasem, zależność od warunkowego semafora lub od zakończenia innej czynności). Zmiana stanu systemu następuje w zdarzeniu po zakończeniu jednej czynności i przed rozpoczęciem kolejnej. Pojawia się tu pojęcie pochodne do czynności: *aktywność obiektu* – definiowane przez chronologicznie uporządkowany, w kolejnych po sobie przedziałach czasu, ciąg czynności: {<*ei* , *ej*>,<*ej* , *ek*>,*…*}. Zdarzenie końcowe pewnej czynności jest jednocześnie zdarzeniem początkowym dla kolejnej czynności (o ile następna istnieje). Szczególną czynnością jest *NOP*, czyli w istocie bezczynność, wprowadzona dla odwzorowania bierności obiektu   
w pewnym przedziale czasowym. Czynności realizowane w rzeczywistości równolegle przez obiekt muszą zostać przydzielone do różnych aktywności, co skutkuje realizacją   
w wielowątkowym modelu programowania współbieżnego. Warto podkreślić, że przyjęte powyżej założenia umożliwiają praktyczne przejście pomiędzy modelami obydwu podejść (stricte zdarzeniowe oraz oparte na aktywnościach), a ponadto mają zastosowanie   
w relatywnie najnowszej klasie modeli – zorientowanej na *agenty programowe*.

Symulacja zdarzeniowa w pakiecie DisSimLab

Praktyczną realizacją w języku Java jest *pakiet DisSimLab* zaimplementowany z wykorzystaniem uznanych bibliotek odpowiedzialnych za przesyłanie komunikatów (Dispatcher) oraz otwartą architekturę wtyczek (PlugIn). Główne składowe pakietu zaprezentowano na uproszczonym diagramie klas z rysunku 1:

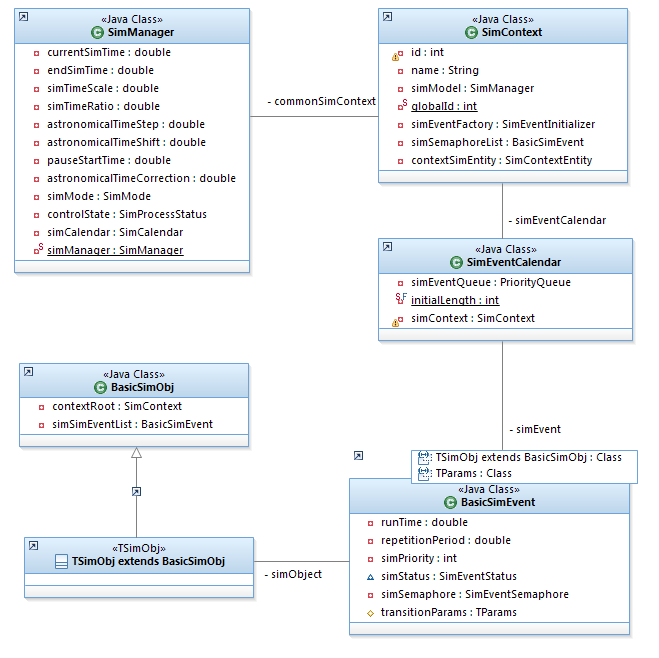
* simcore – pakiet grupujący klasy odpowiedzialne za zarządzanie eksperymentem, upływ czasu symulacyjnego, zdarzenia i zmiany stanu;
* broker – klasy umożliwiające przesyłanie komunikatów pomiędzy obiektami symulacyjnymi (agentami, środowiskiem);
* random – pakiet klas generatora liczb (pseudo)losowych;
* monitors – klasy odpowiedzialne za monitorowanie, gromadzenie i udostępnienie do analizy statystycznej szeregów czasowych pochodzących ze wskazanych zmiennych programowych.



Rys. 1. Diagram składowych pakietu DisSimLab.

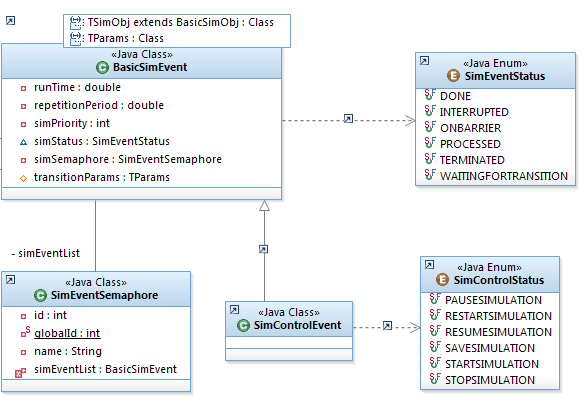
Każdy obiekt symulacyjny *o*∈*O* klasy *c*∈*CO* musi dziedziczyć z abstrakcyjnej klasy BasicSimObject. W jego zestawie atrybutów *Ac* przechowywane są wartości zmiennych stanu. Zdarzenia *e=*<*t, feS*> skojarzone ze zmianami stanu powstają w obiektach dziedziczących z generycznej klasy BasicSimEvent. Główne metody tej klasy to: stateChange() – odpowiadająca funkcji zmiany stanu *feS*, terminate() – usuwająca zdarzenie bez realizacji, interrupt() – usuwająca zdarzenie dodatkowo realizując zdefiniowane instrukcje, reschedule() – przeplanowująca zdarzenie na inny czas, simTime() – aktualny czas symulacyjny. Atrybutami są parametry sterujące: planowany czas i priorytet wykonania zmiany stanu (runSimTime, priority) oraz opcjonalny krok czasowy (repetitionPeriod) – dla modelu deterministycznego ze stałym krokiem.

W zależności od użytego konstruktora powoływany jest obiekt zdarzenia jednego   
z rodzajów: dla zmiany stanu zaplanowanej wyłącznie do kalendarza (na chwilę runSimTime lub warunkowo na semaforze SimEventSemaphore), dla zmiany stanu planowanej w kalendarzu i rejestrowanej w pośredniku komunikatów Dispatcher   
(po wykonaniu zmiany stanu rozsyłane jest dodatkowo powiadomienie do obiektów zarejestrowanych w Dispatcher i nasłuchujących zdarzenia określonej klasy). Przyjęty model rozgłaszania zdarzeń umożliwia informowanie wszystkich obiektów zainteresowanych określoną zmianą stanu o jej wystąpieniu (np. w celu utworzenia zdarzeń zależnych).



Rys. 2. Diagram klas pakietu DisSimLab/simcore.

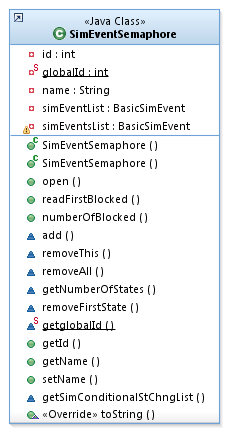
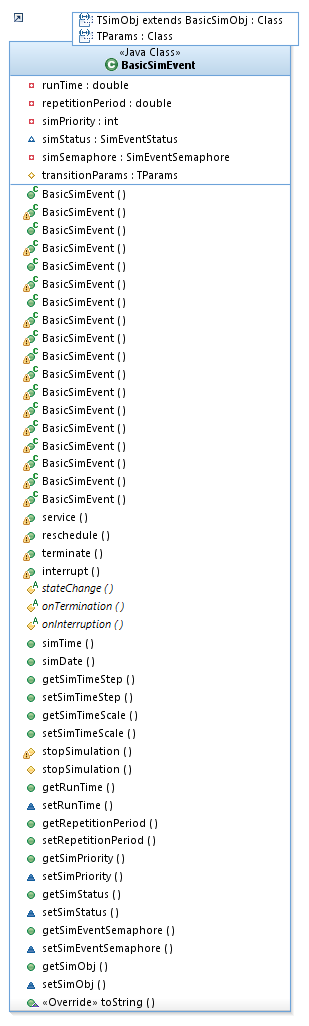
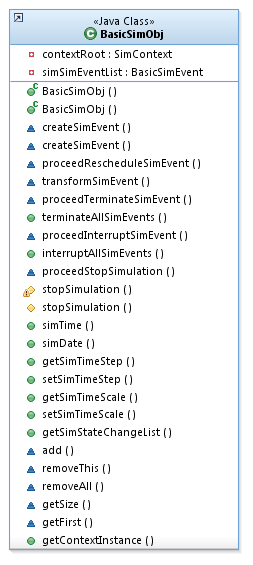
Struktura kalendarza zdarzeń uporządkowanych wg wartości runSimTime   
i priority wkomponowana jest w obiekt kontekstu (SimContext). DisSimLab umożliwia włączenie do jednego programu symulacyjnego wielu niezależnie opracowanych kontekstów modelujących na swój sposób różne części rzeczywistego systemu. Każdy kontekst musi realizować interfejsy wtyczki (Plugin, ISimCont). Komunikacja pomiędzy obiektami pochodzącymi z różnych kontekstów opiera się na wspólnym dla nich pośredniku komunikatów, ulokowanym w obiekcie zarządzającym (SimModel).



Rys. 3. Diagram klas pakietu DisSimLab/simcore.

Obiekt zarządzający, powoływany nadrzędnie wobec kontekstów wg wzorca singleton, jest kompleksowo odpowiedzialny za: sterowanie startem, stopem i pauzami w eksperymencie oraz trybem upływu czasu (jak najszybciej, astronomiczny, skalowany względem astronomicznego). Ponadto, utrzymuje informacje o wszystkich kontekstach (zatem pośrednio o obiektach i kalendarzach zdarzeń), wskazuje konkretne zdarzenie do wykonania w każdej chwili symulacyjnej i ustala na tej podstawie bieżącą wartość czasu symulacyjnego, obowiązującą dla wszystkich kontekstów oraz obiektów.

Uzupełnieniem podstawowych funkcji pakietu są klasy odpowiedzialne za: generowanie liczb i procesów pseudolosowych wg zadanych rozkładów (RandomGen), monitorowanie zmian i gromadzenie wartości przyjmowanych przez wskazane zmienne stanu w różnych chwilach symulacyjnych (MonitoredVar) a także za obliczenia różnych statystyk na zgromadzonych przez monitory szeregach czasowych (Statistics).



Rys. 4. Diagram klas pakietu DisSimLab/simcore.

Wybrane klasy pakietu DisSimLab

**abstract class BasicSimEvent<TSimObj extends BasicSimObj, TParams> implements INotificationEvent**

// Add to common simObj in the default context for the whole model (when stateChange is very general/common to model) - without any "in" parameters

public BasicSimEvent(double delay) throws SimControlException

// Add with delay to common simObj in the default context for the whole model

public BasicSimEvent(double delay, TParams params) throws SimControlException

// Add with priority to common simObj in the default context for the whole model

public BasicSimEvent(TParams params, int priority) throws SimControlException

// Add with delay and priority to common simObj in the default context for the whole model

public BasicSimEvent(double delay, TParams params, int priority) throws SimControlException

//Add immediately to entity TSimObj in the default context - without any "in" parameters

public BasicSimEvent(TSimObj entity) throws SimControlException

//Add with delay to entity TSimObj in the default context - without any "in" parameters

public BasicSimEvent(TSimObj entity, double delay) throws SimControlException

//Add with delay and parameters to entity TSimObj in the default context

public BasicSimEvent(TSimObj entity, double delay, TParams params) throws SimControlException

//Add with delay and priority to entity TSimObj in the default context

public BasicSimEvent(TSimObj entity, double delay, int priority) throws SimControlException

//Add with delay, parameters and priority to entity TSimObj in the default context

public BasicSimEvent(TSimObj entity, double delay, TParams params, int priority) throws SimControlException

// Add with parameters to common simObj in the default context for the whole model and synchronize/block with barrier

public BasicSimEvent(SimEventSemaphore barrier, TParams params) throws SimControlException

// Add in the default context for the whole model and synchronize/block with barrier

public BasicSimEvent(TSimObj entity, SimEventSemaphore barrier) throws SimControlException

// Add with parameters in the default context for the whole model and synchronize/block with barrier

public BasicSimEvent(TSimObj entity, SimEventSemaphore barrier, TParams params) throws SimControlException

// Add with delay and parameters in the default context for the whole model and synchronize/block with barrier

public BasicSimEvent(TParams params, double period) throws SimControlException

//Add as periodic event with parameters to entity TSimObj in the default context

public BasicSimEvent(TSimObj entity, TParams params, double period) throws SimControlException

//Add as periodic event with parameters and priority to common simObj in the default context

public BasicSimEvent(TParams params, double period, int priority) throws SimControlException

//Add as periodic event with parameters and priority to entity TSimObj in the default context

public BasicSimEvent(TSimObj entity, TParams params, double period, int priority) throws SimControlException

//Change runtime to new value (current time plus delay)

public boolean reschedule(double delay) throws SimControlException

//Remove event from Calendar and call onTermination()

public boolean terminate() throws SimControlException

//Remove event from Calendar and call onInterruption ()

public boolean interrupt() throws SimControlException

//Change state according to the code

protected abstract void stateChange() throws SimControlException

//Current simulation time (double value)

public double simTime()

//Set repetition time and make event periodic

public void setRepetitionPeriod(double repetitionPeriod)

//Set “run and publish” event

public void setPublishable(boolean publishable)