УДК 004.94

О.А. Змеев, А.В. Приступа

РАЗРАБОТКА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММНОГО КОМПЛЕКСА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ

Рассмотрены существующие средства разработки имитационных моделей. Предложен подход к построению объектно-ориентированного программного комплекса имитационного моделирования систем массового обслуживания. В качестве основного требования, предъявляемого к разрабатываемому комплексу, рассматривается обеспечение наибольшей гибкости при моделировании.

На сегодняшний день существует множество систем, специализирующихся на построении моделей только одного типа (производственных, экономических, систем управления запасами и т.п.). При этом разработчик модели зачастую может управлять только параметрами модели, но не ее поведением.

С другой стороны, существуют языки имитационного моделирования, которые обеспечивают наибольшую гибкость моделирования, однако процесс разработки моделей занимает длительное время, и эти модели труднее модифицировать и использовать.

И, наконец, существуют универсальные среды (Arena, AutoMod, AweSim, Extend, ProModel, Simplex3, Taylor ED), позволяющие разрабатывать различные модели, предоставляющие средства для написания кода в процессе разработки моделей, однако их стоимость составляет тысячи долларов [2].

В этой связи разработка системы имитационного моделирования на основе новых технологий программирования представляется авторам довольно актуальной задачей, тем более что многие особенности в реализации систем имитационного моделирования СМО позволяют достаточно успешно применять для их разработки методы объектно-ориентированного анализа и проектирования [1]. Основным требованием, предъявляемым к такой системе, будем считать предоставление пользователю наибольшей гибкости при разработке моделей.

СТРУКТУРА МОДЕЛИ

Рассмотрим, что представляет собой написание кода в процессе разработки модели и как оно может повысить гибкость системы имитационного моделирования.

Первой и наиболее простой составляющей можно считать возможность задавать и изменять глобальные переменные (например, время моделирования) и параметры объектов (например, параметры закона распределения, максимальная длина очереди и т.д.) перед запуском модели. Следующей составляющей будем считать возможность использования математических функций и выражений. И, наконец, третьей, и самой главной (в смысле гибкости), составляющей является возможность изменять значения переменных в момент изменения состояния имитационной модели [3].

При этом первые две возможности реализованы во всех системах, а вот третьей – создавать определяемые пользователем переменные и управлять изменением их значений в определенных состояниях модели – так не хватает во многих средах имитационного моделирования. В этой связи целесообразно рассмотреть следующий подход к разработке имитационных моделей.

Модель состоит из нескольких компонентов. Каждый из компонентов может использоваться либо повторно (т.е. берется уже существующий компонент, разработчик изменяет необходимые параметры и включает его в модель), либо может быть разработан с нуля. В обоих случаях необходимо приложение, позволяющее работать с этими компонентами. В качестве примера на рис.1 приведено графическое представление компонента и основных его элементов.

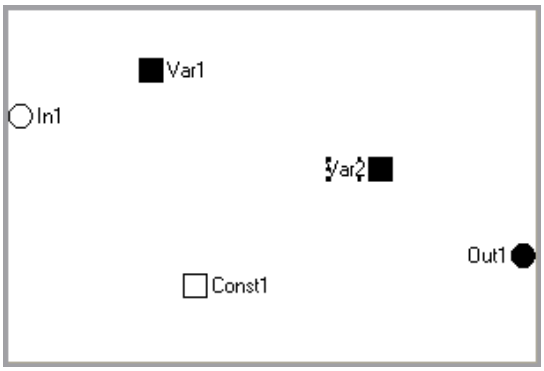

Рис. 1. Вид модельного компонента

Каждый компонент может включать в себя константы, переменные и порты. Константы и переменные могут принимать значения различных типов, основные из которых – целочисленный (`Integer`), вещественный (`Double`) и логический (`Boolean`). У переменных есть еще один дополнительный тип – времени (`Time`), который используется для расчета времени наступления следующего события. Значения констант не меняются в процессе выполнения имитационной модели. Значения переменных изменяются при наступлении событий по законам, определяемым разработчиком модели. Для удобства редактирования свойств элементов, входящих в компонент, желательно реализовать дерево и инспектор элементов (рис. 2 и 3) подобно тому, как это сделано в современных средах разработки (Delphi, Visual C++, Visual Basic и т.п.).

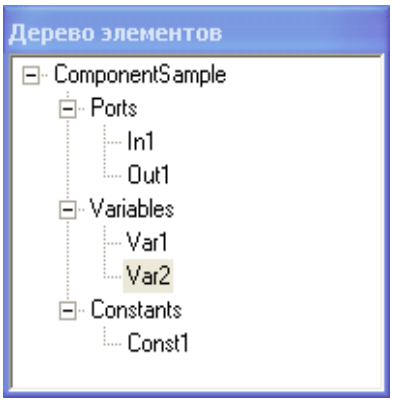

Рис. 2. Дерево элементов

Рассмотрим свойства переменных:

– `BehaviorType` – способ изменения переменной (непрерывно/дискретно);

– DataType – тип переменной (Integer/Double/Boolean/Time);

– Name – имя переменной;

– Value – начальное значение переменной.

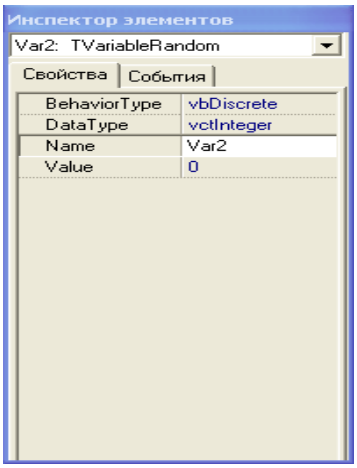

Рис. 3. Инспектор элементов

Рассмотрим свойства переменных:

– BehaviorType – способ изменения переменной (непрерывно/дискретно);

DataType – тип переменной (Integer/Double/Boolean/Time);

– Name – имя переменной;

– Value – начальное значение переменной.

Свойства констант:

– DataType – тип константы (Integer/Double/Boolean/Time);

– Name – имя константы;

– Value – значение константы.

Свойства портов: – Name – имя порта.

СОБЫТИЯ

Основой имитационного моделирования являются события – нечто, происходящее в системе, изменяющее ее состояние. В общем случае событие можно записать в следующей форме: On <Условие> do <Тело>, но все же целесообразней различать события двух видов [5]:

1) событие типа DoOnce, выполняющееся в тот момент, когда условие становится истинным;

2) событие типа DoAlways, выполняющееся всегда, когда условие истинно.

Для получения списка событий необходимо выбрать вкладку «События» в инспекторе элементов. Каждое событие имеет свое название, события можно добавлять, редактировать и удалять.

Для создания и редактирования событий необходима отдельная форма, позволяющая выбирать тип события (DoOnce или DoAlways), записывать комплексные условия, истинность которых влечет собой наступление события, и тело события в виде последовательности операторов, записанных на внутреннем языке системы.

Заметим, что в теле события можно также использовать случайные переменные. Например, допустима конструкция Var2:=Var2+R, при этом переменная R, в отличие от переменной состояния, обладает дополнительными свойствами Distribution и Parameters, которые позволяют задать для нее закон распределения с требуемыми параметрами. В этом случае каждое обращение к этой переменной будет вызывать запуск соответствующей процедуры, генерирующей случайные числа в соответствии с заданным законом.

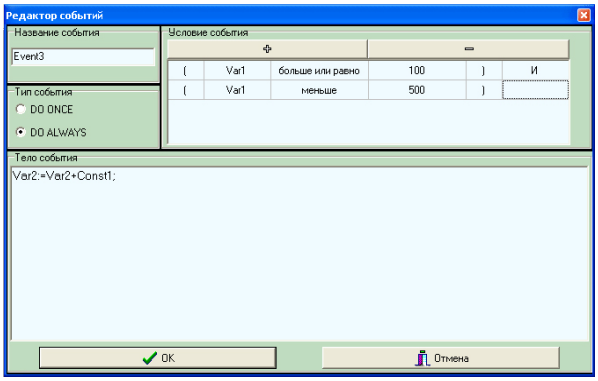

Рис. 4. Редактор событий

ПРИМЕР 1. СОБЫТИЯ В ПРОСТЕЙШЕЙ МОДЕЛИ СМО

Для начала определим необходимые переменные.

Переменные состояния:

– TArrive (Time) – время прибытия заявки;

– TService (Time) – время окончания обслуживания заявки;

– EntitiesInServer (Integer) – число заявок в устройстве обслуживания;

– EntitiesInQueue (Integer) – число заявок в очереди.

Случайные переменные:

– ArriveTime (Double) – интервал между заявками;

– ServiceTime (Double) – время обслуживания.

Теперь запишем события.

Событие 1. Поступление заявки;

```
On (T >= TArrive) Do Always
EntitiesInQueue: = EntitiesInQueue + 1;
TArrive: = T + ArriveTime;
```

Событие 2. Начало обслуживания:

```
On (EntitiesInServer = 0) and
                (EntitiesInQueue  >  0)
Do Always
EntitiesInQueue: = EntitiesInQueue - 1;
EntitiesInServer:  =  EntitiesInServer
+ 1;
TService: = T + ServiceTime;
```

Событие 3. Окончание обслуживания:

```
On (T >= TService) and
      (EntitiesInServer  =  1)  Do  Al-
ways
EntitiesInServer: = 0;
```

Можно заметить, что переменная T ни в одном из событий не изменяется. Но вспомним, что переменные TArrive и TService имеют тип Time. Это означает, что T после выполнения всех событий (или невыполнения, если условие окажется ложным) будет всегда принимать меньшее из этих значений (метод модельных событий).

ПРИМЕР 2.
МОДЕЛЬ СТРАХОВОЙ КОМПАНИИ

Для начала определим необходимые константы и переменные.

Константы:

– `N` – верхний предел количества клиентов в компании.

Переменные состояния:

– `TArrive (Time)` – время прибытия нового клиента;

– `TInsuranceEvent (Time)` – время наступления страхового случая;

– `TLeave (Time)` – время выхода клиента из компании;

– `K (Integer)` – количество клиентов в компании;

– `S (Double)` – капитал компании.

Случайные переменные:

– `AT (Double)` – чистая интенсивность процесса прибытия клиентов;

– `IET (Double)` – чистая интенсивность наступления страховых случаев;

– `LT (Double)` – чистая интенсивность выхода клиентов из компании;

– `SPlus (Double)` – размер «скачка вверх» капитала;

– `SMinus (Double)` – размер «скачка вниз» капитала.

Теперь запишем события.

Событие 1. Прибытие нового клиента.

```
On (T >= TArrive) Do Always
K: = K + 1;
S: = S + SPlus;
TArrive: = T + F (AT, N, K);
```

Событие 2. Наступление страхового случая.

```
On (T >= TInsuranceEvent) and
      (S > 0) Do Always
S: = S – SMinus;
TInsuranceEvent: = T + F (IET, K);
```

Событие 3. Выход клиента из компании.

```
On (T >= TLeave) Do Always
K: = K – 1;
TLeave: = T + F (LT, K);
```

ЗАКЛЮЧЕНИЕ

Объектно-ориентированная технология позволяет строить довольно гибкие системы. Это касается и построения систем имитационного моделирования. В данной статье изложен возможный подход к построению объектно-ориентированного комплекса имитационного моделирования систем массового обслуживания.

ЛИТЕРАТУРА

1. *Змеев О.А., Лезарев А.В.* Шаблон объектного проектирования для реализации функциональности процесса моделирования в имитационных моделях систем массового обслуживания // Вестник Томского гос. ун-та. 2002. № 275. С. 108–111.
2. *Змеев О.А., Приступа А.В.* Классификация коммерческих систем имитационного моделирования // Материалы Всерос. научно-практич. конф. «Наука и практика: диалоги нового века». Ч. 3. Информационные технологии и математическое моделирование (14 ноября 2003 г., г. Анжеро-Судженск). Томск: Твердыня, 2003. С. 93–95.
3. *Лоу А.М., Кельтон В.Д.* Имитационное моделирование. Классика CS. 3-е изд. СПб.: Питер, 2004. 848 с.
4. *Рыжиков Ю.И.* Имитационное моделирование. Теория и технологии. СПб.: Корона принт, 2004. 384 с.
5. *Шмидт Б.* Искусство моделирования и имитации. М.: Франтэра, 2003. 550 с.

Статья представлена кафедрой прикладной информатики факультета информатики Томского государственного университета, поступила в научную редакцию «Информатика» 21 мая 2004 г.